Learning Influence Probabilities In Social Networks

Amit Goyal
University of British Columbia
Vancouver, BC, Canada
goyal@cs.ubc.ca

Francesco Bonchi
Yahoo! Research
Barcelona, Spain
bonchi@yahoo-inc.com

Laks V. S. Lakshmanan
University of British Columbia
Vancouver, BC, Canada
laks@cs.ubc.ca

ABSTRACT

Recently, there has been tremendous interest in the phenomenon of influence propagation in social networks. The studies in this area assume they have as input to their problems a social graph with edges labeled with probabilities of influence between users. However, the question of where these probabilities come from or how they can be computed from real social network data has been largely ignored until now. Thus it is interesting to ask whether from a social graph and a log of actions by its users, one can build models of influence. This is the main problem attacked in this paper. In addition to proposing models and algorithms for learning the model parameters and for testing the learned models to make predictions, we also develop techniques for predicting the time by which a user may be expected to perform an action. We validate our ideas and techniques using the Flickr data set consisting of a social graph with 1.3M nodes, 40M edges, and an action log consisting of 35M tuples referring to 300K distinct actions. Beyond showing that there is genuine influence happening in a real social network, we show that our techniques have excellent prediction performance.

Categories and Subject Descriptors H.2.8 [Database Management]: Database Applications - *Data Mining*
General Terms: Algorithms
Keywords: Social networks, Influence, Viral marketing.

1. INTRODUCTION

In recent years, there has been tremendous interest in the phenomenon of influence exerted by users of an online social network on other users and in how it propagates in the network. The idea is that when a user sees their social contacts performing an action such as joining an online community (say TapIt[1]), that user may decide to perform the action themselves. In truth, when a user performs an action, she may have any one of a number of reasons for doing so: she may have heard of it outside of the online social network and may have decided it is worthwhile; the action is very popular (e.g., buying an iPhone 4G may be such an action); or she may be genuinely influenced by seeing her social contacts perform that action. If there is genuine influence, it can be leveraged for a number of applications, arguably the most famous among which is viral marketing [4, 13, 9]. Other applications include personalized recommendations [17, 16] and feed ranking in social networks [15]. Besides, patterns of influence can be taken as a sign of user trust and exploited for computing trust propagation [6, 23, 5, 18] in large networks and in P2P systems.

While many of the applications mentioned above essentially assume that influence exists as a real phenomenon, two key pieces are missing from the picture. First, while some empirical studies have reported evidences of influence propagating on the social linkage [2, 7], others authors have challenged the fact that influence really exists and that it propagates between users. Indeed, Watts [22, 20, 21] challenges the very notion of influential users that are often assumed in viral marketing papers. As well, in a recent paper, Anagnostopoulos *et al.* [1] have developed techniques for showing that influence is *not* genuine and using them, showed that in the Flickr tagging data, while there is substantial social correlation in tagging behavior, it cannot be attributed to influence. This raises the question, is there evidence of genuine influence in any real social network data? The second missing piece is a systematic study of models of influence. In particular, all viral marketing papers assume that they are given as input a social graph with edges labeled by the probability with which a user's action will be influenced by her neighbor's actions. *To our knowledge, the question how or from where one can compute these probabilities of influence has been largely left open.*

In this paper, our goal is to address both the issues above: we devise various probabilistic models of influence, and w.r.t. them we show that influence is genuinely happening in a real-world social network.

The starting observation is that while real social networks don't come with edges labeled with influence probabilities, they do come with an *action log*. Informally, an action log is a table that chronicles any actions performed by every user. It is thus interesting to ask whether by analyzing the action log together with the network, we can study the two questions above. In undertaking such a study, two things should be kept in mind. First, any models proposed for influence should be compatible with the assumptions made in

[1]TapIt is a water bottle refilling network founded in 2008 to give people free access to clean sustainable water on the go: `tapitwater.com`.

WSDM'10, February 4–6, 2010, New York City, New York, USA.

applications such a viral marketing. Viral marketing papers typically assume a diffusion model of influence which satisfies a property known as submodularity. While we defer a formal definition to Section 4, intuitively it can be thought of as a law of diminishing returns. Second, the action log is huge in size. Thus, any algorithms developed for learning and testing the influence models should make minimal number of scans over this data.

In this paper, we make the following contributions:

- We propose a solution framework. By starting with the diffusion models assumed in viral marketing, we elicit desiderata for models of probabilistic influence. These include a mandatory *submodularity* property and a desirable, but not mandatory *incrementality* property.
- We propose a variety of probabilistic models of influence between users in a social network. We show that all of them satisfy submodularity while all with the exception of one satisfies incrementality. Intuitively, an incremental model allows efficient testing.
- We develop algorithms for learning (the parameters of) all the proposed models, taking as input a social graph and an action log. We optimize the scans and show that our algorithms can learn all the models in no more than two scans. A highlight is that some of our models let us predict not just whether a user will perform an action but the time by which she will perform it.
- One of the most accurate models is what we call *continuous time* model. Unfortunately, it is the most expensive to test (it's not incremental). To mitigate this, we develop an approximate model called *discrete time* model, which is incremental and is much more efficient to test.
- It turns out evaluating (testing) the learned models is far from trivial. Thereto, we develop algorithms for testing all the models learned. The algorithms require just one scan over the action log.
- Last but not the least, we put the models and algorithms to test on the Flickr data set where for actions, we take users joining online communities. The data set consists of a graph with 1.3M nodes and more than 40M edges and an action log with 35M tuples referring to 300K different actions. Our results show there is genuine influence between users. In particular, we introduce the metrics of *user influenceability* and *action influence quotient*. Users (actions) with high values for this metric do experience genuine influence compared to those with low values. Our results also show that all proposed models have a reasonable performance, and continuous time model has the best performance. Discrete time model has an almost identical performance while it is much more efficient to test. We also show that our models can predict the time at which a user will perform an action with an impressive error margin.

Section 2 provides relevant background and discusses related work. In Section 3, we give a formal statement of the problem studied while in Section 4, we develop a solution framework and in Section 5 we present the models for probabilistic influence. Section 6 presents the algorithms for learning the models and for evaluating them, while Section 7 presents results from an extensive set of experiments.

2. BACKGROUND AND RELATED WORK

Suppose we are given a social network together with the estimates of reciprocal influence between individuals in the network, and suppose that we want to push a new product in the market. The idea behind *viral marketing* is that by targeting the most influential users in the network we can activate a chain-reaction of influence driven by word-of-mouth, in such a way that with a very small marketing cost we can actually reach a very large portion of the network. The mining problem of *influence maximization* is the following: given such a network with influence estimates, how to select an initial set of k users such that they eventually influence the largest number of users in the social network.

Domingos and Richardson [4, 13] were the first to consider the propagation of influence and the problem of identification of influential users by a data mining perspective. The problem is tackled by means of a probabilistic model of interaction, and heuristics are given for choosing the users.

Kempe *et al.* [9] attacked roughly the same problem as a problem in discrete optimization (which is known to be NP-complete), obtaining provable approximation guarantees in several preexisting models coming from mathematical sociology. In particular their work focuses on two fundamental propagation models, namely *Linear Threshold Model* and *Independent Cascade Model*. They also proposed a broader framework that that simultaneously generalizes the Linear Threshold and Independent Cascade models, and that has equivalent formulations in terms of thresholds and cascades. We next recall the threshold formulation.

General Threshold Model. At a given timestamp, each node is either active (an adopter of the innovation, or a customer which already purchased the product) or inactive, and each node's tendency to become active increases monotonically as more of its neighbors become active. Time unfolds deterministically in discrete steps. As time unfolds, more and more of neighbors of an inactive node u may become active, eventually making u become active, and u's activation may in turn trigger further activations by nodes to which u is connected. In the General Threshold Model each node u has a monotone activation function $f_u : 2^{N(u)} \rightarrow [0, 1]$, from the set of neighbors N of u, to real numbers in $[0, 1]$, and a threshold θ_u, chosen independently and uniformly at random from the interval [0,1]. A node u becomes active at time $t + 1$ if $f_u(S) \geq \theta_u$, where S is the set of neighbors of u that are active at time t.

Under all the propagation models discussed in [9], the influence maximization problem is shown to be NP-hard. Kempe *et al.* however show that the *influence spread* of a set of nodes is a function with the nice features of being monotone and submodular (see Section 4). Exploiting these properties they present a greedy approximation algorithm. Indeed, for any monotone and submodular function f with $f(\emptyset) = 0$, the problem of finding a set S of fixed cardinality k such that $f(S)$ is maximal, can be approximated by a greedy algorithm within a factor of $(1 - 1/e)$, as shown in [11].

A limitation of [9] is the efficiency of their greedy algorithm, which requires to compute the influence spread given a seed set. For this difficult task they run Monte-Carlo simulations of the propagation model for sufficiently many times

to obtain an accurate estimate, resulting in very long computation time. A recent line of research [10, 3] has started developing methods for improving the efficiency of the greedy algorithm for influence maximization.

Leskovec *et al.* [10] study the propagation problem from a different perspective namely *outbreak detection*: how to select nodes in a network in order to detect the spread of a virus as fast as possible? They present a general methodology for near optimal sensor placement in these and related problems. By exploiting submodularity they develop an efficient algorithm based on a "lazy-forward" optimization in selecting new seeds, achieving near optimal placements, while being 700 times faster than the simple greedy algorithm. In spite of this big improvement over the basic greedy algorithm, their method still faces serious scalability problems as shown in [3]. In that paper, Chen *et al.* improve the efficiency of the greedy algorithm and propose new degree discount heuristics that produce influence spread close to that of the greedy algorithm but much more efficiently.

All the papers discussed above assume the basic framework and propagation models of [9], where the influence probabilities $p_{v,u}$ on the edges are given as input. In this paper instead we study how this probabilities can be produced by mining past influence cascades, or in other terms, the past behavior of users.

Tang *et al.* [19] introduce the problem of topic-based social influence analysis. Given a social network and a topic distribution for each user, the problem is to find topic-specific subnetworks, and topic-specific influence weights between members of the subnetworks. They propose a Topical Affinity Propagation (TAP) approach using a graphical probabilistic model. They also deal with the efficiency problem by devising a distributed learning algorithm under the Mapreduce programming model. Moreover, they also discuss the applicability of their approach to the *expert finding* problem.

Independently and concurrently with us, Saito *et al.* [14][2] have studied the same problem we tackle in this paper, focussing on the Independent Cascade model of propagation. They formally define the likelihood maximization problem and then apply EM algorithm to solve it. While their formulation is elegant, it is not scalable to huge datasets like the one we are dealing in this work. This is due to the fact that in each iteration, the EM algorithm must update the influence probability associated to each edge.

3. PROBLEM DEFINITION

We are given a *social graph* in the form of an undirected graph $G = (V, E, \mathcal{T})$ where the nodes V are users. An undirected edge $(u, v) \in E$ between users u and v represents a social tie between the users. $\mathcal{T} : E \rightarrow \mathbb{N}$ is a function labeling each edge with the timestamp at which the social tie was created.[3] We're also given an *action log*, a relation $Actions(User, Action, Time)$, which contains a tuple (u, a, t_u) indicating that user u performed action a at time t_u. It contains such a tuple for every action performed by every user of the system. We will assume that the projection of *Actions* on the first column is contained in the set of nodes V of the social graph G. In other words, users in the *Actions* table correspond to nodes of the graph. We let $\mathcal{A}$ denote the universe of actions. In the following, we assume for ease of exposition that a user performs an action at most once. We denote with A_u the number of actions performed by user u in the training set, with $A_{u\&v}$ the number of actions performed by both u and v in the training set, with $A_{u|v}$ the number of actions either u or v performs in the training set. Clearly, $A_{u|v} = A_u + A_v - A_{u\&v}$. We also use A_{v2u} to denote the number of actions propagated from v to u in the training set. We next define propagation of actions.

Definition 1 (Action propagation). *We say that an action $a \in \mathcal{A}$ propagates from user v_i to v_j iff: (i) $(v_i, v_j) \in E$; (ii) $\exists (v_i, a, t_i), (v_j, a, t_j) \in Actions$ with $t_i < t_j$; and (iii) $\mathcal{T}(v_i, v_j) \leq t_i$. When this happens we write $prop(a, v_i, v_j, \Delta t)$ where $\Delta t = t_j - t_i$.*

Notice that there must be a social tie between v_i and v_j, both must have performed the action after the moment in which their social tie was created. This leads to a natural notion of a propagation graph, defined next.

Definition 2 (Propagation graph). *For each action a, we define a propagation graph $PG(a) = (V(a), E(a))$, as follows. $V(a) = \{v \mid \exists t : (v, a, t) \in Actions\}$; there is a directed edge $v_i \xrightarrow{\Delta t} v_j$ in $E(a)$ whenever $prop(a, v_i, v_j, \Delta t)$.*

The propagation graph consists of users who performed the action, with edges connecting them in the direction of propagation. Observe that the propagation graph is a DAG. Each node can have more than one parent; it is directed, and cycles are impossible due to the time constraint which is the basis for the definition of propagation. Note that the propagation graph can possibly have disconnected components. In other words, the propagation of an action is just a directed instance (a flow) of the undirected graph G, and the log of actions $Actions(User, Action, Time)$ can be seen as a collection of propagations. When a user performs an action, we say that it is activated w.r.t. that action. Once a user activates, it becomes contagious and cannot de-activate. It may now influence all its inactive friends. The power to influence the neighbors is what we model as influence probability. The problem we tackle in this paper is how to learn influence probabilities among the users, by mining the available set of past propagations. Formally, we want to learn a function $p : E \rightarrow [0, 1] \times [0, 1]$ assigning to both directions of each edge $(v, u) \in E$ the probabilities: $p_{v,u}$ and $p_{u,v}$.

4. SOLUTION FRAMEWORK

In the following, we introduce the framework we adopt which is an instance of the General Threshold Model. Consider an inactive user u and the set of its activated neighbors S, and suppose that each neighbor $v \in S$ activates after v and u became neighbors. To predict whether u will activate, we need to determine $p_u(S)$, the joint influence probability of S on u. If $p_u(S) \geq \theta_u$, where θ_u is the activation threshold of user u, we can conclude that u activates. For ease of exposition, assume individual probabilities of influence between users are static, i.e., are independent of time. We do not need this assumption for our results. Since influence probabilities are meant for use in viral marketing [9] [8], our definitions must be consistent with the diffusion models used in these papers. These

[2]At the time of our submission, this paper was not yet published. We thank the reviewers for pointing it to us.

[3]For convenience, we assume social ties are never broken. This assumption is inessential.

papers typically assume the diffusion models are *monotone*, which says the function $p_u(S)$ should satisfy: $p_u(S) \leq p_u(T)$ whenever $S \subseteq T$. Moreover, it should be *submodular*, i.e., $p_u(S \cup \{w\}) - p_u(S) \geq p_u(T \cup \{w\}) - p_u(T)$ whenever $S \subseteq T$. There can be various ways to define $p_u(S)$. In this paper, for computational ease, we assume that the probability of various friends influencing u are independent of each other. Hence, the joint probability $p_u(S)$ can be defined as follows:

$$p_u(S) = 1 - \prod_{v \in S} (1 - p_{v,u}) \tag{1}$$

In the context of testing a learned model, we need to be able to compute and update the influence probabilities on the fly. That is, as new neighbors get activated, the joint influence probability needs to be updated. We should be able to compute $p_u(S \cup \{w\})$ incrementally without revisiting the neighbor set influence probabilities, i.e., solely in terms of $p_u(S)$ and $p_{w,u}$. This is not part of the requirement imposed by the diffusion models assumed for viral marketing. Thus, this is a desirable, but not mandatory property.

THEOREM 1. *The joint influence probability as defined in Eq. 1 is monotone and submodular. Besides, it can be updated incrementally if the individual influence probabilities $p_{v,u}$ are static.*

Proof. Let S be the set of neighbors of u that are active and suppose a new neighbor w of u gets activated. The new joint influence probability $p_u(S \cup \{w\})$ can be computed incrementally from $p_u(S)$ as follows.

$$\begin{aligned} p_u(S \cup \{w\}) &= 1 - (1 - p_{w,u}) * \prod_{v \in S} (1 - p_{v,u}) \\ &= 1 - (1 - p_{w,u}) * (1 - p_u(S)) \\ &= p_u(S) + (1 - p_u(S)) * p_{w,u} \end{aligned} \tag{2}$$

The monotonicity can be seen from Eq. 2. The difference $p_u(S \cup \{w\}) - p_u(S)$ is clearly non-negative as the domain of individual probabilities is $[0, 1]$. Similarly, submodularity can be shown as follows.

$$\begin{aligned} &p_u(S \cup \{w\}) - p_u(S) - p_u(T \cup \{w\}) + p_u(T) \\ &\quad = (1 - p_u(S)) * p_{w,u} - (1 - p_u(T)) * p_{w,u} \\ &\quad = (p_u(T) - p_u(S)) * p_{w,u} \geq 0 \end{aligned}$$

since, by monotonicity, $p_u(T) \geq p_u(S)$. □

User Influenceability. As mentioned in the introduction, there can be three reasons which prompt any user to perform an action. First, influence from friends and family members. Second, she is affected by some external event(s). And the last is that she is a very active user and is doing things without getting influenced by anyone.

In this work, we mainly focus on modeling and learning the influence propagation from neighbors. For some users, external influence plays a significant role and for others that is not the case. Users who are initiators of actions and who are more influenced by external factors are unpredictable or less influenceable. So, we define an *influenceability score* representing how influenceable a user is, as the ratio between the number of actions for which we have evidence that the user was influenced, over the total number of actions performed by the user. More precisely we define:

$$infl(u) = \frac{|\{a \mid \exists v, \Delta t : prop(a, v, u, \Delta t) \wedge 0 \leq \Delta t \leq \tau_{v,u}\}|}{A_u} \tag{3}$$

In the equation above, we can use any appropriate value for the parameter $\tau_{v,u}$. We propose to use the average time delay, defined as follows:

$$\tau_{v,u} = \frac{\sum_{a \in \mathbf{A}} (t_u(a) - t_v(a))}{A_{v2u}} \tag{4}$$

where $t_u(a)$ is the time when u performs a and $\mathbf{A}$ is the set of actions in the training data. We conjecture that users with a high value for $infl(u)$ may exhibit a high degree of being influenced by their neighbors compared to those with a low value for this metric.

Action Influenceability. We define the *influence quotient* for an action to distinguish between actions for which there is more evidence of influence propagation from the rest of the actions. More precisely, we define:

$$infl(a) = \frac{|\{u \mid \exists v, \Delta t : prop(a, v, u, \Delta t) \wedge 0 \leq \Delta t \leq \tau_{v,u}\}|}{\text{number of users performing } a} \tag{5}$$

We expect that for actions with high $infl(a)$, predictions (based on influence models) of user performing those actions will yield a relatively higher precision and recall values compared to other actions. We will revisit this in the experimental section.

5. MODELS

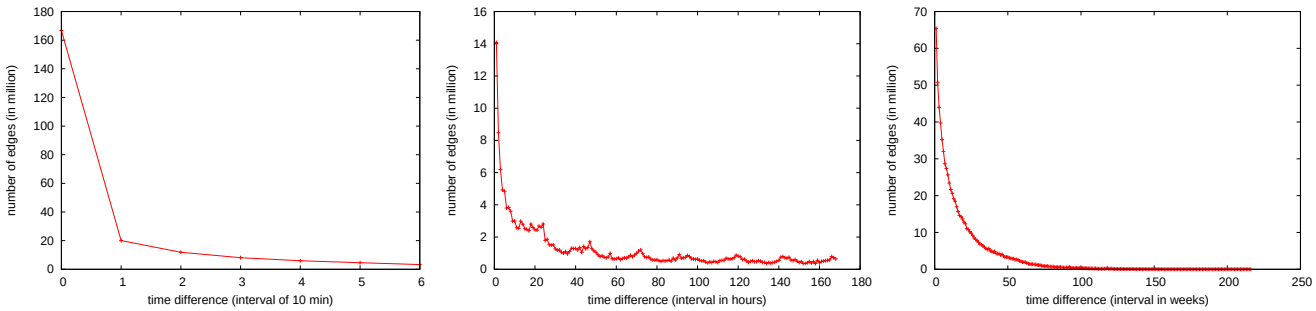
Recall from the previous section that we assume the probabilities of influence by individual neighbors of a user are independent. Thus, if we have a model for capturing individual influences, we can compute the joint influence using Eq. 1. Next, we propose 3 types of models to capture $p_{v,u}$, the probability with which u is influenced by its neighbor v. The first class of models assumes the influence probabilities are static and do not change with time. The second class of models assumes they are continuous functions of time. As a preview, it will turn out continuous time models are by far the most accurate, but they are very expensive to test on large data sets. Thus, we propose an approximation known as Discrete Time Models where the joint influence probabilities can be computed incrementally and thus efficiently. In all the models we propose, we also discuss how to learn estimates of various parameters from the training data set.

5.1 Static Models

These models are independent of time and are the simplest to learn and test. We present three instances of static models.

Bernoulli distribution. Under this model, any time a contagious user v tries to influence its inactive neighbor u, it has a fixed probability of making u activate. If u activates, it is a successful attempt. Each attempt, which is associated with some action, can be viewed as a Bernoulli trial. The Maximum Likehood Estimator (MLE) of success probability is the ratio of number of successful attempts over the total number of trials. Hence, influence probability of v on u using MLE is estimated as:

$$p_{v,u} = \frac{A_{v2u}}{A_v} \tag{6}$$



(a) (b) (c)

Figure 1: Frequency of common actions Vs. the time difference between two users performing actions. (a) during the first hour at a granularity of 10 minutes; (b) during the first week at hourly granularity (without considering the cases in which the time difference is less than one hour, i.e., the cases in (a)); (c) the rest of the dataset with weekly granularity

Jaccard Index. The Jaccard index is often used to measure similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. We adapt the Jaccard index to estimate $p_{v,u}$ as follows:

$$p_{v,u} = \frac{A_{v2u}}{A_{u|v}} \tag{7}$$

Partial Credits (PC). When a user u in a network is influenced to perform an action, it may be influenced by the combination of its neighbors who have performed the action before. Thus, it is reasonable to infer that each of these predecessors shares the "credit" for influencing u to perform that action. Suppose user u performs an action a at time $t_u(a)$ and S its set of activated neighbors such that $\forall v \in S, \mathcal{T}(v,u) \leq t_v(a) < t_u(a)$. Let $|S| = d$. Then, in the partial credits model, we give each of u's neighbors an equal credit $1/d$ for making u perform the action. In general, the credit given to user $v \in S$ who performed an action a before u can be defined as:

$$credit_{v,u}(a) = \frac{1}{\sum_{w \in S} I(t_w(a) < t_u(a))} \tag{8}$$

where I is an indicator function.

The notion of partial credit is orthogonal to whether we use Bernoulli or Jaccard as the base model. Thus, we have two additional models. In the *Bernoulli model with partial credit*, the influence probability is estimated as follows:

$$p_{v,u} = \frac{\sum_{a \in \mathbf{A}} credit_{v,u}(a)}{A_v} \tag{9}$$

where $\mathbf{A}$ is the set of actions in the training data. In the *Jaccard model with partial credit*, the influence probability is estimated as follows:

$$p_{v,u} = \frac{\sum_{a} credit_{v,u}(a)}{A_{u|v}} \tag{10}$$

In case of static models, the joint influence probability $p_u(S)$ as defined in equation 1 can be computed incrementally as stated in Theorem 1.

5.2 Continuous Time (CT) Models

In reality, influence probability may not remain constant independently of time. It is natural to expect that when a user first comes to see/hear of its neighbor(s) performing an action, it may feel the urge to explore it and that, with time, this urge (i.e., influence) may decay. To understand whether this is true in practice, we computed the number of actions that propagated between pairs of neighbors in Flickr and plotted it against the time that elapsed between them: see Figure 1. The figure shows the behavior at three levels of granularity – weeks, hours, and intervals of 10 minutes. In all cases, the data consistently shows an exponential decay behavior, confirming our intuition above.

Accordingly, we define $p^t_{v,u}$, the probability of v influencing its neighbor u at time t, as follows.

$$p^t_{v,u} = p^0_{v,u} e^{-(t-t_v)/\tau_{v,u}} \tag{11}$$

where $p^0_{v,u}$ is the maximum strength of v influencing u. In the exponential decay model, the maximum strength is realized right after v performing the action, i.e., when $t = t_v$. This maximum strength, $p^0_{v,u}$, can be estimated in exactly the same was as for the static models. Thus, we can have four variations of the continuous time model corresponding to the four static models discussed earlier. We omit the obvious detail. The parameter $\tau_{v,u}$ is called the *mean life time*. It corresponds to the expected time delay between v performing an action and u performing the same action. Once $p^t_{v,u}$ is defined, we can derive the joint probability of influence, $p^t_u(S)$, of u being influenced at time t by the combination of its active neighbors, can be derived exactly as for static models. More precisely, we have:

$$p^t_u(S) = 1 - \prod_{v \in S} (1 - p^t_{v,u}) \tag{12}$$

The parameter $\tau_{v,u}$ can be estimated as the average time delay in propagating an action from v to its neighbor u in the training set. Formally, it is defined as in equation 4.

In this class of continuous time models, the joint influence probability $p^t_u(S)$ changes as each time step as it is a continuous function of time. As a new neighbor activates and becomes contagious, there may be a sharp increase in $p^t_u(.)$ and then it starts decreasing again with time. Hence, the function $p^t_u(.)$ is a piecewise continuous function. Since the probability $p^t_{v,u}$ changes at each time step, the joint influence probability cannot be computed incrementally. Every time a new neighbor activates, we have to compute it from scratch. If the size of set S is d, then there would be at most d local maxima for the function. If $\max_t \{p^t_u(.)\} \geq \theta_u$, the activation threshold of u, we conclude that u activates.

In addition to predicting the activation state of an user, this class of models enables us to predict time at which the user is most likely to perform the action. The details appear in Section 6.3.

5.3 Discrete Time (DT) Models

As noted above, Continuous Time Models are not incremental in nature, hence they are very expensive in terms of run time required for testing. Therefore, in this section, we propose an approximation to Continuous Time Models, called *Discrete Time Models*. Here, we say that the influence of an active user v on its neighbor u remains constant at $p_{v,u}$ for a time window of $\tau_{v,u}$ after v performs the action. After that it drops to 0, i.e. a user v is contagious for u in the time interval $[t_v, t_v + \tau_{v,u}]$. It allows us to use the incrementality property established in Theorem 1. Here, the definition of S needs to be modified such that it contains only contagious neighbors of u. Hence, when a contagious neighbor w becomes non-contagious, we need to update $p_u(S)$ and it can be incrementally updated as follows.

$$p_u(S \setminus w) = \frac{p_u(S) - p_{w,u}}{1 - p_{w,u}} \tag{13}$$

Analogous to Static Models, there can be 4 variations of Discrete Time models depending on how the constant influence probability $p_{v,u}$ is estimated, i.e., using Bernoulli, Jaccard, or their particl credit variants. For brevity, we only give the equation for partial credits here as other cases are easier.

The Partial Credit definition (8) for discrete time should be modified as

$$credit_{v,u}^{\tau_{v,u}}(a) = \frac{1}{\sum_{w \in S} I(0 < t_u(a) - t_w(a) \leq \tau_{v,u})} \tag{14}$$

where I is the indicator function. The new influence probability $p_{v,u}$ can be computed as stated in equations 9 and 10 with the new definition of *credit*. As we show in the experiments section, these models provide an efficient yet effective approximation for Continuous Time Models.

6. ALGORITHMS

In this section, we present algorithms for learning the parameters of the various models proposed in the previous section, as well as algorithms for testing the learned models. One of the key aspects we pay attention to is efficiency of not just the training algorithms (which learn the parameters) but also that of the testing algorithms which apply the models on the test data and make predictions. As mentioned before, any model that enjoys the incremental property affords an efficient testing algorithm.

The input to these algorithms consists of a social graph together with an action log. We assume the action log is sorted on action-ids and tuples on an action are chronologically ordered. This allows the algorithms to process the data one action at a time. In practice, action log tends to be huge (from tens to hundreds of millions of tuples) so we optimize our algorithms to minimize the number of scans over the action log.

6.1 Learning the Models

As in any machine learning approach, the first step is to learn the parameters of a model. We note that *our algorithms are able to learn all the models simultaneously in no more than two scans of the (training sub-set of the) action log table*. Furthermore, to learn parameters for static and continuous time Models, the algorithm needs only one scan of the actions log. The overview is presented in Algorithm 1. To illustrate our algorithms, we will use a running example shown in Figure 2. Recall, action log is sorted on action-ids and then by time. The algorithm maintains the tuples for the current action a in a tree-based data structure *current_table* indexed on user-ids. As it reads a new tuple of the form (u, a, t_u) saying user u performs the action a at time t_u, we look into the *current_table* for those neighbors v of u, such that the link between u and v has been established before either of them performs a. E^{t_v} represents the set of edges in the social graph at time t_v. It is worth noting that we don't need to assume social ties are never broken, as long as we can efficiently find E^{t_v}. Since the data is sorted in chronological order, $t_v \leq t_u$.

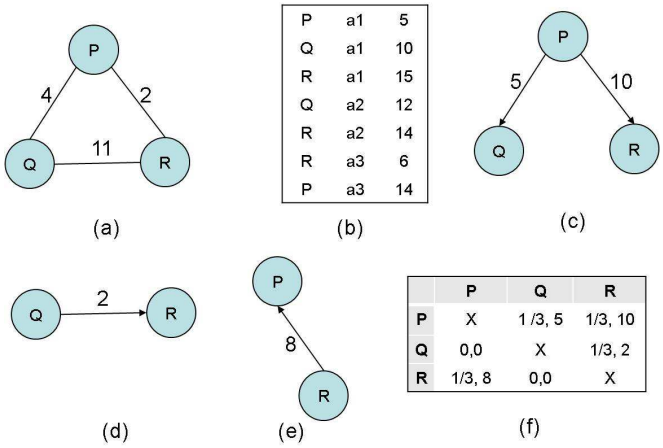


Figure 2: (a) Undirected social graph containing 3 nodes and 3 edges with timestamps when the social tie was created; (b) Action Log; (c) Propagation Graph for action a1 (d) Propagation Graph for action a2 (e) Propagation Graph for action a3 (f) Influence Matrix.

Algorithm 1 Learning - Phase1

```
1:  for each action a in training set do
2:    current_table = φ
3:    for each user tuple < u, a, t_u > in chronological order do
4:      increment A_u
5:      parents = φ
6:      for each user v : (v, a, t_v) ∈ current_table && (v, u) ∈
        E^{t_v} do
7:        if t_u > t_v then
8:          increment A_{v2u}
9:          update τ_{v,u}
10:         insert v in parents
11:       increment A_{v&u}
12:     for each parent v ∈ parents do
13:       update credit_{v,u}
14:     add (u, a, t_u) to current_table
```

Lines 1-6 of the algorithm are self-explanatory. The condition in line 7 ensures that we avoid the cases when the two users perform a at the same time as then it is questionable

whether propagation actually happened. Next, for all the interesting neighbors from which the action propagated, we update the required counts/parameters (lines 8-11).

As an example, Figure 2(a) shows a social graph containing three users P, Q and R with three edges among them. The edges are labeled with timestamps at which the two users became friends. The action log containing 3 actions $a1$, $a2$ and $a3$ is presented in Figure 2(b). Using the social graph and action log, the propagation graphs for actions $a1$, $a2$ and $a3$ are shown in Figure 2(c), (d) and (e). Edges are directed in Propagation Graphs and labeled with time taken to propagate the action. Note that even though both Q and R perform $a1$, $PG(a1)$ doesn't contain the edge from Q to R because when Q performed $a1$ at time 10, R was not in its neighborhood. They became friends at time 11. Define the influence matrix of a model to be an $(n \times n)$ matrix IM, with $IM[i,j] = (p_{i,j}, \tau_{i,j})$ or $IM[i,j] = (p^0_{i,j}, \tau_{i,j})$, depending on whether it's a discrete or continuous time model. Figure 2(f) shows the influence matrix containing parameters learnt for Continuous Time Model with Bernoulli as the base model for computing $p^0_{i,j}$. For instance, the entry $IM[P,Q]$ shows that $p^0_{P,Q}$ is 1/3 and $\tau_{P,Q}$ is 5.

For learning partial credits, we maintain the tree-based data structure *parents* containing the contagious neighbors from which the action has been propagated. Next, the algorithm goes through the *parents* list again and updates the partial credits as defined in Eq. 8 (lines 12-13). Finally, the tuple (u, a, t_u) from the action log is added to *current_table* (line 14). Notice that static models and continuous time models are learned in one scan.

To learn $infl(u)$, τ (learned from first scan) is needed and this requires a second scan of the action log. Similarly, learning parameters of discrete time models also requires τ beforehand. The second phase of the learning algorithm is described in Algorithm 2. The algorithm is very similar to Algorithm 1 except in Step 6 we require that $t_u - t_v \leq \tau_{v,u}$. Notice that $infl(u)$ is updated whenever we find at least one neighbor from which u is influenced.

Algorithm 2 Learning - Phase2

```
1:  for each action a in training set do
2:    current_table = φ
3:    for each user tuple < u, a, t_u > in chronological order do
4:      parents = φ
5:      for each user v : (v, a, t_v) ∈ current_table && (v, u) ∈
        E^{t_v} do
6:        if 0 < t_u − t_v < τ_{v,u} then
7:          increment A_{v2u}
8:          insert v in parents
9:      for each parent v ∈ parents do
10:       update credit^{τ_{v,u}}_{v,u}
11:     if parents != φ then
12:       update infl(u)
13:     add (u, a, t_u) in current_table
```

6.2 Evaluating the Models

An overview of the evaluation algorithm for Static Models is given in Algorithm 3. We assume the same sort order for the action log as for the training algorithms. We maintain a *results_table* with entries of the form $< u, p_u, perform_u >$ as we scan the action log, where the flag $perform_u$ represents whether the user u has actually performed the action in question or not: its value is 0 if u never performs the action but at least one of its neighbors does, is 1 if u performs it and at least one of its neighbors performs it before it, and is 2 if u is the initiator of the action in its neighborhood. p_u represents the probability of the user u performing the action given its neighbors who have already performed the action. At any instant, the *results_table* contains all the activated users for the current action and their neighbors.

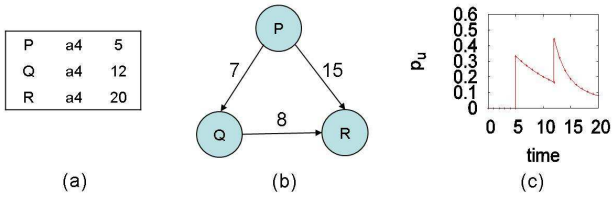


Figure 3: (a) Action Log; (b) Propagation Graph for action a4 (c) $p_R(.)$ w.r.t time for Continuous Time Model with Bernoulli.

As we scan the actions log and read a new tuple of the form $< v, a, t_v >$, we add v and all its neighbors to the *results_table* with the appropriate influence probability and $perform_v$ flag. It may be possible that v is already present in the table because one or more of its neighbors are already active. In that case, we just update the $perform_v$ flag to 1 (lines 4-5). If it is not present in the table, then that means it is the initiator of the current action in its neighborhood, hence the $perform_v$ flag should be set to 2 (lines 6-7). Similarly, some of its neighbors may already be present in the table and in that case, we update the influence probability of v over them incrementally (lines 8-10). At the end of reading all tuples for an action, the *results_table* contains all the users who are active or are neighbors of one or more active users.

We depict the performance of our models using ROC curves, which plot the true positive rate (TPR = TP/(TP+FN)) against the false positive rate (FPR = FP/(FP+TN)), where TP represents true positives, FN represents false negatives etc. The appropriateness of ROC curves over precision recall curves for binary classification has been recognized [12]. The closer the hump of the curve to the point $(0, 1)$ the better the performance. In our problem setting, we ignore all the cases when none of the user's friends is active, as then the model is inapplicable. Hence, we define TP as cases when user performs the action, at least one of its neighbors performs the action before it and model estimates it performs the action. FP is the number of cases when user doesn't perform the action, at least one of its neighbors performs action and model estimates it performs the action. Similarly, TN is the number of cases when user doesn't perform the action, and at least one of its neighbors performs the action and the model estimates it doesn't perform the action. Finally, FN is the number of cases when the user performs the action, at least one of its neighbors performs the action before it and the model estimates it doesn't perform the action.

After processing all the tuples for an action, the algorithm scan the *results_table* to update TP, FN, FP and TN (lines 13-17). It is straightforward to adapt this algorithm to evaluate Discrete Time Models. We omit the details here due to the lack of space.

In case of continuous time models, testing becomes complex. Algorithm 4 provides an overview. Here, we store in the *results_table* the time t_u at which user u performs the action a. Once the *results_table* is formed for an action (in lines 2-10), the algorithm iterates over all the entries in it.

Algorithm 3 Evaluate-Basic

```
1: for each action a in test set do
2:   results_table = φ
3:   for each user tuple < v, a, t_v > in chronological order do
4:     if v ∈ results_table then
5:       set perform_v flag to 1
6:     else
7:       add v to results_table with p_v=0 and perform_v=2
8:     for each user u : (v, u) ∈ E^{t_v} do
9:       if u ∈ results_table then
10:        update p_u incrementally as in Theorem 1
11:      else
12:        add u to results_table with appropriate p_u and
           perform_u=0
13:  for each entry < u, p_u, perform_u > in results_table do
14:    if (perform_u == 1 && p_u ≥ θ_u) it is TP
15:    if (perform_u == 1 && p_u < θ_u) it is FN
16:    if (perform_u == 0 && p_u ≥ θ_u) it is FP
17:    if (perform_u == 0 && p_u < θ_u) it is TN
```

For each entry $\langle u, p_u, perform_u, t_u \rangle$, it collects all the relevant neighbors and keeps them in chronological order in *sorted_parents* table (lines 11-14). Next, the algorithm tries to find the global maximum of joint influence probability of u performing the action w.r.t time. Whenever a new neighbor performs the action, the joint influence probability increases sharply and then starts decreasing again until another neighbor gets activated. So, if there are d neighbors who perform a, then the joint probability distribution would have (up to) d local maxima. To find the global maximum, we have to analyze all the local maxima (lines 15-18). If it is more than the threshold θ_u, then we conclude that u activates. The required metrics: TP, FN, FP and TN are updated accordingly (lines 19-23).

Algorithm 4 Evaluate-Complex

```
1: for each action a in test set do
2:   results_table = φ
3:   for each user tuple < u, a, t_u > in chronological order do
4:     if u ∈ results_table then
5:       set perform_u flag to 1
6:     else
7:       add u to results_table with p_u=0 and perform_u=2
8:     for each user v : (v, u) ∈ E^{t_u} do
9:       if v ∉ results_table then
10:        add v in results_table with p_v=0 and perform_v=0
11:  sorted_parents = φ
12:  for each entry < u, p_u, perform_u, t_u > in results_table do
13:    for each user v : perform_v! = 0, (v, u) ∈ E^{t_v} do
14:      add v to sorted_parents
15:    for each neighbor v_i ∈ sorted_parents list do
16:      compute p_u(t_{v_i}) at time t_{v_i} considering neighbors
         from v_1 to v_i
17:      if p_u(t_{v_i}) > p_u then
18:        update p_u
19:  for each entry < u, p_u, perform_u, t_u > in results_table do
20:    if (perform_u == 1 && p_u ≥ θ_u) it is TP
21:    if (perform_u == 1 && p_u < θ_u) it is FN
22:    if (perform_u == 0 && p_u ≥ θ_u) it is FP
23:    if (perform_u == 0 && p_u < θ_u) it is TN
```

As an example, consider the social graph and influence probabilities in Figure 2. Let us assume there is an action $a4$ in test set whose action log is given in Figure 3(a). The corresponding propagation graph is shown in Figure 3(b). Finally, the probability of R performing the action w.r.t. is plotted in 3(c).

6.3 Predicting Time

Time conscious models like CT and DT enable us not only to predict whether a user performs a particular action or not, but also to predict the time interval $[b, e]$ in which she is most likely to do it. We next explain how.

Whenever a new neighbor activates, there is a sharp increase in the joint influence probability which makes it a piecewise continuous function. We assume that user u is most likely to get activated in the first region where $p_u^t(.) \geq \theta_u$. In other terms, we take the first local maximum of the joint probability function which is not less than θ_u. We say the user has now entered into the contagious interval and can activate anytime. We label this time t as the left-bound of the interval b. For the right-bound e, we add to b the *half life period*[4] of the influence of v over u, or more specifically, $\tau_{v,u} * ln(2)$, where v is the neighbor of u by virtue of which u entered the contagious zone. Intuitively, by time $\tau_{v,u} * ln(2)$, half the actions that are propagated from w to u have indeed been picked up by u. Thus, we predict that given an action a that did propagate from v to u, by the half life period after v performs a, u would have performed a.

Since it would be complex to asses prediction accuracy w.r.t. an interval $[b, e]$ we decide for our experiments to predict an exact point in time. In viral marketing applications, tightness of lower bound is not critical, as in case the user performs the action early, it does not hurt. Hence, we decide to perform our experiments on the upper bound e.

7. EXPERIMENTAL EVALUATION

Dataset preparation. In our problem setting, we need both the social network and the actions log. We consider Flickr social network and we consider "joining a group" as the action[5]. We started with 6,2 millions users having 71 millions edges. We projected this graph on the subset of users who is a member of at least one group. This gave us 1,450,347 users with 40,562,923 edges among them.

This social graph has 34,766 connected components where the largest connected component consist of 1,319,573 users with 40,450,082 edges (99.72%). Rest of the components have less than 75 users, so we ignore them. Total number of tuples in action log after the filtering are 35,967,169. As in any machine learning approach, we split the dataset into a training and a test set. In our experiments, we split the dataset based on actions such that each action can appear completely either in training or test dataset.

Qualitative Evaluation. We compare the different models by means of ROC curves. Each point in ROC curve corresponds to one possible value of activation threshold θ_u which is same for all users. Figure 4 compares the four Static Models introduced in Section 5.1. Similarly, Figure 5 examines the 4 variants of Discrete Time (DT) models (Section 5.3). These figures show that Bernoulli is slightly better than Jaccard model and among two Bernoulli variants, Partial Credit wins by a small margin. In the rest of the section, we only use Bernoulli model to compare different classes of models.

Figure 6 shows the comparisons between the three different classes of models. It verifies the claim that time conscious models work far better than Static Models. Among time conscious models, CT and DT perform equally well.

[4] http://en.wikipedia.org/wiki/Half-life

[5] http://www.flickr.com/groups

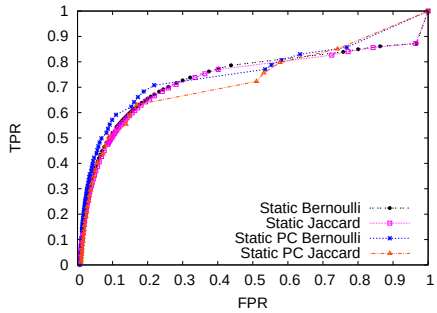


Figure 4: ROC comparisons of Static Models.

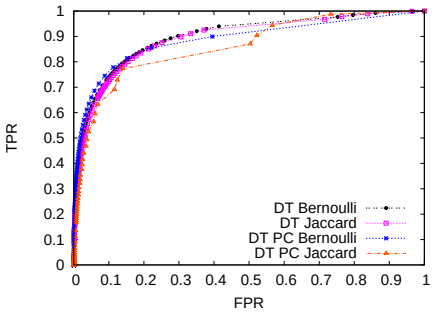


Figure 5: ROC comparisons of Discrete Time Models.

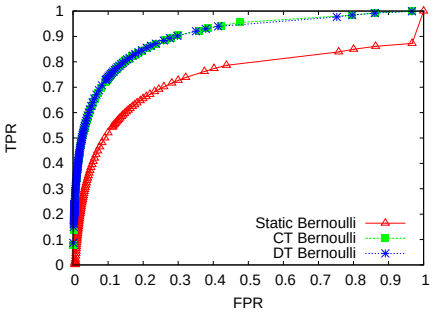
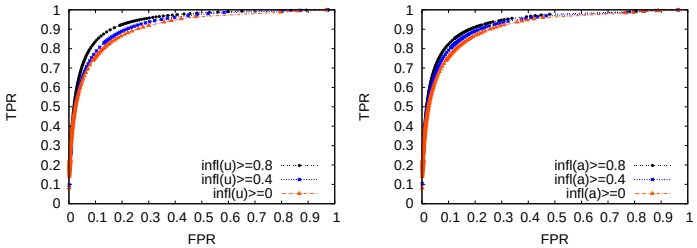
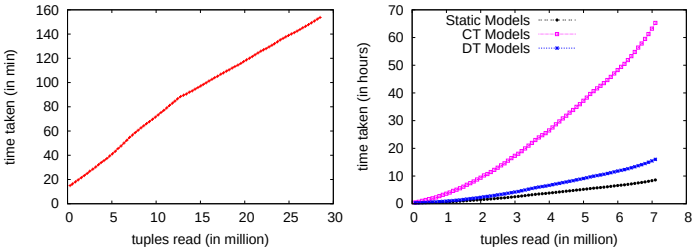


Figure 6: ROC comparisons of Static, CT, and DT models.



(a) (b)

Figure 7: CT Bernoulli model: ROC curves for different slices of users influenceability (a) and actions influenceability (b).



(a) (b)

Figure 11: (a) training runtime. (b) Testing runtime comparison of all 3 classes of models.

Figure 7 reports the ROC curves for different slices of users influenceability (a) and actions influenceability (b). The plots confirm the intuition that larger influenceability leads to an easier prediction of influence.

Predicting Time. While for DT models it is inexpensive to compute both lower and upper bounds of the contagious time interval, we found the upper bound computed is not tight. Hence, we conclude that while DT models are good in predicting the activation state of the users w.r.t. actions, they are not capable of predicting the time range in which the user is most likely to perform the action. Therefore in the following we focus on CT models (in particular Bernoulli).

In figures 8, 9 and 10, we show the power of CT models to predict the time by which users are likely to perform actions. The model attempts to predict the upper bound on time only when the user is active and prediction of activation state is correct, i.e. the TP cases. Figure 8 shows the root mean square error (RMSE) in days against the accuracy of predicting upper bound. Accuracy of time prediction is defined as the ratio between the number of cases when the prediction of upper bound by the model is correct, over the total number of cases it examines. In this plot we removed the 2.5% outliers both from the negative and positive side. The RMSE value revolves around 70-80 days. If we try to increase accuracy beyond 85%, RMSE increases sharply. In the subsequent figures, we choose an operating point for CT model corresponding to 82.5% TPR and 17.5% FPR. This is chosen as the intersection point of the ROC curve and diagonal line from [1,0] to [0,1]. At this point, Accuracy is 73% and RMSE is 75 days. In figure 9, X-axis is the error in predicting time and Y-axis is the number of times the CT model makes that error. It clearly shows that most of the time, the error in prediction is very small. Finally, Figure 10 shows the coverage of CT model. Here, a point (x, y) means that for y% of cases, the error is within $\pm$x. In particular, for 95% cases, the error is within 20 weeks.

Scalability evaluation. Figure 11 (a) and (b) shows the scalability of our algorithms. In the training phase, the runtime is a linear function of number of the tuples read. Memory usage (not reported in the figure) remains constant and independent of the number of tuples read, both in training and testing phases. Since Static and DT models are incremental in nature, they are far more efficient than CT models for testing. In conclusion, DT models achieves the same quality as CT models but much more efficiently.

8. CONCLUSIONS AND DISCUSSION

Previous works about influence propagation in social networks typically assume the social graph which is input to the problem has its edges labeled with probabilities of influence. In this paper we studied how to learn such probabilities form a log of past propagations. We proposed both static and time-dependent models for capturing influence, presented algorithms for learning the parameters of the various models and for testing the models. Our algorithms are optimized to minimize the scans over the action log, a key input to the problem of inferring probabilities of influence. This is significant since the action log tends to be huge. We ran an extensive set of experiments to test our learning algorithms. One of the highlights is that in addition to predicting whether a user will perform an action, we also observed that the predictions of our algorithms on users with a high influenceability score tend to have a high precision. In addition, we are able to predict, to within tight margins, the time by which an influenced user will perform an action after its neighbors have performed the action. In addition to demonstrating these aspects, our experiments also show that while testing the proposed continuous time model is very expensive, the discrete time model can be tested much

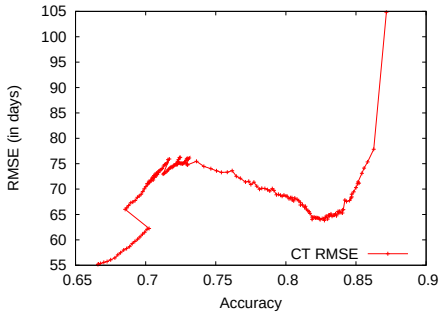


Figure 8: Root Mean Square Error (in days) vs Accuracy in predicting time for CT models.

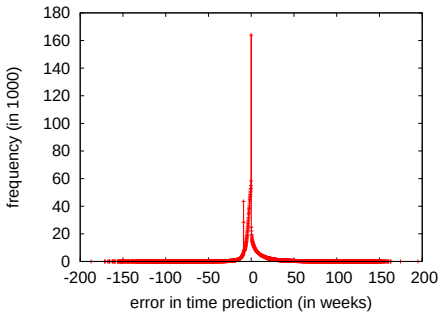


Figure 9: Distribution of error in time prediction.

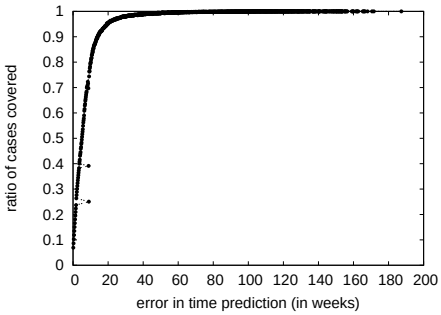


Figure 10: Coverage of CT models as a function of time prediction error.

more efficiently and yet can yield accuracy levels very close to that of the continuous time model.

Several challenges remain. While this work was motivated by the assumptions of viral marketing, it's interesting to consider the impact of this work in the reverse direction. For instance, viral marketing works essentially ignore the effect of time and assume edges have constant influence probabilities as labels. It is important to formulate and solve viral marketing taking into account the time varying nature of influence. Similarly, factoring in user influenceability and action influence quotient is important. Indeed, a user with low influenceability might "attenuate" some of the incoming influence from neighbors. The same user may be more influenced by neighbors on actions with high influence quotient than on other actions. Accounting for these phenomena in viral marketing is an interesting direction for future work. Last but not the least, learning optimal user activation thresholds would be an interesting future work.

9. REFERENCES

[1] A. Anagnostopoulos, R. Kumar, and M. Mahdian. Influence and correlation in social networks. In *Proc. of the 14th ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD'08)*.

[2] M. Cha, A. Mislove, and P. K. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *Proc. of the 18th Int. Conf. on World Wide Web (WWW'09)*.

[3] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *Proc. of the 15th ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD'09)*.

[4] P. Domingos and M. Richardson. Mining the network value of customers. In *Proc. of the 7th ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD'01)*.

[5] J. Golbeck and J. Hendler. Inferring binary trust relationships in web-based social networks. *ACM Trans. Internet Technol.*, 6(4):497–529, 2006.

[6] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *Proc. of the 13th Int. Conf. on World Wide Web (WWW'04)*.

[7] S. Hill, F. Provost, and C. Volinsky. Network-based marketing: Identifying likely adopters via consumer networks. *Statistical Science*, 21(2):256Ű–276, 2006.

[8] D. Kempe, J. M. Kleinberg, and É. Tardos. Influential nodes in a diffusion model for social networks. In *Automata, Languages and Programming, 32nd International Colloquium, (ICALP'05)*.

[9] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proc. of the Ninth ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*.

[10] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *Proc. of the 13th ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD'07)*.

[11] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsŮi. *Mathematical Programming*, 14(1):265–294, 1978.

[12] F. J. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth Int. Conf. on Machine Learning (ICML'98)*.

[13] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proc. of the Eighth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)*.

[14] K. Saito, R. Nakano, and M. Kimura. Prediction of information diffusion probabilities for independent cascade model. In *Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES'08)*.

[15] J. J. Samper, P. A. Castillo, L. Araujo, and J. J. M. Guervós. Nectarss, an rss feed ranking system that implicitly learns user preferences. *CoRR*, abs/cs/0610019, 2006.

[16] X. Song, Y. Chi, K. Hino, and B. L. Tseng. Information flow modeling based on diffusion rate for prediction and ranking. In *Proceedings of the 16th international conference on World Wide Web (WWW'07)*.

[17] X. Song, B. L. Tseng, C.-Y. Lin, and M.-T. Sun. Personalized recommendation driven by information flow. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR'06)*.

[18] M. Taherian, M. Amini, and R. Jalili. Trust inference in web-based social networks using resistive networks. In *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services (ICIW'08)*.

[19] J. Tang, J. Sun, C. Wang, and Z. Yang. Social influence analysis in large-scale networks. In *Proc. of the 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'09)*.

[20] D. Watts. Challenging the influentials hypothesis. *WOMMA Measuring Word of Mouth, Volume 3*, pages 201–211, 2007.

[21] D. Watts. Viral marketing for the real world. *Harvard Business Review*, pages 22–23, May 2007.

[22] D. Watts and P. Dodds. Influential, networks, and public opinion formation. *Journal of Consumer Research*, 34(4):441–458, 2007.

[23] C.-N. Ziegler and G. Lausen. Propagation models for trust and distrust in social networks. *Information Systems Frontiers*, 7(4-5):337–358, 2005.