MASTER MATHÉMATIQUES VISION APPRENTISSAGE

ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

MASTER THESIS

Learning information cascades in social networks

Author:
Igor COLIN

Supervisor:
Nidhi HEGDE

September 16, 2013

Abstract

Social networks are increasingly used for diffusion of information. Characterizing the influence networks used by content dissemination can provide for rich recommendation systems. However, some challenges araise from this problem. First, the path of information dissemination is unknown: only the infection times are observed. Some assumptions have to be made in order to recover the path for each dissemination. Also, even if some algorithms exist for recovering the influence network, they are all quite slow when applied on a several hundred nodes network. Moreover, this work assumes that all content follow the same path. In reality, influence strongly depends on the type of content.Characterizing influence paths based on content type can significantly improve recommendations based on content type and social context.

Keywords: social networks, inference, optimization, graph, cascades of information.

Acknowledgement

I would like to thank Nidhi Hegde for her enriching supervision and her availability. I also thank Francis Bach for his help regarding the theoretical part of the internship.

Finally, I am grateful to the Technicolor Lab. It was a real pleasure to work there.

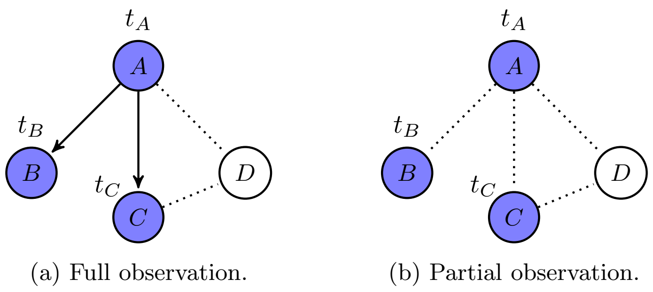
Contents

List of Figures

1 Introduction

Social networks are increasingly used for information diffusion. When an information is disseminated over such a network, one can often notice that some paths are more frequent than others. Even if formal connections between users (friends, *followers*, etc.) are well defined and easily retrievable, influences between members of a given social network are more complex and cannot *a priori* be obtained. Being able to establish influences would constitute a very useful dataset for a recommandation tool for instance.

First, the notion of influence must be defined. When user A shares a content, it becomes accessible for each of its relations. If one or several relations share in turn this content, we then consider that A has influenced these users. For a given content, like a video or an article, one can access the time of diffusion of the content for each user – the timestamp of sharing; one cannot know the real path of diffusion. Figure 1 illustrates this: even if user A influenced users B and C, the only data available are the times of sharing t_A, t_B and t_C.



(a) Full observation. (b) Partial observation.

Figure 1: Incomplete observation illustration. Dashed edges represent formal connexions (e.g. *follower*). Arrows represent influence.

Well-built models are needed in order to address the issue of incomplete observations and to infer the network of influence in an acceptable time.

Several algorithms able to estimate the network of influence already exist. They use several techniques such as greedy methods [GRLK10] or reformulation as convex optimization problem [RBS11, ML10]. and are all based on one model for information dissemination: the independent cascade model [KKT03].

However, these algorithms are still quite slow to infer large networks, even 200 nodes networks. In this report, we will focus on studying available algorithms and optimizing their computation time. We will also try to propose a method for inferring networks in the case of a multi-type information dissemination – e.g. cat videos would not have the same network of influence as political news.

2 Cascade models

Cascade models are a way to represent the diffusion of information in a network. They are widely used for content diffusion modelling. Although many types of cascade models exist, they are based on the same general idea.

Let us consider a discrete time representation and a directed network $G = (V, E)$. If a node $i \in V$ is infected at time t, a child j of i will be infected at time $t+1$ with a certain probability p_{ij}. This probability is not necessarily the same for all children and can depend on many factors, such as the global infection state or the duration of the infection. If at time $t = 0$, only one node is infected, such diffusion will result in a tree of infected nodes.

The model can easily be extended to a continuous time representation: if a node i is infected at time t_i, it will still infect on of its child j with some probability p_{ij}. Moreover, if the node j is infected, its infection time t_j will also be a random variable: the infection delay $(t_j - t_i)$ is often distributed according to a power or an exponential law.

2.1 Linear threshold model

Although the linear threshold model is not exactly a cascade model, it is a general model of diffusion that can be connected to a cascade model, as explained in [KKT03] and [MR07]. Each node is assumed to have a activation level, symbolized by a threshold $t_j \in (0, 1]$, usually distributed uniformly at random. Also, each node $i \in V$ has an influence level on his children w_{ij}. At a given time t, a node j will be infected if the global influence of his infected parents is superior than the threshold, i.e.:

$$\sum_{\substack{i \in \mathcal{P}(j) \\ i \text{ infected}}} w_{ij} \geq t_j.$$

This model is typically used in marketing policy: the more persons around you buy a product, the more likely you are to buy it too. It has not been used for recovering a network of influence; its main application is to find the most influencial nodes of an already specified graph – in order to find the appropriate persons to give free samples to.

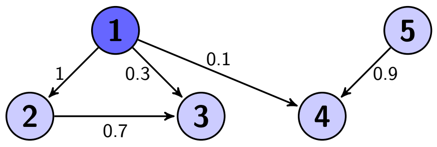
Figure 2 represents an example of a diffusion using the threshold model. Here, we consider that each node has an activation level of 0.8. Influence is shown on edges.

2.2 Independent cascade model

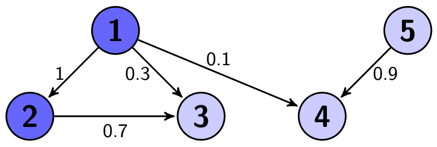
The independent cascade model is very popular and is first presented in [KKT03]. Here, the main assumption is that the probability of infection does not depend on the global infection state. For a node $i \in V$ and j such that $(i, j) \in E$, the probability p_{ij} is a function of the form:

$$p_{ij} = f_{ij}(t_i, t),$$

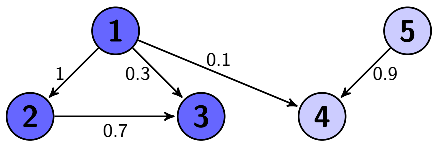
where t_i corresponds to the time when i got infected. Such a definition is valid because it can be shown that the infection distribution does not depend on the order of infection. Typically, if a node j has 2 parents, i_1 and i_2, if i_1 tries to



(a) $t = 0$



(b) $t = 1$



(c) $t \geq 2$

Figure 2: Example of a diffusion with the threshold model.

infect j first, we will obtain the same infection probability on j than if i_2 tries to infect first.

One possible application of this model is news propagation. When a website publishes a piece of news, some other websites that are influenced by the original one will then publish the new information too, and so on. The independent cascade model is the most common for network inferring [RBS11, GRLK10, ML10].

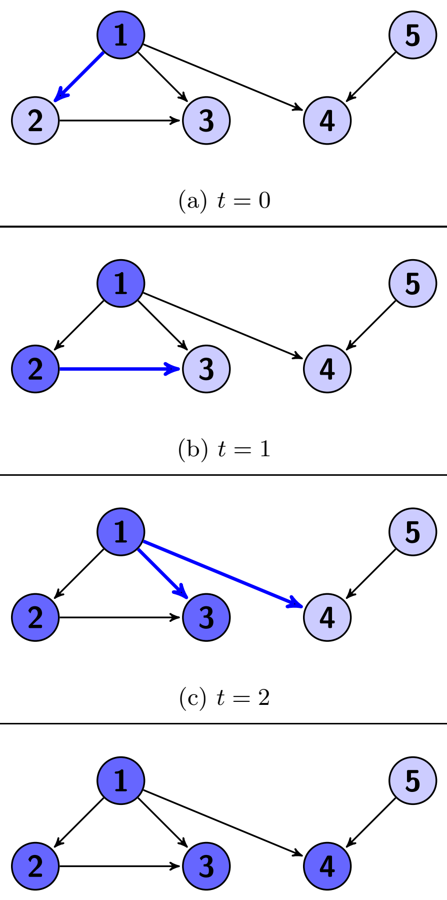
Figure 3 represents an example of a diffusion with the independent cascade model. We see that node 1 infects node 2 at $t = 1$ but only tries to infect node 3 at $t = 3$. This is due to the fact that even if a node has influence on another one, it only has some probability to propagate the infection at each time. Moreover, node 3 is already infected at $t = 3$ so the action of node 1 has no consequence.

2.3 Decreasing cascade model

The last model we will present is more general than independent cascade model and is presented in [KKT05]. Here, the infection probabilities can depend on the child other parents. At a time t, let us consider an infected node i and an uninfected child j. Let S be the set of nodes that have already tried to infect j. The probability p_{ij} will then be a function of the form:

$$p_{ij} = f_{ij}(t_i, t, S).$$

Moreover, another assumption of this model is that for $S \subseteq T \subseteq V$ and any



(a) $t = 0$

(b) $t = 1$

(c) $t = 2$

(d) $t \geq 3$

Figure 3: Example of a diffusion with the independent cascade model.

(i, j, t_i, t), we have:

$$f_{ij}(t_i, t, S) \geq f_{ij}(t_i, t, T).$$

In other words, at a time t, the more a node j has resisted to infection attempts, the harder it will be for a parent i to infect it. The last assumption this model needs is the order independence assumption. As it is not a direct consequence, we have to assume that the order in which the parents of a node try to infect it has no influence on the final infection probability distribution.

The decreasing cascade model is an interesting variant of the independent cascade model. However, we will not use it for network recovering and will mainly focus on the independent cascade model.

3 Inferring algorithms

Based on a certain information diffusion model, we want to recover a network of influence by using concrete observations of propagations. However, the observations will be incomplete: we will not observe the path of diffusion, but only the times of infection of each node. An observation can be considered as a vector $\mathbf{t} \in \mathbb{R}^{|V|}$, where t_i corresponds to the time of infection of node i. If i is never infected we will use the convention $t_i = +\infty$.

3.1 The NetInf algorithm

The NetInf algorithm is presented in [GRLK10]. Several assumptions have to be done on the information diffusion model. First, we considerer a continuous time model. Also, an infected node i can transmit its infection to one of his child with a probability β, where β is uniform over the network. This is a heavy assumption: we assume that every influences are the same. Then, if a node effectively transmits its infection, the infection delay (the difference between i infection time and its child infection time) follows either a power-law or an exponential-law, also uniform over the network. Finally, a *super-node* m is introduced: this node is connected to every node by an ϵ-edge in order to take account for possible jumps in diffusion of information.

With these assumptions, one can express the likelihood of a given cascade (a vector of times) for all possible tree T and deduce the likelihood of a cascade for a graph G. However, this problem is NP-hard to solve exactly because one would have to compute likelihood for every possible tree over all possible networks. Thus, the NetInf method proposes a simplified approach.

First, instead of considering every possible spanning tree of a graph G in the likelihood of a cascade c over this particular graph, only the most likely propagation tree is considered. Thus each cascade c is associated to one tree T_c and the improvement in likelihood obtained by adding one edge of this tree can be processed. Finally, for a given size k, the graph is construct by selecting the k edges bringing the highest improvement in the likelihood, using a greedy algorithm.

This algorithm is simple to implement, quite fast and can be easily parallelized. However, it makes some heavy assumptions over the homogeneity of influences.

3.2 The ConNie algorithm

The ConNie algorithm is presented in [ML10] and presents some similarity with the NetInf algorithm. In this case too, it is considered that time is continuous and that diffusion of information follows an independent cascade model. The infection delay is considered to follow either an exponential or a power-law too.

Let $\mathcal{C}$ be a set of cascades of information and $\mathbf{t}^c$ be a vector of times corresponding to cascade $c \in \mathcal{C}$. The aim of this algorithm is to maximize the likelihood:

$$L(A; \mathcal{C}) = \prod_{c \in \mathcal{C}} f(A; c) g(A; c) \tag{1}$$

where $A = (\alpha_{ij})_{i,j}$ represents the weights of the edges – typically the parameter of a power-law, and:

$$
\begin{aligned}
f(A;c) &= \prod_{i;t_i^c<+\infty} P_{A*i}(i \text{ infected at } t_i^c | \mathbf{t}_{-i}) \\
g(A;c) &= \prod_{i;t_i^c=+\infty} P_{A*i}(i \text{ never infected} | \mathbf{t}_{-i}).
\end{aligned}
$$

The maximization problem (1) can be transposed to a convex optimization problem with a simple change of variable. Finally, a term is added to the objective function in order to control the sparsity of the inferred graph. This term simply corresponds to a l_1-penalty of the edges weights. The final problem can be expressed as $|V|$ convex problems:

$$
\begin{aligned}
\min \quad & \sum_{k:t_i^{c_k}<+\infty} -\hat{\gamma}_{c_k} - \sum_{k:t_i^{c_k}=+\infty} \sum_{j:t_j^{c_k}<+\infty} \hat{B}_{ji} + \rho \sum_j \exp\left(-\hat{B}_{ij}\right) \\
\text{s.t. } & \hat{B}_{ji} \leq 0, \, \forall j \\
& \hat{\gamma}_{c_k} \leq 0, \, \forall k \\
& \log \left[\exp \hat{\gamma}_{c_k} + \prod_{j:t_j^{c_k} \leq t_i^{c_k}} \left(1 - w_j^{c_k} + w_j^{c_k} \exp \hat{B}_{ji}\right) \right] \leq 0, \, \forall k,
\end{aligned}
$$

for each $i \in V$, where w correspond to the infection delay distribution, ρ is the l_1-penalty coefficient and

$$
\begin{aligned}
\hat{B}_{ji} &= \log(1 - A_{ji}) \\
\hat{\gamma}_{c_k} &= \log\left(1 - \prod_{j:(j,i)\in E} \left(1 - w(t_i^{c_k} - t_j^{c_k}) A_{ji}\right)\right).
\end{aligned}
$$

3.3 The NetRate algorithm

NetRate is an algorithm developped in [RBS11]. The main idea behind NetRate is also to reformulate the problem as a convex optimization problem. Although it does not contain an explicit sparsity constraint, the likelihood expression forces some sparsity over the edges of the network. From now on, we will mainly focus on this algorithm.

Let $G = (V, E)$ be a network of influence. For $(i, j) \in E$, let us denote f_{ij} as the likelihood of node j being infected by node i. That is, if node i is infected at time t_i, the likelihood of node j being infected between times T_1 and T_2 is given by:

$$\int_{T_1}^{T_2} f_{ij}(t|t_i)\mathrm{d}t.$$

The survival function of a node j to a node i is defined as the probability that node i still has not infected node j at a given time:

$$S_{ij}(t|t_i) = 1 - \mathbb{P}(t_j < t|t_i) = 1 - \int_0^t f_{ij}(t'|t_i)\mathrm{d}t'.$$

Given these definitions, we can denote that:

$$f_{ij}(t|t_i) = -S'_{ij}(t|t_i) \tag{2}$$

Finally, the hazard rate is defined as the instantaneous probability of infection:

$$H_{ij}(t|t_i) = \frac{f_{ij}(t|t_i)}{S_{ij}(t|t_i)}.$$

Following the reasoning of [ABGG08] and using (2), H_{ij} can be rewritten:

$$H_{ij}(t|t_i) = -\frac{S'_{ij}(t|t_i)}{S_{ij}(t|t_i)},$$

or

$$H_{ij}(t|t_i) = -\big(\log S(t|t_i)\big)'.$$

Using the fact that $S_{ij}(t_i|t_i) = 1$, we finally obtain:

$$S(t|t_i) = \exp\left(-\int_{t_i}^{t} H_{ij}(t'|t_i)\mathrm{d}t'\right). \tag{3}$$

Therefore, one only needs to know H_{ij} in order to recover S_{ij} and f_{ij}, by using respectively (3) and (2). We will mainly focus on the study of the hazard rate functions for this reason.

We can now establish the expression of the likelihood of a given observation. Let $\mathbf{t} = (t_i)_{i\in V}$ be the observations vector associated to a cascade of information spreading over the network G. For a node i, t_i corresponds to the time of infection of node i during this cascade. We use the convention $t_k = +\infty$ if node k is never infected by the cascade; also, we assume that the cascade is observed over a finite duration T – starting from time 0. A node infected beyond that threshold will be considered as non-infected.

Let j be a node such that $t_j \leq T$. The likelihood of node j being infected by a node i, given observations $\mathbf{t}$, can then be written as a combination of the likelihood of infection caused by node i and the survival from node j to every other already infected node:

$$L(i \text{ infected } j \text{ at } t_j|t_{-j}) = f_{ij}(t_j|t_i) \times \prod_{k\neq i: t_k<t_j} S_{kj}(t_j|t_k)$$

The survival term brings some sparsity to the edges of the network. Given two distincts nodes of the network, k and l, the events $\{k \text{ infected } j \text{ first}\}$ and $\{l \text{ infected } j \text{ first}\}$ are disjoints. We can then write the likelihood of infection of node j as the sum over every potential parent:

$$L(t_j|t_{-j}) = \sum_{i:t_i<t_j} f_{ij}(t_i|t_j) \times \prod_{k\neq i: t_k<t_j} S_{kj}(t_j|t_k),$$

or

$$L(t_j|t_{-j}) = \sum_{i:t_i<t_j} H_{ij}(t_j|t_i) \times \prod_{k: t_k<t_j} S_{kj}(t_j|t_k).$$

In order to use as much information as possible from the vector of observations, we will also consider the likelihood associated to nodes that did not get infected – or that got infected after the threshold T. Thus, given a node j such that $t_j > T$, we write the likelihood associated to the survival of this node as follow:

$$L(t_j|t_{-j}) = L(j \text{ non-infected until } T) = \prod_{i:t_i \leq T} S_{ij}(T|t_i).$$

Finally, as we make the assumption that infections are conditionally independent given the infected parents, we can write the global likelihood associated to the observations vector $\mathbf{t}$:

$$L(\mathbf{t};\mathbf{H}) = \left(\prod_{i:t_i>T}\prod_{t_j\leq T} S_{ji}(T|t_j)\right)\left(\prod_{i:t_i\leq T}\prod_{j:t_j<t_i} S_{ji}(t_i|t_j)\sum_{j:t_j<t_i} H_{ji}(t_i|t_j)\right),$$

where

$$\begin{aligned}\left[\mathbf{H}\right]_{ij} &= H_{ij}(\cdot|\cdot)\\ S_{ij}(t|t_j) &= \exp\left(-\int_{t_j}^{t} H_{ij}(t'|t_j)\mathrm{d}t'\right).\end{aligned}$$

The network inference problem can now be reformulated as an optimization problem. Using the log-likelihood, we now have to solve:

$$\begin{aligned}&\underset{\mathbf{H}}{\text{minimize}} && -\sum_{c\in C}\log L(\mathbf{t}^c;\mathbf{H}) \\ &\text{subject to} && H_{ij}\in\mathcal{M} \qquad\qquad (i,j)\in V\times V,\end{aligned} \tag{4}$$

where C is the set of all observed cascades and $\mathcal{M}$ is the set of all admissible hazard functions. With an appropriate choice of $\mathcal{M}$, it is possible to obtain a convex optimization problem in $\mathbf{H}$ with likelihood functions $(f_{ij})_{(i,j)\in V^2}$ corresponding to usual models.

We have experimented the NetRate algorithm on two different sets $\mathcal{M}$, based on the experiments of [RBS11].

Exponential model For this type of model, the set $\mathcal{M}$ is defined by $\mathcal{M} = \{H : (t_1|t_2) \mapsto \alpha\mathbf{1}_{\{t_1>t_2\}}\ ,\ \alpha \in \mathbb{R}_+\}$, that is the set of constant hazard rates (the positivity condition being necessary for any considered hazard rate).

Rayleigh model For the Rayleigh model, $\mathcal{M}$ is defined by $\mathcal{M} = \{H : (t_1|t_2) \mapsto \alpha(t_1 - t_2)_+\ ,\ \alpha \in \mathbb{R}_+\}$.

For each of these models, the problem is to find the coefficients α_{ij} corresponding to each H_{ij}. A zero coefficient α_{ij} means that node i has not the possibility to infect node j, thus $(i,j) \notin E$. The functions associated to each model are recapitulated in Table 1. These models are convenient because on one hand they have usual likelihood functions and on the other hand they make problem (4) convex.

Model	H	S	f
Exponential	α	$\exp(-\alpha(t_1 - t_2)_+)$	$(\alpha \exp(-\alpha(t_1 - t_2)))\mathbf{1}_{\{t_1 > t_2\}}$
Rayleigh	$\alpha(t_1 - t_2)_+$	$\exp(-\frac{\alpha}{2}(t_1 - t_2)_+)$	$\alpha(t_1 - t_2)_+ \exp(-\frac{\alpha}{2}(t_1 - t_2))$

Table 1: Functions associated to each model.

4 Results

4.1 Networks used for testing

We used two types of network in order to apply NetRate algorithm: synthetic networks, generated with a kronecker generator [LCKF05] and real networks – mainly from Twitter – from SNAP dataset [Les12].

4.1.1 Kronecker generator

Generating networks that are closed to real social networks is an actual challenge. Many works have been made in order to characterize social networks. It is then essential that the generation algorithm used fits the most important properties:

- The degrees are distributed from a power-law [FFF99]. That is, for a given social network, there exists $\gamma > 0$ such that for any degree d, the number n_d of nodes of degree d is given by $n_d \propto d^\gamma$.
- Real graphs have relatively small effective diameter [SRT$^+$01]. The effective diameter of a graph is the minimum diameter over all possible subgraphs using 90% of the nodes.
- The number of edges and the number of nodes of a growing graph follow the densification power law [LKF05]: for a graph $G(t) = (V(t), E(t))$ growing over time, there exists a such that $E(t) \propto N(t)^a$, for all t.

The kronecker generator developped in [LCKF05] allows for realistic network generation.

The idea of the kronecker generator is pretty simple. Let $G = (V, E)$ be a graph and A its adjacency matrix. Let $A^{\otimes 2} = A \otimes A$ be the kronecker product of A with itself. Then the network $G^{(2)}$ obtained from $A^{(2)}$ is a new network that contains $|V|^2$ nodes. So, with some seed G, we can iterate several times in order to get a graph $G^{(n)}$ with an appropriate size.

As an example, we define an adjacency matrix A as follow:

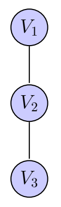
$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Using the kronecker product, the matrix $A^{\otimes 2}$ can be written:

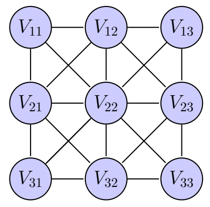
$$A^{\otimes 2} = \begin{pmatrix} A & A & 0 \\ A & A & A \\ 0 & A & A \end{pmatrix}.$$

This kronecker product is illustrated on Figure 4. We purposely removed the self-loop for clarity.

However, this approach is deterministic. In order to add some randomness to the generator, we consider a matrix A that takes its values in $[0, 1]$ instead of $\{0, 1\}$. Then, for each element a of $A^{\otimes n}$, we assign the value 1 with probability a and 0 with probability $1 - a$, to obtain a real adjacency matrix.



(a) Graph associated with A.



(b) Graph associated with $A^{\otimes 2}$.

Figure 4: Illustration of Kronecker multiplication on graphs. Self-loops are omitted.

4.1.2 SNAP dataset

SNAP library is a very complete set of tools in graph study and inference that can be found at [Les12]. It was originally developed by Jure Leskovec and is now a community tool. The code source available is written in `C++` language and implements many useful classes and methods such as forest fire generation or basic statistics on network structures. There exists many input/output formats that one can use for saving work or plotting results. Still on Jure Leskovec's website, a large collection of real datasets can be found. This collection contains many different types of data: Facebook networks, food reviews, publications networks. However, the datasets formats are not necessarily the same so a little time of adaptation can be needed if one wants to use various types of data. In this case, we mainly used the Twitter dataset because it was the most difficult to get influence network from.

4.2 Gradient descent algorithms

Since NetRate reformulate the problem as a convex optimization problem, we are now looking for efficient way to solve this problem. In [RBS11], the method is implemented using the `CVX` toolbox, which does not lead to acceptable performances. We will then try three different gradient descent methods and try to determine which one is more adapted to this method.

4.2.1 ISTA and FISTA

The first algorithm we will use is ISTA ([BJM+11], [BT09]). This algorithm allows to solve efficiently problems of the form:

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ F(x) = f(x) + g(x),$$

where f is a convex, differentiable function, defined over $\mathbb{R}^n$ with L-Lipschitz continuous gradient and g is just convex, possibly non-smooth. The main idea of ISTA is to use a quadratic approximation of f. Let $Q(\cdot, y)$ be the quadratic approximation of F around y defined by:

$$Q(x, y) = g(x) + f(y) + \langle x - y, \nabla f(y)\rangle + \frac{L}{2}\|x - y\|_2^2.$$

The Lipschitz property of ∇f ensures that, for all $x, y \in \mathbb{R}^n$:

$$Q(x,y) \geq F(x).$$

Let us denote p_L the function that maps an element y of $\mathbb{R}^n$ to the element minimizing the quadratic approximation in y:

$$p_L : y \mapsto \arg\min_x Q(x,y).$$

Depending on the form of g, p_L can be much easier to compute than the minimum of F. The idea is then to minimize repeatedly the approximation until convergence. ISTA is presented in Algorithm 1. If the Lipschitz constant is

Algorithm 1 ISTA

```
Require: $x_0$, $L$-Lipschitz $\nabla f$
  $k = 0$
  repeat
    $x_{k+1} \leftarrow p_L(x_k)$
    $k \leftarrow k + 1$
  until convergence
```

unknown or if it improves the performance to consider local Lipschitz constant, it is possible to use the backtracking version of ISTA presented in Algorithm 2.

Algorithm 2 ISTA with backtracking

```
Require: $x_0$, $\eta$, $L_0 > 0$
  $k = 0$
  repeat
    $L_{k+1} \leftarrow L_k$
    while $F(p_{L_{k+1}}) > Q_{L_{k+1}}(p_{L_{k+1}}(x_k), x_k)$ do
      $L_{k+1} \leftarrow \eta L_{k+1}$
    end while
    $x_{k+1} \leftarrow p_{L_{k+1}}(x_k)$
    $k \leftarrow k + 1$
  until convergence
```

We also use FISTA, a variant of ISTA. It uses the two previous iterations to compute the new descent. That is, instead of computing the minimum of the quadratic approximation $Q(\cdot, x_k)$ in order to get x_{k+1}, it computes the minimum of $Q(\cdot, y_k)$, where y_k is defined by:

$$y_k = x_k + \alpha_k(x_k - x_{k-1}) \tag{5}$$

and α_k is a positive coefficient. With an appropriate choice for α_k, the convergence rate of FISTA can be $\mathcal{O}(1/t^2)$ which is a significative improvement from ISTA $\mathcal{O}(1/t)$ rate of convergence. We use the α_k introduced in [BT09] and defined by:

$$\alpha_k = \frac{t_k - 1}{t_{k+1}}$$

where $(t_k)_{k\geq 1}$ is defined by the recurrence:

$$\begin{cases} t_1 = 1 \\ t_{k+1} = \dfrac{1+\sqrt{1+4t_k^2}}{2}. \end{cases}$$

FISTA is presented in Algorithm 3. As for ISTA, it can be computed using a backtracking for the Lipschitz coefficient.

Algorithm 3 FISTA

```
Require: $x_0$, $L$-Lipschitz $\nabla f$
  $x_1 = x_0$
  $y_1 = x_0$
  $t_1 = 1$
  $k = 2$
  repeat
    $x_{k+1} \leftarrow p_L(y_k)$
    $t_{k+1} \leftarrow \frac{1+\sqrt{1+4t_k^2}}{2}$
    $y_{k+1} \leftarrow x_{k+1} + \frac{t_k - 1}{t_{k+1}}(x_{k+1} - x_k)$
    $k \leftarrow k+1$
  until convergence
```

4.2.2 Projected gradient descent

We will compare ISTA and FISTA to a more classic projected gradient descent method. At a given iteration k, we want to find a descent direction $\mathbf{p}^{(k)}$ and a step $\alpha^{(k)}$ such that

$$f(x^{(k-1)}) - f(x^{(k-1)} + \alpha^{(k)}\mathbf{p}^{(k)}) \tag{6}$$

is as big as possible. We will use $-\nabla f(\mathbf{x})$ as a descent direction: we only have to choose a good descent step α. Here, we cannot compute the optimal step that maximizes (6), but we can approximate it. In order to evaluate the quality of a step, we use the Armijo condition: for a given descent direction $\mathbf{p}$ at a position $\mathbf{x}$, a step α is acceptable if:

$$f(\mathbf{x}+\alpha\mathbf{p}) \leq f(\mathbf{x}) + c_1 \alpha \mathbf{p}^T \nabla f(\mathbf{x}), \tag{7}$$

where $c_1 \in (0,1)$ is usually small, typically $c_1 = 10^{-4}$. We now have to find a valid descent step at a minimal cost. Therefore, we use quadratic and cubic interpolation to quickly compute a step candidate.

Let $\mathbf{x}$ be the current position and $\mathbf{p}$ a given descent direction. We define φ by:

$$\varphi : \begin{cases} \mathbb{R} & \mapsto & \mathbb{R} \\ \alpha & \mapsto & f(\mathbf{x}+\alpha\mathbf{p}). \end{cases}$$

Let α_0 be an initial candidate. If α_0 verifies (7), then α_0 is a valid candidate and we can stop. Otherwise, we form a quadratic approximation of φ, using previously computed $\varphi(0)$, $\varphi'(0)$ and $\varphi(\alpha_0)$:

$$\tilde{\varphi}(\alpha) = \varphi(0) + \alpha\varphi'(0) + \alpha^2 \left(\frac{\varphi(\alpha_0) - \varphi(0) - \alpha_0\varphi'(0)}{\alpha_0^2} \right).$$

The approximation reaches its minimum at α_1 defined by:

$$\alpha_1 = \frac{\varphi'(0)\alpha_0^2}{2(\varphi(\alpha_0) - \varphi(0) - \alpha_0\varphi'(0))}.$$

If α_1 is a valid candidate, we can use it as a descent step, else we form a cubic approximation of φ, using previously computed $\varphi(0)$, $\varphi'(0)$, $\varphi(\alpha_0)$ and $\varphi(\alpha_1)$. With a similar reasoning, we obtain a candidate α_2 defined by:

$$\alpha_2 = \min\left(\frac{-b \pm \sqrt{b^2 - 3a\varphi'(0)}}{3a}\right),$$

with

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{\alpha_0^2\alpha_1^2(\alpha_1 - \alpha_0)} \begin{pmatrix} \alpha_0^2 & -\alpha_1^2 \\ -\alpha_0^3 & \alpha_1^3 \end{pmatrix} \begin{pmatrix} \varphi(\alpha_1) - \varphi(0) - \varphi'(0)\alpha_1 \\ \varphi(\alpha_0) - \varphi(0) - \varphi'(0)\alpha_0 \end{pmatrix}.$$

Again, if α_2 is a valid candidate, we can stop. Otherwise, we repeat previous step using the two latest candidates, until α_k is acceptable. Algorithm 4 summarizes the backtracking process. Regarding the value of the initial step α_0, we use:

Algorithm 4 Backtracking

```
Require: $\alpha_0$, $\mathbf{x}$, $\mathbf{p}$
  if $\alpha_0$ is acceptable then
      return $\alpha_0$
  else
      $\alpha_1 \leftarrow \arg\min_\alpha \tilde{\varphi}_{quad}(\alpha; \alpha_0)$
      if $\alpha_1$ is acceptable then
          return $\alpha_1$
      else
          $k \leftarrow 1$
          repeat
              $\alpha_{k+1} \leftarrow \arg\min_\alpha \tilde{\varphi}_{cubic}(\alpha; \alpha_k, \alpha_{k-1})$
              $k \leftarrow k + 1$
          until $\alpha_k$ is acceptable
          return $\alpha_k$
      end if
  end if
```

$$\alpha_0 = \alpha^{(k-1)} \frac{\mathbf{p}^{(k-1)T}\nabla f(\mathbf{x}^{(k-1)})}{\mathbf{p}^{(k)T}\nabla f(\mathbf{x}^{(k)})}.$$

Using this backtracking, we can now specify the projected gradient algorithm we will use. Let $\mathbf{l}$ and $\mathbf{u}$ be respectively lower and upper bounds on $\mathbf{x}$ coordinates. We need to set to 0 the component of the descent direction $\mathbf{p}$ that would bring $\mathbf{x}$ outside of the admissible set, for any descent step α. The i-th component of the descent direction $\mathbf{p}$ is given by:

$$p_i = \begin{cases} 0 & \text{if } x_i \geq u_i \text{ and } \nabla f(\mathbf{x})_i < 0 \\ 0 & \text{if } x_i \leq l_i \text{ and } \nabla f(\mathbf{x})_i > 0 \\ -\nabla f(\mathbf{x}) & \text{if } x_i \in (l_i, u_i). \end{cases}$$

Given these definitions, the projected gradient descent is presented in Algorithm 5.

Algorithm 5 Projected gradient descent

```
Require: $\mathbf{x}^{(0)}$, $(\mathbf{l}, \mathbf{u})$
  $k \leftarrow 0$
  repeat
    Compute direction descent $\mathbf{p}^{(k+1)}$
    $\alpha^{(k+1)} \leftarrow$ Backtracking($\alpha_0^{(k+1)}, \mathbf{x}^{(\mathbf{k})}, \mathbf{p}^{(k+1)}$)
    $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k+1)}\mathbf{p}^{(k+1)}$
    Projection of $\mathbf{x}^{(k+1)}$ on the admissible set
    $k \leftarrow k + 1$
  until convergence
  return $\mathbf{x}^{(k)}$
```

4.3 Implementation

Before implementing gradient descent algorithms with our model, two remarks must be added. First, we can use the fact that problem (4) can be solved independently over each column of $\mathbf{H}$: we can therefore solve $|V|$ problems of size $|V|$ instead of one problem of size $|V|^2$. Also, if a given pair of node (i, j) is such that $t_j^c > t_i^c$ in any cascade where $t_i^c \leq T$, then the hazard rate H_{ji} is necessarily zero. Thanks to these properties, we can set many hazard rates directly to 0 and compute each column H_{*i} independently.

Let us formulate explicitly the problem for the exponential model, as it is simpler and can easily be extrapolated to the Rayleigh model. For each $i \in V$, we aim to solve the problem:

$$\begin{array}{ll} \underset{\mathbf{H_{*i}}}{\text{minimize}} & -\sum_{c \in C} \log L_i(\mathbf{t^c}; \mathbf{H_{*i}}) \\ \text{subject to} & H_{ji} \in \mathcal{M}_{\exp} \qquad j \in V, \end{array} \tag{8}$$

where L_i is the part of the likelihood that depends on $(H_{ji})_{j \in V}$. Using the expressions developped in Table 1, we can rewrite the problem as follow:

$$\begin{array}{ll} \underset{\mathbf{x} \in \mathbb{R}^\mathbf{n}}{\text{minimize}} & \mathbf{w}_i^T \mathbf{x} - \sum_{c : t_i^c \leq T} \log\left(\mathbf{u}_{i,c}^T \mathbf{x}\right) \\ \text{subject to} & \mathbf{x} \succeq 0, \end{array} \tag{9}$$

where

$$\begin{cases} \mathbf{w}_i = \left(\sum_{c \in C} (\min(t_i^c, T) - t_j)_+\right)_{j \in V} \\ \mathbf{u}_{i,c} = \left(\mathbf{1}_{\{t_j^c < t_i^c\}}\right)_{j \in V}. \end{cases}$$

This formulation allows to preprocess $\mathbf{w}$ and $(\mathbf{u}_{i,c})_{c \in C}$ and make optimization algorithms faster. Finally, if we define f and g as follow:

$$f : \mathbf{x} \mapsto \mathbf{w_i}^T \mathbf{x} - \sum_{c \in C} \log\left(\mathbf{u}_{i,c}^T \mathbf{x}\right)$$

$$g : \mathbf{x} \mapsto \iota_{\{\mathbf{x} \succeq 0\}} = \begin{cases} 0 & \text{if } x \succeq 0 \\ +\infty & \text{otherwise} \end{cases}$$

we can use ISTA on $F = f + g$. In this case, f is not defined on $\mathbb{R}^n$ but only on the positive orthant of $\mathbb{R}^n$. This is different from what we assumed in ISTA definition but will not be problematic because ISTA is a descent algorithm. The gradient of f is defined by:

$$\nabla f : \mathbf{x} \mapsto \mathbf{w_i} - \sum_{c \in C} \frac{1}{\mathbf{u}_{i,c}^T \mathbf{x}} \mathbf{u}_{i,c}.$$

Unfortunately, ∇f is not likely to be Lipschitz continuous (in fact, ∇f is Lips-

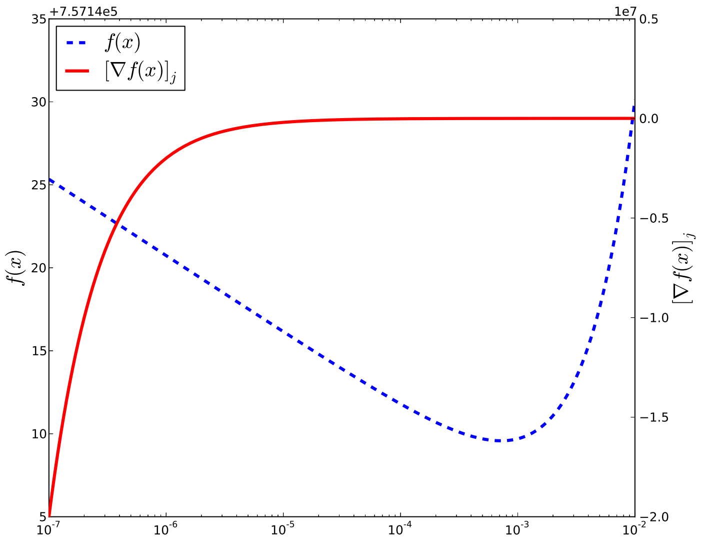


Figure 5: Example where the i-th component of $\nabla f(x)$ is clearly not Lipschitz continuous.

chitz continuous iff the solution of problem (9) is 0). We will however use ISTA with backtracking for the Lipschitz coefficient in order to find adapted local Lipschitz coefficient. Figure 5 illustrates the fact that $\nabla f(x)$ is not Lipschitz continuous: it represents the evolution of the j-th component of $\nabla f(x)$ when x takes its values in $\{x_0 + te_j,\, t \in [10^{-5}, 10^{-2}]\}$, where e_j is the j-th element of the canonical base of $\mathbb{R}^n$ and $x_0 = (\mathbf{1}_{\{j \neq i\}})_{j \in V}$. In this case, there exists a cascade c in which j is the only node infected prior to i. For that reason, the j-th component of $\nabla f(x)$ explodes around 0. Generally speaking, for a given cascade c, if the coordinates associated to previously infected nodes jointly tend toward 0, then $\|\nabla f(x)\|$ will grow as $1/\|x\|$.

We implemented ISTA algorithm on a real dataset of 200 Twitter users. Originally the dataset contained about 500 users but we use a truncated one for computational reasons. We also use various numbers of cascades based on information collected in SNAP library ([Les12]). We used an initial Lipschitz parameter $L_0 = 1$. It is possible to choose to take the maximum eigenvalue of $\nabla^2 f$ as initial value but it is quite long to compute. Also, every 10 iterations, we reset L to L_0, instead of keeping it increasing. Figure 6 presents the evolution of F with ISTA iterations, when $F = f + g$ and when $F = f_\epsilon + g$ (see

FISTA implementation below for explanations about f_ϵ). We can see that ISTA

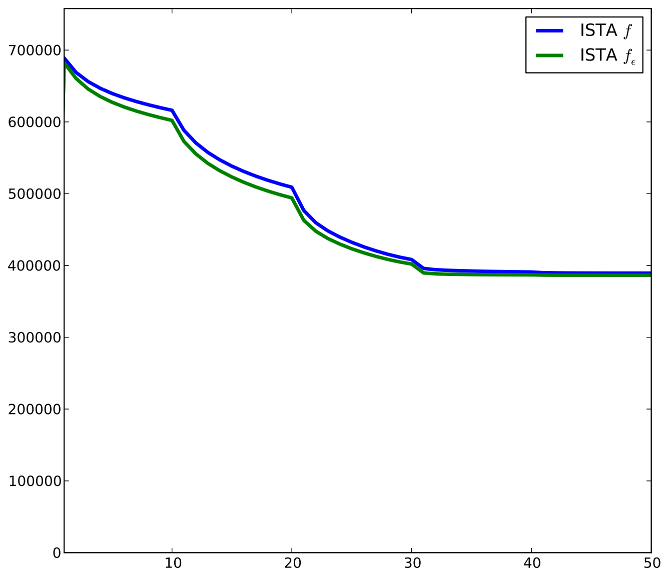


Figure 6: Evolution of F with ISTA iterations.

algorithm converges in about 30 iterations. Additionally, we see that Lipschitz coefficient reinitialization improves the descent (but the reinitialization step is longer than other iterations). Figures 7 and 8 shows the evolution of precision with the number of iterations. This time, we can see that precision seems to converge around 60 iterations.

The implementation of FISTA is more problematic. As mentioned previously, the function f is not defined over all $\mathbb{R}^n$. It was not an issue in the ISTA implementation but it makes the FISTA implementation impossible: as FISTA is not a descent algorithm, it could go out of the domain when adding the second term of (5). We address this problem by setting a threshold $\epsilon > 0$ and we consider f_ϵ defined by:

$$f_\epsilon(\mathbf{x}) = f(\text{proj}_\epsilon(\mathbf{x})) + (\mathbf{x} - \text{proj}_\epsilon(\mathbf{x}))^T \nabla f(\text{proj}_\epsilon(\mathbf{x}))$$

where proj_ϵ is defined by:

$$\text{proj}_\epsilon : \mathbf{x} \mapsto (\max(x_i, \epsilon))_{i \in V}.$$

With an appropriate choice for ϵ, the global minimum of $f_\epsilon + g$ can stay the same and f_ϵ is now defined over $\mathbb{R}^n$. Moreover, ∇f_ϵ is now Lipschitz continuous. Figure 9 illustrates this linear prolongation.

We implemented FISTA in the same way than ISTA. Figure 10 shows the evolution of F with the number of iteration. Again, the objective function seems to converge quite quickly. Figure 11 shows that FISTA allows precision to converge quicker than in the ISTA case. Table 2 shows the computations

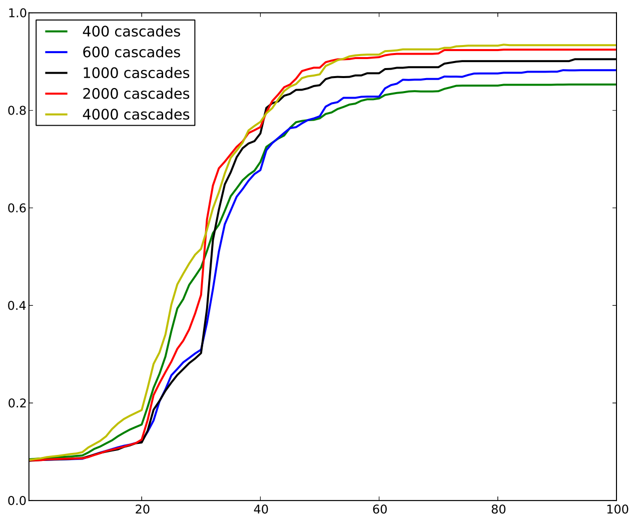


Figure 7: Evolution of precision with iterations for several numbers of cascades when $F = f + g$.

Algorithm	400 cascades	1000 cascades	2000 cascades
ISTA with f	0.22 s	0.54 s	1.42 s
ISTA with f_ϵ	0.62 s	1.44 s	3.66 s
FISTA with f_ϵ	2.75 s	5.79 s	27.4 s

Table 2: Computation times for each iteration

times for each iteration of the algorithms. There is obviously improvements to be made on FISTA, as it should not take much longer than ISTA. As f_ϵ is longer to compute than f: it could be a good improvement to try to make it faster.

Although ISTA and FISTA can be very efficient in some cases, the non-Lipschitz gradient function of problem (4) sometimes leads to poor performances.

In order to compare ISTA and projected gradient descent algorithm, we run both algorithms for different number of cascades. All methods lead to quite similar results. However, computation times are really in favor of projected gradient descent method, as shown in Table 3.

Algorithm	400 cascades	1000 cascades	2000 cascades
ISTA	10.52 s	20.95 s	35.26 s
Projected gradient descent	4.13 s	6.55 s	9.61 s

Table 3: Computation times for identical convergence criterion.

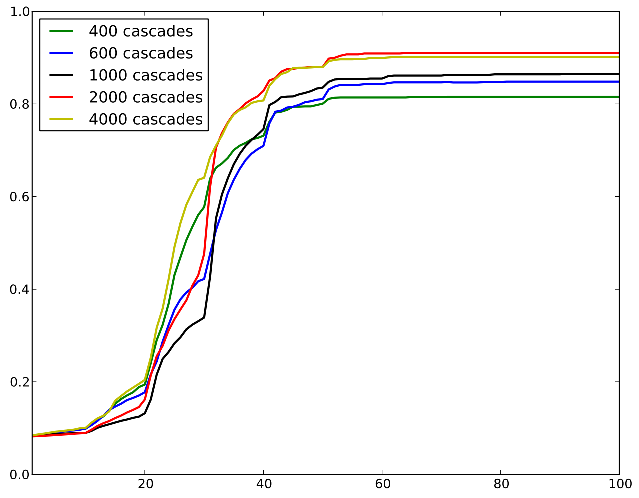


Figure 8: Evolution of precision with iterations for several numbers of cascades when $F = f_\epsilon + g$.

4.4 Likelihood measurements

Since real networks cannot provide for ground truth of influence network, we measure the performance of NetRate algorithm by computing the network of influence on a training set of cascades and then computing the log-likelihood of a testing set of cascades, given the inferred network of influence. Figure 12 shows the evolution of log-likelihood for a testing set of 200 synthetic cascades on a 200 nodes network. We can see that NetRate does not need a lot of cascades: about 2 cascades by node is enough to maximize the likelihood on the testing set.

We introduce a penalty parameter λ such that the function to optimize is now:

$$\tilde{f} : \mathbf{x} \mapsto f(\mathbf{x}) + \lambda \|\mathbf{x}\|_1.$$

We can see on Figure 13 that this parameter can increase the likelihood on the testing set if it is well calibrated. Here we have considered a small training set of only 100 cascades for a 200 nodes network.

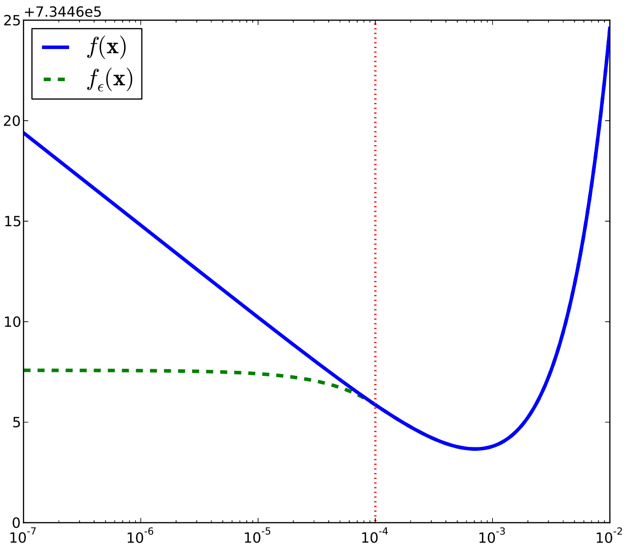


Figure 9: Illustration of the linear prolongation of f below $\epsilon = 10^{-4}$.

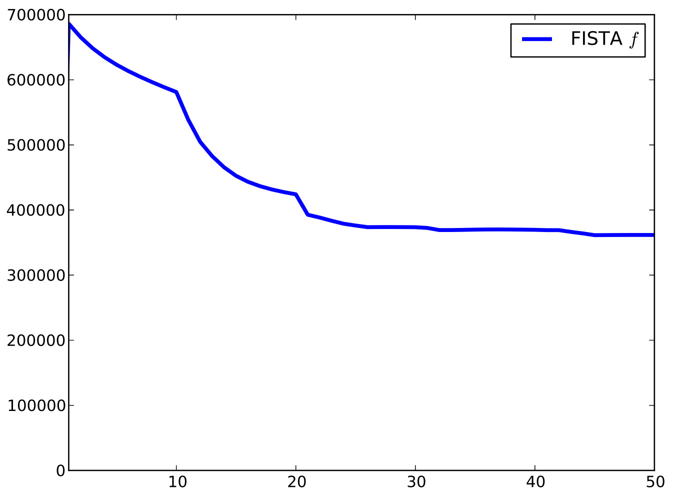


Figure 10: Evolution of F with FISTA iterations.

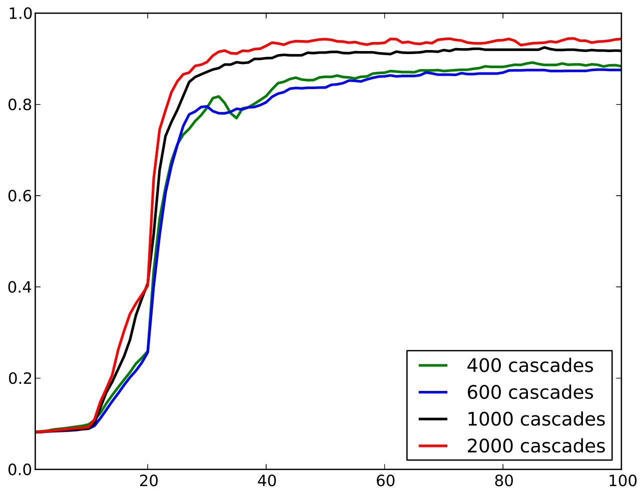


Figure 11: Evolution of precision with iterations for several numbers of cascades when $F = f_\epsilon + g$.

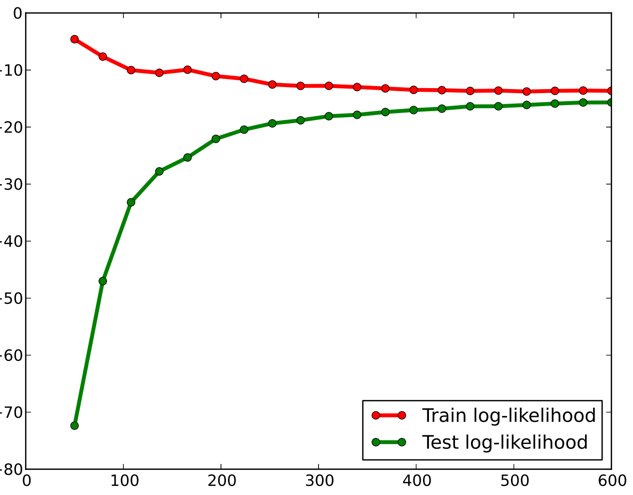


Figure 12: Evolution of average log-likelihood with the number of cascades.

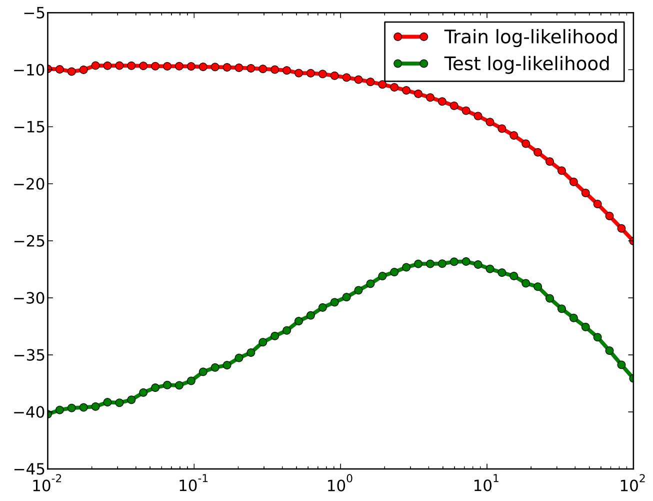


Figure 13: Evolution of average log-likelihood with the penalty parameter.

5 Future work

5.1 Diffusion of several types of cascades

5.1.1 Model

In usual diffusion models for cascades of information, the type of information being diffused is not considered: only one network of influence has to be inferred. The goal is then to infer the matrix $\mathbf{x}$, where x_{ij} represents the influence parameter of i over j. If $x_{ij} = 0$, i is considered to have no influence on j.

Here we want to label each cascade with the corresponding type of information begin diffused. Let K be the number of different types we consider. Now, K matrices of influence have to be found. Let $\mathbf{x} = (\mathbf{x}^{(k)})_{1 \leq k \leq K}$ be the matrices corresponding to each type. For each cascade $c \in \mathcal{C}$, we also introduce a latent variable z_c, that determines the type of c. Finally, we define τ_k as the prior probability of label k, for $1 \leq k \leq K$. This variable must be inferred with $\mathbf{x}$.

The likelihood of a cascade c with label z can now be expressed as:

$$\begin{aligned} L_{K,c}(\mathbf{x}, \tau; \mathbf{t}^c, z) &= p(\mathbf{t}^c, z | \mathbf{x}, \tau) \\ &= \sum_{k=1}^{K} \mathbf{1}_{\{z=k\}} p(z=k) p\left(\mathbf{t}^c |\ \mathbf{x}^{(k)}\right) \\ &= \sum_{k=1}^{K} \mathbf{1}_{\{z=k\}} \tau_k L\left(\mathbf{x}^{(k)}; \mathbf{t}^c\right), \end{aligned}$$

where $L(\mathbf{x}; \mathbf{t}^c)$ is the single-type likelihood of the cascade c. We can express the likelihood of a set of cascades $\mathcal{C}$ with respective labels $\mathbf{z}$ as:

$$L_K(\mathbf{x}, \tau; \mathbf{t}, \mathbf{z}) = \prod_{c \in \mathcal{C}} L_{K,c}(\mathbf{x}, \tau; \mathbf{t}^c, z_c).$$

5.1.2 Analysis

We solve this problem using an EM algorithm: the expectation step will focus on the latent variables $(z_c)_{c \in \mathcal{C}}$ whereas the maximization step will compute new values of $(\mathbf{x}^{(k)})_{1 \leq k \leq K}$ and $(\tau_k)_{1 \leq k \leq K}$.

For $1 \leq k \leq K$ and $t > 0$, we define $\mathbf{x}^{(k)}(t)$ as the estimation of the influence matrix of type k at iteration t. Similarly, we define $\tau_k(t)$ as the estimation of the prior probability of label k at iteration t.

E-step The goal of the E-step is to compute the following function:

$$Q(\mathbf{x}', \tau' |\ \mathbf{x}(t), \tau(t)) = \mathbb{E}_{Z|\ X, \tau}\left[\log L_K(\mathbf{x}', \tau'; \mathbf{t}, \mathbf{z})\right].$$

Using the previous expression of the likelihood, we obtain:

$$\begin{aligned} \log L_K(\mathbf{x}', \tau'; \mathbf{t}, \mathbf{z}) &= \sum_{c \in \mathcal{C}} \log L_{K,c}(\mathbf{x}', \tau'; \mathbf{t}^c, z_c) \\ &= \sum_{c \in \mathcal{C}} \log \left(\sum_{k=1}^{K} \mathbf{1}_{\{Z=k\}} \tau_k L\left(\mathbf{x}^{(k)}; \mathbf{t}^c\right) \right) \\ &= \sum_{c \in \mathcal{C}} \sum_{k=1}^{K} \mathbf{1}_{\{Z=k\}} \left(\log \tau_k + \log L\left(\mathbf{x}^{(k)}; \mathbf{t}^c\right) \right). \end{aligned}$$

With this expression we can write:

$$\begin{aligned}
Q(\mathbf{x}', \tau' |\, \mathbf{x}(t), \tau(t)) &= \mathbb{E}_{Z_c|\ X,\tau}\left[\log L_K(\mathbf{x}', \tau'; \mathbf{t}, \mathbf{z})\right] \\
&= \sum_{c\in\mathcal{C}} \sum_{k=1}^{K} \mathbb{E}_{Z_c|\ X,\tau}\left[\mathbf{1}_{\{Z_c=k\}}\right] \left(\log \tau'_k + \log L\left(\mathbf{x}'^{(k)}; \mathbf{t}^c\right)\right) \\
&= \sum_{c\in\mathcal{C}} \sum_{k=1}^{K} \mathbb{P}\left(Z_c = k | \mathbf{x}(t), \tau(t)\right) \left(\log \tau'_k + \log L\left(\mathbf{x}'^{(k)}; \mathbf{t}^c\right)\right).
\end{aligned}$$

If we define $(a_{c,k}(t))_{c,k}$ as follow:

$$a_{c,k}(t) = \mathbb{P}\left(Z_c = k | \mathbf{x}(t), \tau(t)\right) = \frac{\tau_k(t) L(\mathbf{x}^{(k)}(t); \mathbf{t}^c)}{\sum_{j=1}^{K} \tau_j(t) L(\mathbf{x}^{(j)}(t); \mathbf{t}^c)},$$

we can write, finally:

$$Q(\mathbf{x}', \tau' |\, \mathbf{x}(t), \tau(t)) = \sum_{c\in\mathcal{C}} \sum_{k=1}^{K} a_{c,k}(t) \left(\log \tau'_k + \log L\left(\mathbf{x}'^{(k)}; \mathbf{t}^c\right)\right).$$

M-step The M-step now simply consists in maximizing $Q\left(\cdot, \cdot | \mathbf{x}(t), \tau(t)\right)$. The maximization can be done independently in τ and in each $\mathbf{x}$. The maximization problem in τ can be written as:

$$\begin{aligned}
&\text{maximize} && \sum_{c\in\mathcal{C}} \sum_{k=1}^{K} a_{c,k}(t) \log \tau_k \\
&\text{subject to} && \sum_{k=1}^{K} \tau_k = 1.
\end{aligned}$$

Using the dual formulation we obtain the solution $\tau(t+1)$ of this problem:

$$\tau_k(t+1) = \frac{\sum_{c\in\mathcal{C}} a_{c,k}(t)}{\sum_{c\in\mathcal{C}} \sum_{j=1}^{K} a_{c,j}(t)}, \text{ for } \ 1 \le k \le K.$$

In the single-type case, the maximization according to $\mathbf{x}$ could be reformulated as:

$$\begin{aligned}
&\text{minimize} && -\sum_{c\in\mathcal{C}} \log L(\mathbf{x}; \mathbf{t}^c) \\
&\text{subject to} && \mathbf{x} \succeq 0.
\end{aligned}$$

or:

$$\begin{aligned}
&\text{minimize} && \mathbf{w}_i^T \mathbf{x}_{*i} - \sum_{c\in\mathcal{C}} \log\left(\mathbf{u}_{i,c}^T \mathbf{x}_{*i}\right) \\
&\text{subject to} && \mathbf{x}_{*i} \succeq 0 && \text{for } i \in V,
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{w}_i &= \left(\sum_{c\in\mathcal{C}} \left(\min(t_i^c, T) - t_j^c\right)_+\right)_{j\in V}, \\
\mathbf{u}_{i,c} &= \left(\mathbf{1}_{\{t_j^c < t_i^c\}}\right)_{j\in V}.
\end{aligned}$$

Now the problem can be expressed as follow:

$$\begin{aligned} &\text{minimize} \quad -\sum_{c\in\mathcal{C}}\sum_{k=1}^{K} a_{c,k}(t)\log L\left(\mathbf{x}^{(k)};\mathbf{t}^c\right) \\ &\text{subject to} \quad \mathbf{x}^{(k)} \succeq 0\ ,\ \ 1 \le k \le K. \end{aligned}$$

This problem can be solved separately in k and again in $x_{*i}^{(k)}$. We finally obtain for $1 \le k \le K$ and $i \in V$, the following problem:

$$\begin{aligned} &\text{minimize} \quad < \mathbf{w}_i^{(k)}, \mathbf{x}_{*i}^{(k)} > - \sum_{c\in\mathcal{C}} a_{c,k}(t)\log\left(\mathbf{u}_{i,c}^T \mathbf{x}_{*i}^{(k)}\right) \\ &\text{subject to} \quad \mathbf{x}_{*i}^{(k)} \succeq 0, \end{aligned}$$

where

$$\begin{aligned} \mathbf{w}_i^{(k)} &= \left(\sum_{c\in\mathcal{C}} a_{c,k}(t)\left(\min(t_i^c, T) - t_j^c\right)_+\right)_{j\in V} \\ \mathbf{u}_c &= \left(\mathbf{1}_{\{t_j^c < t_i^c\}}\right)_{j\in V}. \end{aligned}$$

5.2 Scoop.it! dataset

Scoop.it! is a website designed for article publishing and sharing. An user can create a topic and fill it with several articles. These articles can be original or "rescooped" from another user. The notion of influence is applicable here: if user A often rescoop content from user B, we would like to say that B influences A.

The data from Scoop.it! could be a very interesting testing set for multi-type diffusion as it allows to identify the information type quite easily: since article are shared, one can simply determine the general topic in order to verify whether or not the classification from the EM algorithm is relevant.

6 Conclusion

Inferring a network of influence is a very challenging problem. Several assumptions on the way the information is disseminated have to be made in order to solve it efficiently. These assumptions does not seem to be too restrictive and still allow for satisfying results. Here we were able to consequently decrease the computation time for NetRate algorithm: instead of a very generic toolbox, we were able to find a well-adapted method to solve the optimization problem induced by NetRate problem formulation. We also added a global sparsity constraint that offered a substantial likelihood increase, while keeping the computation time at an acceptable level.

In the future, it would be interesting to implement the EM algorithm presented in Section 5.1 in order to classify the type of information being disseminated while inferring the corresponding network. Also, it would be interesting to explore the inference problem using the threshold model instead of the independent cascade model, as it fits better to some information dissemination phenomena.

References

[ABGG08] Odd O Aalen, Ørnulf Borgan, Håkon K Gjessing, and Stein Gjessing. *Survival and event history analysis: a process point of view.* Springer, 2008.

[BJM$^+$11] Francis Bach, Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, et al. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 2011.

[BT09] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM Computer Communication Review*, volume 29, pages 251–262. ACM, 1999.

[GRLK10] 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.

[KKT03] 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.

[KKT05] David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *Automata, languages and programming*, pages 1127–1138. Springer, 2005.

[LCKF05] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Knowledge Discovery in Databases: PKDD 2005*, pages 133–145. Springer, 2005.

[Les12] Jure Leskovec. SNAP Dataset. `http://snap.stanford.edu/data/index.html`, 2012.

[LKF05] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[ML10] Seth A Myers and Jure Leskovec. On the convexity of latent social network inference. *arXiv preprint arXiv:1010.5504*, 2010.

[MR07] Elchanan Mossel and Sebastien Roch. On the submodularity of influence in social networks. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 128–134. ACM, 2007.

[RBS11] Manuel Gomez Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. *arXiv preprint arXiv:1105.0697*, 2011.

[SRT$^+$01] Georgos Siganos, U. C. Riverside, Sudhir L Tauro, U. C. Riverside, Michalis Faloutsos, and U. C. Riverside. A simple conceptual model for the internet topology. *IEEE Global Internet*, 2001.