Influence in social networks

Orestis Tourgelis Provatas

AM: 2013525

Supervisor:
Stamatopoulos Panagiotis
Dissertation committee:
Yiannis Cotronis
Hadjiefthymiades Stathes

National and Kapodistrian University of Athens
Department of Physics
Masters of Control and Computing

***Index terms*—** Social networks; influence diffusion model; influence maximization
***Λέξεις κλειδιά*—** Κοινωνικά δίκτυα; μοντέλα διάδοσης της επιρροής; μεγιστοποίηση της επιρροής

Περιεχόμενα

Περίληψη

Στην εργασία αυτή μελετήσαμε δύο κεντρικά προβλήματα σχετικά με την επιρροή στα κοινωνικά δίκτυα. Το πρώτο είναι το πρόβλημα της επιρροής στα κοινωνικά δίκτυα για το οποίο παρουσιάζουμε την μαθηματική θεμελείωση πίσω από τους δύο βασικότερους αλγορίθμους που συναντούμε στη βιβλιογραφία καθώς προχωρούμε και στην υλοποίησή τους. Επίσης παρουσιάζουμε και δύο βασικά μοντέλα για την διάδοση της επιρροής, το μοντέλο ανεξάρτητης διάδοσης και το μοντέλου γραμμικού κατωφλίου. Η συνάρτηση επιρρής και στα δύο αυτά μοντέλα παρουσιάζει την σημαντική ιδιότητα submodularity το οποίο εξασφαλίζει την αποδοτικότητα του greedy αλγόριθμου.

Το δεύτερο πρόβλημα που μελετήσαμε είναι το πρόβλημα της μάθησης των πιθανοτήτων επιρροής από δεδομένα πραγματικών κοινωνικών δικτύων. Υλοποιήσαμε τους αλγορίθμους που προτείνονται στη βιβλιογραφία τους οποίους χρησιμοποιήσαμε για να υπολογίσουμε της πιθανότητες επιρροής και να αποτιμήσουμε την ακρίβεια πρόβλεψης των διάφορων μοντέλων. Χρειάστηκαν μικρές τροποποιήσεις στους αρχικούς αλγόριθμους για την προσαρμογή τους σε κατευθυνόμενους γράφους. Επίσης δημιουργήσαμε μία βιβλιοθήκη υλοποιημένη σε ruby η οποία μετρατρέπει τα δεδομένα του github archive σε κατάλληλη μορφή αξιωποιήσημη από τους αλγορίθμους. Κατασκευάσαμε λοιπόν ένα καινούργιο σύνολο δεδομένων κατάλληλο για ανάλυση της επιρροής.

Τρέξαμε τους αλγορίθμους για την μάθηση της επιρροής στα δεδομένα των κοινωνικών δικτύων digg και github. Επιβεβαιώσαμε την αποτελεσματικότητα των μοντέλων στην πρόβλεψη της διάδοσης των likes/stars μέσα στο δίκτυο. Ωστόσο σε αντίθεση με την βιβλιογραφία δεν διαπιστώσαμε κάποια βελτίωση με την προσθήκη του χρόνου σαν παράγοντα στην συνάρτηση μοντελοποίησης της πιθανότητας επιρροής. Παρατηρήσαμε μία μικρή βελτίωση στην προβλεψιμότητα όταν χρησιμοποιήσαμε μόνο τους χρήστες για τους οποίους είχαμε ενδείξεις επιρροής πάνω από ένα δεδομένο κατώφλι.

Επίσης προτείνουμε μία καινούργια προσέγγιση στον υπολογισμό της πιθανότητας επιρροής εισάγωντας την διακρισή ανάμεσα σε κάποιες ενέργειες των χρηστών. Αντιμετωπίσαμε την ενέργεια της δημιουργίας μιας δημοσίευσης σαν κάτι το οποίο θα είχε διαφορετική επίδραση στους χρήστες από την ενέργεια του like σε

μία δημοσίευση. Η βελτίωση αυτή δεν ήταν ιδιαίτερα εμφανής στα roc curves που χρησιμοποιούνται στην βιβλιογραφία αλλά ήταν αρκετά εμφανής στα precission recall curves που θεωρούνται καλύτερες για την εκτίμηση της απόδοσης όταν τα δεδομένα μας δεν είναι ισοκατανεμημένα.

Abstract

In this thesis we explored two main problems regarding influence in social networks. The one is the influence maximization problem and the main approximation algorithms that can be found throughout literature. In the same context we also present two different models for the information/in- fluence diffusion in the social networks Independent Cascade and Linear Threshold models. Under both of these models the influence function re- mains submodular which ensures the approximation efficiency of the greedy algorithm. We provide a robust python implementation for aforementioned models, the greedy algorithm, as well as an optimized version (lazy-greedy) which exploits the submodular property of the influence function to reduce computation steps.

The second problem regarding influence, is how we can learn these influ- ence probabilities that we take for granted in the first problem, in a real social network. We proceed in the implementation of the algorithms proposed in the literature which we use to calculate the influence probabilities and eval- uate the accuracy of the various proposed models. We have implemented minor modifications to the algorithms to adapt them to a directed network instead of the undirected initial implementation. We also create a ruby gem to parse github archive events into mysql tables, digestible from the learning and evaluating algorithms, producing a new social network dataset.

Running the learning algorithms over the action logs of both digg and github social networks we managed to confirm the findings in literature regarding the predictability of the proposed models. We did not detect any improvement though using time conscious models , which are a lot more expensive computationally compared to the static models. We found a minor improvement in the digg social network when restricting our evaluation only on users with influenceability above a certain threshold, a finding that is not reproducible on github dataset.

We proposed a novice approach in calculating influence probabilities by distinguishing the effect of different types of actions. The new model did not exhibit any improvement in roc curves but there was a significant improve- ment in precision recall curves which might be a better evaluation method for this specific problem due to the fact that our dataset is unbalanced.

Κεφάλαιο 1

Introduction

Online social networks have been around for about 15 years. They gradually became part of our day to day life, and the extend to which they influence our lives is beyond doubt nowadays. Social networks are the way we find new friends and the way we learn news about the world. They also give us the ability to express our opinion as well as our support or disapproval (like or dislike) on the events happening in the context of a network.

Online social networks enabled the spread of information in a global scale. Phenomena such as virality became possible and quite interesting from a marketing as well as a scientific perspective. It is because of the emergence of online social networks that we are now able to study these phenomena on large datasets of social network activity.

For the purposes of this thesis we perceive a social network by its mathematical representation as a graph. Based on the action of a user following another user in the network we can construct a directed graph. We focus mainly on the phenomenon of influence, which a field of a very active research in the last 10 years. An example of influence would be when a user in a network creates a new post and some of her/his social connections, in a directed network these would be the users following her/him, like or dislike the post. In this case we may assume that the users that liked the article were influenced and their action may cascade further through their connections.

Looking into some examples of social influence we can find in the literature. A famous study by Christakis and Fowler[12] was based on the medical records of 12,000 patients. They constructed an offline social network based on these records, with different types of relationships between the users including friendship, sibling, spouse etc. Their aim was to detect a correlation between non-infectious health conditions and the neighbors of a user. Among other results they found that having an obese friend makes an individual 171% more likely to be obese compared to a randomly chosen person. This of course may be an effect of homophily rather

than influence.

Another famous example is the Hotmail phenomenon demonstrated by Hugo and Garnsey [13]. In the early 1990s Hotmail was a relatively small e-mail service provider. In order to boost their brand they started appending in each mail the phrase ”Join the world’s largest e-mail service with MSN Hotmail. http://www.hotmail.com”. The effect of this appended message was that in 18 months Hotmail became the number one e-mail provider with 8 million users.

There are various applications of the study of information and influence propagation, the most obvious of which is viral marketing. The goal in such context would be to identify a small number of nodes in a network that would maximize the spread of information. This is the field of influence maximization which is very interesting from algorithmic and mathematical perspective. The main ideas of this field will be presented in the following chapter.

Κεφάλαιο 2

Influence maximization

2.1 Introduction

In this chapter we present and examine the literature regarding the problem of influence maximization. Furthermore we will delve inside the logic of the basic models and algorithms through which we are able to provide an effective approximate solution to the problem of influence maximization. The models that we examine take as granted the monotonous increase of a node's tendency to activate as it's neighbors are. They also focus on the gradual condition where a node may only be activated and not the opposite. So to put it in simple words: as more of node's u in-neighbors are being activated, the activation probability of node u increases, and in the case of activation the probability that it's neighbors will be activated increases as well.

2.2 Information diffusion models

To approach the problem of influence maximization we need to come up with a model for the information diffusion that will simulate the real life phenomenon. The two most popular such models are the *Independent Cascade model* and the *Linear Threshold model*.

In both models mentioned the influence probability function holds two important properties: *submodularity* and *monotonicity*. The definition of *submodular functions* follows:

Definition 2.2.1. A function f is said to be submodular if the following condition holds:

$$f(S \cup \{u\}) - f(S) \leq f(T \cup u) - f(T),\ where\ S \subseteq T. \tag{2.1}$$

In the context of graphs and the influence probability function this means that the smaller the set of activators that are already selected the biggest the contribution of a newly selected node. This property allows for a good approximation solution as has been shown by Nemhauser et al.[3].

2.2.1 Independent Cascade model

This model was first introduced by Goldenberg et al. [4,5] to model viral marketing and the inspiration behind it came from the field of interacting particle systems. Starting from an initial set of nodes which we name *Seed set*, every node in this set has a probability to activate it's neighboring nodes. The activation process is stochastic and each attempt is independent from any other. Through this process new nodes are added in the active set which may activate new nodes.

Definition 2.2.2. Given a graph $G(V, E)$ and a node u, we define as reachable set X_u^E the set of nodes that it is possible to be reached from node u through the edges of set E.

Based on this definition we will describe the way independent cascade model works. We may see the activation process of each user as a coin toss. If the result is positive we regard the user as activated. And since each activation attempt is independent from every other attempt we can use the *principle of deferred decisions* and perform the coin toss at once. So from the initial graph we can take the activated graph that will contain only the active subset of the edges.

Definition 2.2.3. Independent Cascade Model

Given a graph $G(V, E)$ and the influence probability for each edge $p\{e\}_{e\in E}$. For a set of independent and uniformly distributed random variables in $[0, 1]$, $\{U_e\}_{e\in E}$ we define the set of active edges as $I = \{e \in E : p_e \geq U_e\}$. Independent Cascade Model for graph G and probabilities p defines for any initial seed set S to final set of active nodes as $A_I(S) = \cup_{u\in S} X_u^I$

Set X_u^I can be understood as the influence set of node u in the context of the random activation of edges I. We consider the set of influence of the seed set S as the union of influence sets of all nodes included in S.

What follows is the proof of the important property of submodularity we defined earlier.

Theorem 1. Submodularity in ICM

The function $\sigma(S)$ that gives total influence for a seed set S is submodular.

Απόδειξη. The expression that gives us the total influence is:

$$\sigma(S) = \sum_{i \subseteq E} P(I = i)|A_i(S)| \tag{2.2}$$

The meaning of the above expression is that the total influence is the sum over all subsets i of the product of the probability to get a specific set I with the number of activated nodes. It is easy to prove that any linear combination of submodular functions is also submodular. So we just need to prove that $|A_i(S)|$ is submodular.

$$\begin{aligned} f_i(S \cup \{u\}) - f_i(S) &= |A_i(S) \cup X_u^i| - |A_i(S)| \\ &= |X_u^i| - |A_i(S) \cap X_u^i| \\ &\geq |X_u^i| - |A_i(T) \cap X_u^i| \\ &= |A_i(T) \cup X_u^i| - |A_i(T)| \\ &= f_i(T \cup u) - f_i(T) \end{aligned}$$

We used the formula $|A \cup B| = |A| + |B| - |A \cap B|$ and the fact that $A_i(S)$ is monotone.

□

2.2.2 Linear Threshold model

In this model each node u in the network is influenced by each in-neighbor w with a weight $b_{u,w} \in [0, 1]$. Each node has a threshold θ_u in the interval $[0, 1]$. This threshold represents the total weight that needs to be imposed in this node in order to activate.

The diffusion process takes place in discrete steps. In step t every node that has been activated in the previous step stays activated. New nodes are activated if the following condition holds:

$$\sum_{w \to u, w \ active} b_{u,w} \geq \theta_u \tag{2.3}$$

The threshold for each node is selected uniformly at random. This is how we model the lack of knowledge for the threshold. At the end we will need to take the average price over all the values of the threshold.

The proof that influence function is submodular over this model is based in a more general version called *Triggering Model*. We will just provide its definition:

Definition 2.2.4. Triggering Model

Each node in the graph selects a triggering set at random from the set of its in-neighbors. Initially we set the seed set as activated. For the rest of the nodes if one of nodes from its triggering set gets activated he gets activated too.

It can be shown for that for any instance of the *Triggering Model* influence function is submodular, and also the reduction of *Linear Threshold Model* into it.

2.2.3 Number of simulations

An important note on the previous sections dedicated to the diffusion models is the following. Based on *Chernoff-Hoeffding bounds theorem* we can determine the number of simulations that we need to run in order the results we get are within acceptable limits from the real value:

Proposition 1. If the diffusion process with a seed set S is repeated independently at least

$$\Omega(\frac{n^2}{\epsilon^2}\ln(1/\delta)) \tag{2.4}$$

times, then the mean number of activated nodes we get from these repetitions is a $(1 \pm \epsilon)$-approximation of $\sigma(S)$ with probability at least $1 - \delta$.

2.3 Hill-climbing algorithm

In the diffusion models we described previously submodularity is an important property because it can be used to provide a good approximation for the solution of the *Influence maximization problem*. More precisely we know that locally optimal solutions ensure a good final spread, thus starting from an initially empty set in each iteration we will add in the activated set the node that has the maximum mean contribution across all simulations.

Algorithm 1 Hill-climbing algorithm

1: **procedure** hill-climbing($G(V, E)$, $\{p_e\}_{e \in E}$, k)
2: $S_0 = 0$
3: **for** $i = 0$ **to** k **do**
4: $s_i = argmax_{u \in S_i}[\sigma(S_i \cup \{u\} - \sigma(S_i)]$
5: $S_{i+1} = S_i \cup \{s_i\}$
6: **return** S_k

The term $argmax$ refers to the node that we need to use in the following term in order to maximize. What is not determined here is the way in which $argmax$ will be calculated. This may be approximated using Monte Carlo simulations on the diffusion models we presented earlier.

2.3.1 Efficiency of the Hill-climbing algorithm

In this section we will show that the $hill - climbing$ algorithm ensures that the seed set it returns will give us influence greater than or equal to the $(1 - 1/e)$ of the influence of the optimal set. It is important to note that the term $(1 - 1/e)$ arises in the limit that seed set's size goes to infinity. In order to be able to prove that the $hill - climbing$ algorithm has the efficiency we described earlier we need to walk through some intermediate steps, and definitions.

Lemma 1. Telescoping

For a submodular function f, and for every set A and $B = \{b_1, \ldots, b_k\}$ the following holds:

$$f(A \cup B) - f(A) \leq \sum_{i=1}^{k} [f(A \cup b_i) - f(A)] \tag{2.5}$$

Απόδειξη. We initially define the sets $B_0 = , B_1 = \{b_1\}, \ldots, B_i = \{b_1, \ldots, b_i\}$, or in a recursive formula $B_i = B_{i-1} \cup \{b_i\}$

$$\begin{aligned} f(A \cup B) - f(A) &= \sum_{i=1}^{k} [f(A \cup B_i) - f(A \cup B_{i-1}] \\ &= \sum_{i=1}^{k} [f(A \cup B_{i-1} \cup \{b_i\}) - f(A \cup B_{i-1})] \\ &\leq \sum_{i=1}^{k} [f(A \cup \{b_i\}) - f(A)] \end{aligned}$$

□

Definition 2.3.1. Marginal Increments

Given a set S_{i-1} marginal increment in step i id defined as

$$\delta_i = f(S_i) - f(S_{i-1}) = max(f(S_{i-1} \cup \{u\}) - f(S_{i-1}) \tag{2.6}$$

Lemma 2. Accretion

Let S_i be the set that $hill - climbing$ algorithm is giving us after step i, and a set T of size k. Then it holds:

$$f(S_{i+1}) \geq (1 - \frac{1}{k}) f(S_i) + \frac{1}{k} f(T) \tag{2.7}$$

Απόδειξη.

$$\begin{aligned} f(T) - f(S_i) &\leq f(S_i \cup T) - f(S_i) \\ &\leq \sum_{j=1}^{k} [f(S_i \cup \{t_i\}) - f(S_i)] \\ &\leq \sum_{j=1}^{k} \delta_{i+1} = k\delta_{i+1} \end{aligned}$$

The first step is based on the monotonicity of f. In the last step replacing with $\delta_i = f(S_{i+1}) - f(S_i)$ we get (2.7).

□

Now we are able to prove the following theorem.

Theorem 2. Efficiency of approximation
The $hill - climbing$ algorithm returns a set $\tilde{S}$ for which the following holds:

$$\sigma(\tilde{S}) \geq (1 - \frac{1}{e})\sigma(S^*) \tag{2.8}$$

where S^* is the optimal seed set.

Απόδειξη. First we will show the following:

$$f(S_i) \geq [1 - (1 - \frac{1}{k})^i] f(T) \tag{2.9}$$

We can show this using induction. The following is true $f() \geq 0$.

$$\begin{aligned} f(S_{i+1}) &\geq (1 - \frac{1}{k}) f(S_i) + \frac{1}{k} f(T) \\ &\geq (1 - \frac{1}{k})[1 - (1 - \frac{1}{k})^i] f(T) + \frac{1}{k} f(T) \\ &= [1 - (1 - \frac{1}{k})^{i+1}] f(T) \end{aligned}$$

the first step is based on (2.7), and the second on the assumption of induction. In the case where $i = k$ and $T = S^*$ we get:

$$f(S_k) \geq [1 - (1 - \frac{1}{k})^k] f(S^*)$$

and in the limit $k \to \infty$:

$$\lim_{k \to \infty} (1 - \frac{1}{k})^k = (1 - \frac{1}{e})$$

□

2.4 Lazy Evaluation

In the previous sections we have presented the initial approach in the problem of influence maximization. The algorithms and models presented there are not easily used in big datasets. Various performance improvements on the greedy algorithm have come to surface in the years following the publication from Kempe et al.[6]. Improvements have also been presented in the models used by the algorithms.

The lazy greedy algorithm is a well known optimization technique first proposed by Minoux[7]. The main idea is to avoid calculations that are not necessary. Let f be the submodular and monotone function we aim to maximize. Let $f(u|S)$ be the marginal gain of adding element u to set S, $f(u|S) = f(S \cup \{u\}) - f(S)$. In the i-th iteration of the greedy algorithm, we have selected a set S and the algorithm evaluated $f(w|S)$ for $w \in V \setminus S$. Let's say that in a previous iteration of the algorithm the selected set is $S' \subset S$ and we have evaluated $f(x|S')$ for $x \in V \setminus S$ where $f(x|S') \leq f(w|S)$. Then by the submodularity property of function f we get $f(x|S) \leq f(x|S') \leq f(w|S)$. This means that there is no need to evaluate $f(x|S)$ since we know that this is going to be less or equal to $f(w|S)$.

Algorithm 2 Lazy greedy algorithm

```
1:  procedure LazyGreedy(k, f)
2:      initialize S = ; priority queue Q = ; iteration = 1
3:      for i = 1 to n do
4:          u.mg = f(u|); u.i = 1
5:          insert u into Q with u.mg as the key
6:      while iteration ≤ k do
7:          extract max element u of Q
8:          if u.i = iteration then
9:              S = S ∪ {u}; iteration++
10:         else
11:             u.mg = f(u|S); u.i = iteration
12:             re-insert u into Q
13:     return S
```

The lazy-greedy algorithm can be implemented with a priority queue. For each element we hold two values, $u.mg$ and $u.i$, the first one designates the marginal gain we would get if we added the element to our set, and the second is the iteration that this marginal gain was evaluated. In the first loop the marginal gain for every element is evaluated for $S = $. Then in each iteration we get the element with the maximum marginal gain. If the $u.i$ is equal to $iteration$ we select it as the next element. If not this means that $u.mg$ has been evaluated in a previous iteration

with a different selected set S. So we update the $u.mg$ and we set the $u.i$ as the current iteration.

Κεφάλαιο 3

Learning influence probabilities

3.1 Introduction

In the previous section we have described how to maximize influence in a social network by selecting the correct distinct nodes from the network for a given number of seeds. The big question that remains unanswered here is: How do we get these influence probabilities from one node to its out-neighbors. In this section we describe the algorithms through which we can learn influence probabilities between social connections. These algorithms rely on the knowledge of the network on a given time and the actions each user performs. This chapter is mainly based on the work by Goyal et al.[2].

3.2 Problem description and general approach

In order to learn the influence probabilities among the user in a social network we will need to have some records of their actions, in order to be able to determine whether a user has been influenced by its neighbors. This is what we will call *action_log*. So in order to proceed we will need to have knowledge about the network as well as an *action_log*.

We will use various probabilistic models of influence, which will need to be compatible with the assumptions we have described previously, such as $submodularity$ and $incrementality$.

In this chapter, apart from the learning algorithms, we will also describe evaluation algorithms to test the efficiency of our models. These models not only predict that a user will or will not perform an action but also predict the time in which the action will occur.

3.2.1 Formal problem definition

We have an undirected graph $G = (V, E, T)$ that represents the social network. V is the set of users, an edge $\in E$ represents the social connection between the users and $T : E \rightarrow \mathbb{N}$ is a label on each edge with the timestamp at which these users where connected. We also have an *action_log* which contains tuples of the form (u, a, t_u), which means that user u performed action a at time t_u. We assume that all users in the *action_log* are present in the social graph, and that each user performs an action only once.

We denote with A_u the number of actions performed by user u, with $A_{u\&v}$ the actions that both users u and v performed, with $A_{u|v}$ the actions that either one of them performs. Moreover A_{u2v} denotes the number of actions that we believe propagated from user u to v. As propagation we define:

Definition 3.2.1. Action propagation

We will say that an action $a \in A$ propagates from user u_i to u_j iff:
(i) $(u_i, u_j) \in E$,
(ii) $\exists (u_i, a, t_i), (u_j, a, t_j) \in Actions\ where\ t_i < t_j$,
(iii) $T(u_i, u_j) \leq t_i$. When these conditions hold we write $prop(a, u_i, u_j, \Delta t)$ where $\Delta t = t_j - t_i$.

As we can see in the previous definition there must exist a social tie between two users that perform an action before either one of them performed the action in order to consider it as a propagated action.

Definition 3.2.2. Propagation graph

For each action a, we define a propagation graph $PG(a) = (V(a), E(a))$. $V(a) = \{v | \exists\ t : (v, a, t) \in Actions\}$; there is a directed edge $v_i \rightarrow v_j \in E(a)$ if $prop(a, u_i, u_j, \Delta t)$.

The propagation graph is a directed graph, with edges that connect the users in the direction of propagation as indicated by the time constraints. Also the propagation graph is a directed acyclic graph and may contain disconnected components.

3.3 Mathematical background

The algorithms and the models we aim to discuss here are based on the *General Threshold Model*. In this model at a given moment a user is either active or inactive. Each users tendency to activate increases monotonically as more neighbors become active. As time unfolds more neighbors of user u may activate until user u activates too. Each node has a monotone activation function from the set of its neighbors

to $[0, 1]$, and an activation threshold θ_u. A node becomes active at time $t + 1$ if $f_u(S) \geq \theta_u$ where S is the set of active neighbors at the time.

Let's assume that we have an inactive user u and its active neighbors S. Also let's assume that each neighbors $v \in S$ of u became active after v and u were connected. In order to predict whether u will become active we need to estimate the joint influence probability of S towards u. If $p_u(S) \geq \theta_u$ where θ_u is the activation threshold of u, then we can say that u will activate.

We will assume that the probability with which each of the neighbors of u influences u is independent of one another. So we can define the joint influence probability as:

$$p_u(S) = 1 - \prod_{v \in S}(1 - p_{v,u}) \tag{3.1}$$

In order to be able to evaluate a learned model we need to be able to update the influence probability towards a user on the fly. We should be able to compute $p_u(S \cup \{w\})$ incrementally using only $p_u(S)$ and $p_{w,u}$.

Theorem 3. Monotonicity, Submodularity, Incrementality

The joint influence probability as defined in Eq. (3.1) is monotone and submodular. Besides it can be updated incrementally if the individual influence probabilities $p_{v,u}$ are static.

Απόδειξη. Let S be the set of active neighbors of u and suppose an 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:

$$\begin{aligned} p_u(S \cup \{w\}) &= 1 - (1 - p_{w,u}) * \prod_{v \in S}(1 - p_{v,u}) \\ &= 1 - (1 - p_{w,u}) * (1 - p_u(S)) \\ &= p_u(S) + (1 - p_u(S)) * p_{w,u} \end{aligned} \tag{3.2}$$

The monotonicity can be seen from Eq. (3.2).

$$p_u(S \cup \{w\}) - p_u(S) = (1 - p_u(S)) * p_{w,v}$$

So as can be seen the effect of adding a new node in the active set is always greater or equal to zero.

And we can also show that submodularity holds:

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

since $p_u(T) \geq p_u(S)$ because of monotonicity.

□

Influenceability. There can be many reasons for which a user performs an action. In order to be able to distinguish between users that are being influenced and users that are driven by external factors we will define an influenceability score. This score is going to be ratio of actions for which we have evidence that the action of the user was influenced by its local network, over the total number of actions performed by the user. To formulate this mathematically:

$$infl(u) = \frac{|a|\exists\, u, \Delta t : prop(a, v, u, \Delta t) \wedge 0 \leq \Delta t \leq \tau_{v,u}|}{A_u} \tag{3.3}$$

In the equation above $\tau_{v,u}$ may be defined as the average time delay:

$$\tau_{v,u} = \frac{\sum_{a \in \mathbf{A}} (t_u(a) - t_v(a))}{A_{v2u}} \tag{3.4}$$

where $t_u(a)$ is the time when u performs action a and **A** the set of actions in the training set.

In a similar manner we may define **action influenceability**.

$$infl(a) = \frac{|u|\exists\, u, \Delta t : prop(a, v, u, \Delta t) \wedge 0 \leq \Delta t \leq \tau_{v,u}|}{number\ of\ users\ performing\ a} \tag{3.5}$$

This metric is important and we will use it in evaluating the models. We would expect that actions with higher influenceability value will be easier to be predicted by our models and thus we will get higher precision and recall values compared to other actions.

3.4 Models

We will present three different kind of models. The first class of models assumes that influence probabilities are static and do not change over time. The second class of models sees probabilities as continuous functions of time. The third type of models are essentially a computationally efficient approximation of the second class of models.

3.4.1 Static models

Bernoulli. In this model we consider an active user v which at any given time tries to activate its neighbor u, and has a fixed probability to achieve it. Each

attempt can be viewed as a Bernoulli trial thus influence probability of v on u is given by:

$$p_{v,u} = \frac{A_{v2u}}{A_v} \tag{3.6}$$

We should remark here that the previous definition as well as the following definitions, avoid to incorporate the fact that not all of the actions of user v should be included in the calculations. Formally we should only include the actions performed by user v after the creation of the social link $v \rightarrow u$.

Jaccard index. This is used to measure the similarity between sample sets. It is defined as the fraction of the size of intersection over the size of union of the sample sets.

$$p_{v,u} = \frac{A_{v2u}}{A_{u|v}} \tag{3.7}$$

Partial Credits (PC). When a user in a network performs and action this could mean that he has been influenced by several of his friends/neighbors. So it would be logical to attribute this event evenly to its previously activated neighbors. So if the set of previously activated neighbors of u is of size $|S| = d$ we should attribute an equal credit $1/d$ to all those neighbors.

$$credit_{v,u}(a) = \frac{1}{\sum_{w \in S} I(t_w(a) < t_u(a))} \tag{3.8}$$

Function I is an indicator action, meaning it returns 1 if the condition holds and 0 otherwise. With partial credits we could be using either Bernoulli or Jaccard as the base model. So *Bernoulli with partial credits* would be:

$$p_{v,u} = \frac{\sum_{a \in \mathbf{A}} credit_{v,u}(a)}{A_v} \tag{3.9}$$

And the *Jaccard model with partial credits*:

$$p_{v,u} = \frac{\sum_{a \in \mathbf{A}} credit_{v,u}(a)}{A_{u|v}} \tag{3.10}$$

3.4.2 Continuous Time Models (CT)

In the previous models we made the assumption that the influence probability is not changing with time. But in reality we would assume that when our local network starts to activate around us it would be more probable to be influenced by it. In contrast if time passes since the last activation of any or our neighbors we would expect that it is not very probable for us to activate. This phenomenon has

been validated by observations on flickr network by Goya et al. [2]. This study also showed that influence decays exponentially with time. We may define probability that user v influences user u in time t:

$$p_{v,u}^t = p_{v,u}^0 e^{-(t-t_v)/\tau_{v,u}} \tag{3.11}$$

The initial influence probability $p_{v,u}^0$ may be calculated in a similar manner as described in static models. The parameter $\tau_{v,u}$ is called *mean life time*. It denotes the expected time delay between user v performing an action and this propagates to u. The joint influence probability from all activated neighbors of u becomes:

$$p_u^t(S) = 1 - \prod_{v \in S}(1 - p_{v,u}^t) \tag{3.12}$$

Since this function is dependent from time and the activated neighbors we are not able to compute incrementally its value when a new neighbor is activated since the influence probabilities of any previously activated neighbor will be different. Since each influence probability is maximum when the neighbor first performs the action we expect that joint influence probability function will have as many local maxima as the number of active neighbors. If $max_t\{p_u^t(.)\} \geq \theta_u$ then user u activates.

3.4.3 Discrete Time Models (DT)

The continuous time model as we explained earlier is not incremental. This also means that is computationally expensive and not scalable. In order to overcome this deficiency we will use an approximation that will guaranty us incrementality property.

In *Discrete Time Models* we say that the influence of an active user remains constant at the maximum value for a time window of $\tau_{v,u}$. When this time is over it drops to 0. So we assume that user v is contagious in the interval $[t_v, t_v + \tau_{v,u}]$. Hence when an active neighbor becomes non-contagious we need to update influence probability:

$$p_u(S \setminus w) = \frac{p_u(S) - p_{w,u}}{1 - p_{w,u}} \tag{3.13}$$

For the combination of partial credits with discrete time model we would modify the definition:

$$credit_{v,u}^{\tau_{v,u}}(a) = \frac{1}{\sum_{v \in S} I(0 < t_u(a) - t_v(a) \leq \tau_{v,u})} \tag{3.14}$$

3.5 Learning algorithms

We will present two algorithms for learning influence probabilities. What these algorithms do is to learn the parameters needed to work with all of the models we described in the previous section. The input to these algorithms is a social graph and an action log. With the algorithms presented here we are able to learn all of the parameters necessary for the models we described in the previous section with no more than two scans of the action log.

Algorithm 3 Learning algorithm - Phase 1

```
1:  procedure learning_phase1(Graph, action_log)
2:      for each distinct action a in action_log do
3:          current_table =
4:          for each tuple < u, a, t_u > in chronological order do
5:              increment A_u
6:              parents =
7:              for each user v : (v, a, t_v) ∈ current_table && (v, u) ∈ E^{t_v} do
8:                  if t_u > t_v then
9:                      increment A_{v2u}
10:                     update τ_{v,u}
11:                     insert v in parents
12:                 increment A_{v&u}
13:             for each parent v ∈ parents do
14:                 update credit_{v,u}
15:             add (u, a, t_u) to current_table
```

So we loop through all distinct actions in the action log, having them split in a train and a test set. For each of these actions we loop through each user that performed this action and we eventually add this user in the $current\_table$ the data structure with which we track the users that already performed action a. For each user in the $current\_table$, which means for each user that performed the action before our current user, if there is an edge between the current user and the user we got from the $current\_table$, that was created before current user performed the action a then we assume that the action propagated and we update the related model parameters.

Regarding the update of $\tau_{v,u}$ we just need to keep a sum of time delays $(t_u - t_v)$. When we need to calculate the actual value of $\tau_{v,u}$ we will just divide the sum of delays by A_{v2u}.

The second phase of the learning algorithm loops through the $action\_log$ for a second time. In order to learn the parameters for the discrete time model as well

Algorithm 4 Learning algorithm - Phase 2

```
1:  procedure learning_phase2(Graph, action_log)
2:      for each distinct action a in action_log do
3:          current_table =
4:          for each tuple < u, a, t_u > in chronological order do
5:              parents =
6:              for each user v : (v, a, t_v) ∈ current_table && (v, u) ∈ E^{t_v} do
7:                  if 0 < t_u − t_v < τ_{v,u} then
8:                      increment A_{v2u}
9:                      insert v in parents
10:                 for each parent v ∈ parents do
11:                     update credit_{v,u}^{τ_{v,u}}
12:                 if parents! = then
13:                     update infl(u)
14:             add (u, a, t_u) to current_table
```

as to calculate user influenceability $infl(u)$ we need first to obtain the value for τ which is learned by the first phase of the algorithm. The essential difference in the second run of the algorithm is the fact that the requirement in order to consider that an action propagated from user v to the user u is not solely based on the fact that $t_u > t_v$ but we also demand that the delay is smaller than the average delay for the connection (v, u).

In the previous algorithms E^{t_v} denotes the edges (out-edges if we have a directed graph) that user v had at time t_v. The algorithms presented in this section can be easily modified to run on a directed graph.

3.6 Evaluation algorithms

Following the logic of the learning process in the evaluation algorithm we will use a data structure ($results\_table$) with entries of the form $< u, p_u, perform_u >$. The flag $perform_u$ denotes whether the user u has performed the action. The value 0 means the user never performed the action but at least one of its neighbor did, 1 means that the user performed the action and at least one of its neighbors did it before, and 2 means that the user was the first one to perform the action in its neighborhood. The $results\_table$ contains all the activated users for a specific action up to a specific time.

As we loop through the action log and we read a new tuple $< v, a, t_v >$ we add user v and all of its neighbors to $results\_table$ with the appropriate influence

probability (the influence probability as calculated by the learning process through the edge $< v, neighbor >$) and $perform\_flag = 0$ for v and all of its neighbors. If user v is already in the $results\_table$ this means that another neighbor of v has already performed the same action, so we should update $perform_v = 1$. If node v is not present in the $results\_table$ this means that he performed the action without being influenced by any of its neighbors[1] so we consider it the initiator of the action for its neighborhood and we set $perform_v = 2$. Also some of v's neighbors might already be present in the $result\_table$ so we incrementally update their probability to perform the action. When all action tuples are read the $results\_table$ should include all users that performed the action and all the users that are neighbors to at least one user that performed the action.

This algorithm is used to produce the confusion matrix for this model. First of all we ignore the cases where non of the user's friends is active. We consider as TP the cases where a user performs an action and at least one of its neighbors performs the action before it and the model predicts that the user performs the action and so on for TN, FP, FN.

The next algorithm evaluates the models in which the joint influence probability is time dependent. Whenever a new neighbor performs the action, the joint influence probability towards a user shows a sharp increase and then starts decreasing. So if a user has d (in-)neighbors who performed an action, then the influence probability would have d local maxima. If any of these maxima is above the threshold for the user θ_u we conclude that the user will activate. So in order to calculate the probability for a "parent" user $p\_current$ we need to to re-evaluate the probabilities for all the previous "parent" users with a timelapse from the time each of these parents activated until the time $p\_current$ activated.

[1] This is might introduce inconsistencies when we run the algorithm on github data that we have a partial knowledge of the graph.

Algorithm 5 Evaluate-Basic

```
1: procedure evaluate(Graph, action_log)
2:     for each distinct action a in action_log do
3:         results_table =
4:         for each tuple < v, a, t_v > in chronological order do
5:             if v ∈ results_table then
6:                 perform_v = 1
7:             else
8:                 add v to results_table
9:                 p_v = 0 and perform_v = 2
10:            for each user u : (v, u) ∈ E^{t_v} do
11:                if u ∈ results_table then
12:                    update p_u incrementally
13:                else
14:                    add u to results_table
15:                    p_u probability estimated based on the model
16:                    perform_u = 0
17:        for each entry < u, p_u, perform_u > in results_table do
18:            if perform_u == 1 && p_u ≥ θ_u then
19:                TP+ = 1
20:            else if perform_u == 1 && p_u < θ_u then
21:                FN+ = 1
22:            else if perform_u == 0 && p_u ≥ θ_u then
23:                FP+ = 1
24:            elseperform_u == 0 && p_u < θ_u
25:                TN+ = 1
```

Algorithm 6 Evaluate-Complex

```
1: procedure evaluate(Graph, action_log)
2:     for each distinct action a in action_log do
3:         results_table =
4:         for each tuple < u, a, t_u > in chronological order do
5:             if v ∈ results_table then
6:                 perform_v = 1
7:             else
8:                 add v to results_table
9:                 p_v = 0 and perform_v = 2
10:            for each user u : (v, u) ∈ E^{t_u} do
11:                if u ∉ results_table then
12:                    add v in results_table with
13:                    p_v = 0 and perform_v = 0
14:        sorted_parents =
15:        for each entry < u, p_u, perform_u > in results_table do
16:            for each user v : perform_v! = 0, (v, u) ∈ E^{t_v} do
17:                add v to sorted_parents
18:            for each neighbor v_i ∈ sorted_parents do
19:                compute p_u(t_{v_i}) at time t_{v_i} for u_1 to u_i
20:                if p_u(t_{u_i}) > p_u then
21:                    update p_u
22:        for each entry < u, p_u, perform_u > in results_table do
23:            if perform_u == 1 && p_u ≥ θ_u then
24:                TP+ = 1
25:            else if perform_u == 1 && p_u < θ_u then
26:                FN+ = 1
27:            else if perform_u == 0 && p_u ≥ θ_u then
28:                FP+ = 1
29:            elseperform_u == 0 && p_u < θ_u
30:                TN+ = 1
```

Κεφάλαιο 4

Experimental evalutation of learning algorithms

4.1 Introduction

In the following chapter we will present our findings while using real data which we acquired from various sources. The datasets we are used were both previously used in research and were already in an easily digestible format and in somehow raw data we got from github archive and flickr open api.

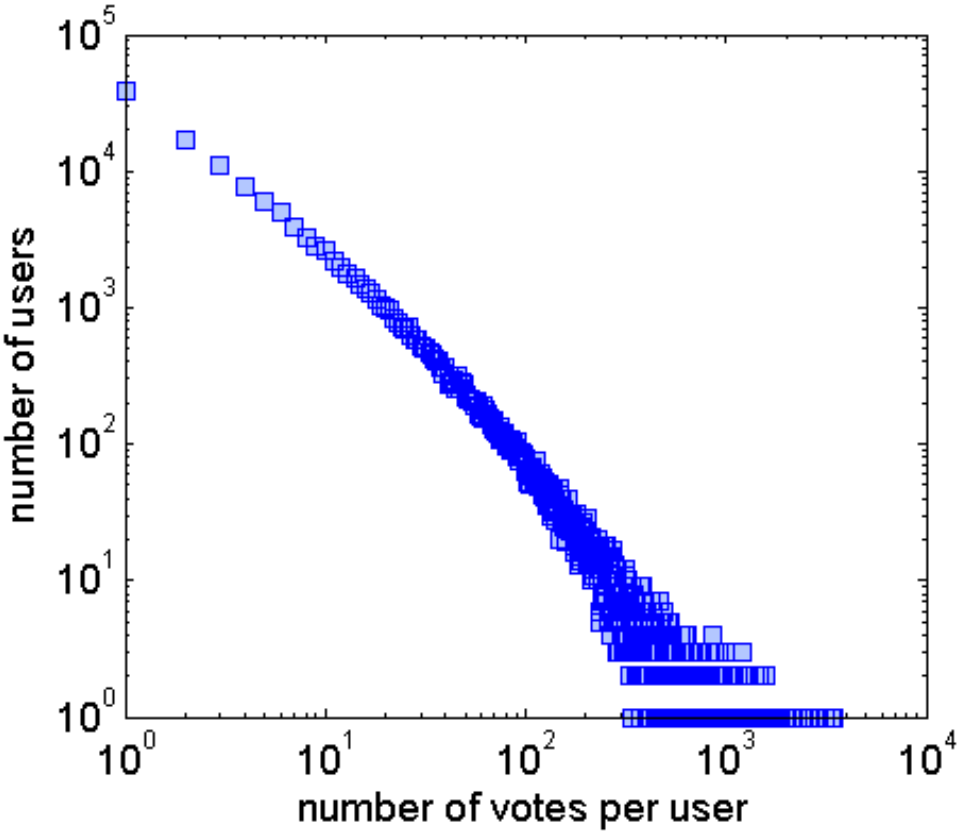
One subtle thing that we need to keep in mind in our discussion regarding directed social graphs is the direction of the edges. In the datasets we acquired either from the sources we reference or by mining online sources on our own the common notion was the ”following” edge. This would mean if a user u is following user v we would represent this as an egde from u to v. Regarding information flow though we would need to have a graph with the inverse relations among users denoting not the ”following” notion of an edge but the information flow, which is in this kind of directed social networks flows from the followed to the following user.

4.2 Digg

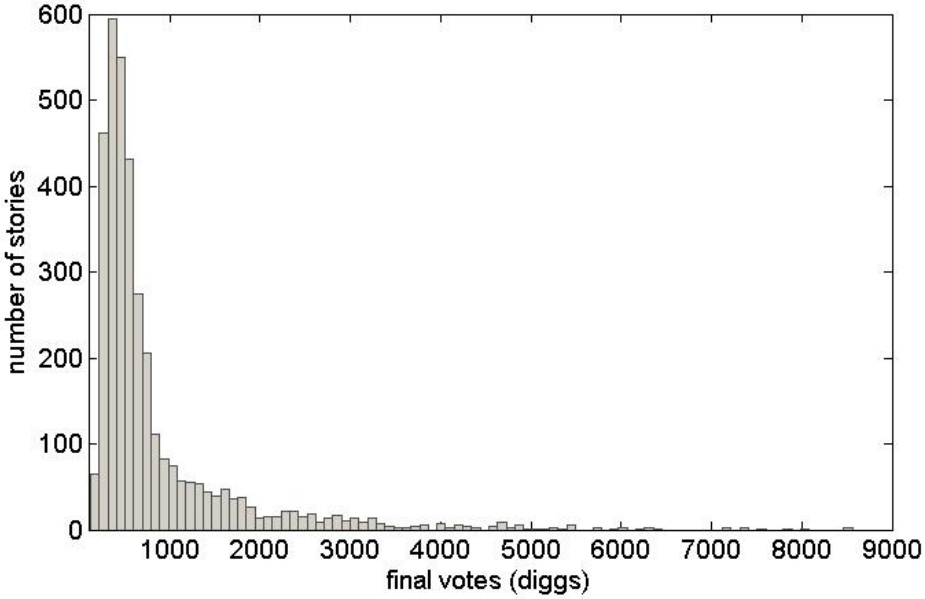
Digg is a news aggregator. In this dataset we have data regarding stories and their votes that made it to front page of Digg over a period of a month in 2009. In this dataset we have records for users relations in order to be able to reconstruct the social graph. The link from user u to user v means that the start user is following the user at the end of the edge. This means that the information flows in the opposite direction from user v to user u.

This dataset is composed of two tables. One votes table which contains approximately

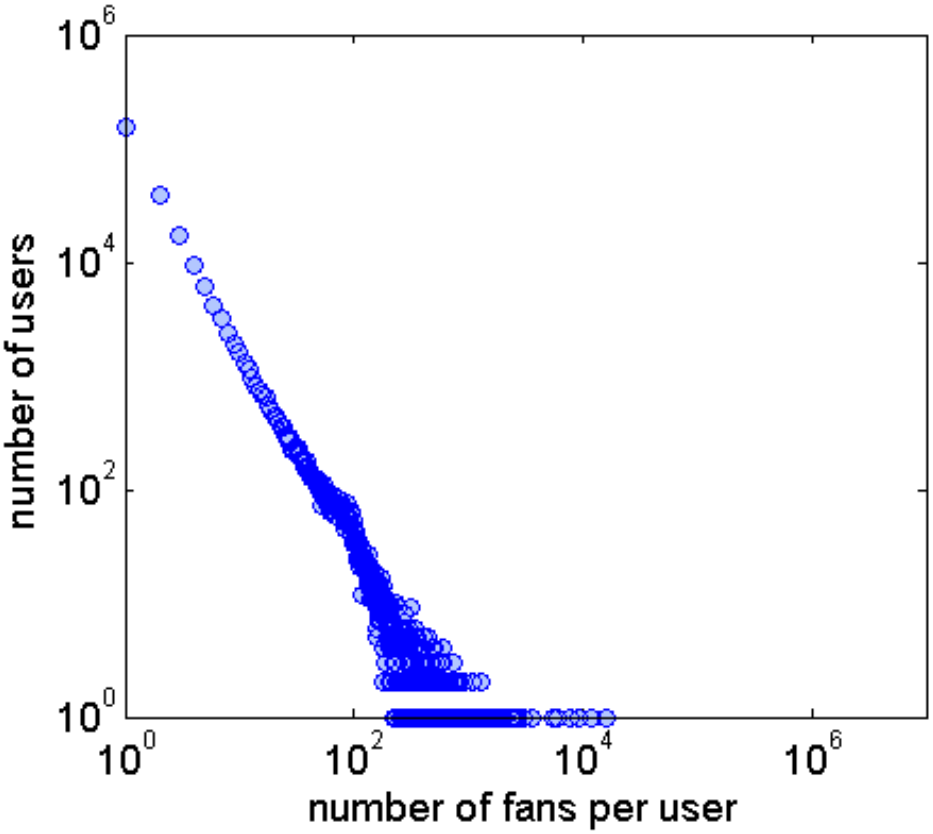
3 milion votes on 3553 stories performed by 139,409 distinct users. The first vote for each story is not actually a vote but the creation act from the submitter of the story. In the following figures we can have an overview of the characteristics of the digg dataset [10].



Σχήμα 4.1: Votes per user



Σχήμα 4.2: Votes per action

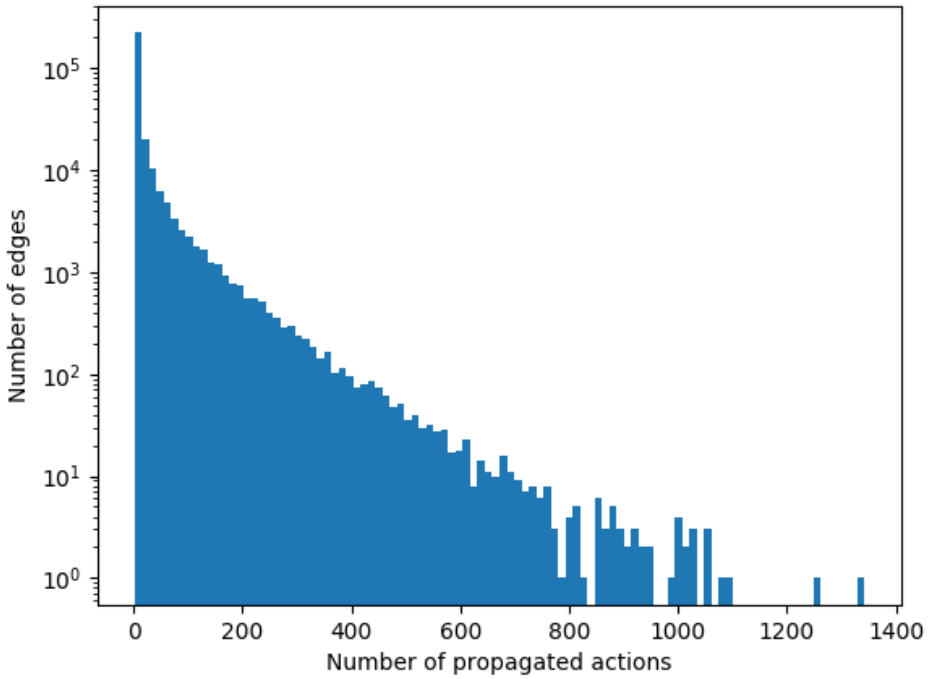


Σχήμα 4.3: Social link distribution

4.2.1 Statistics regarding learning probability

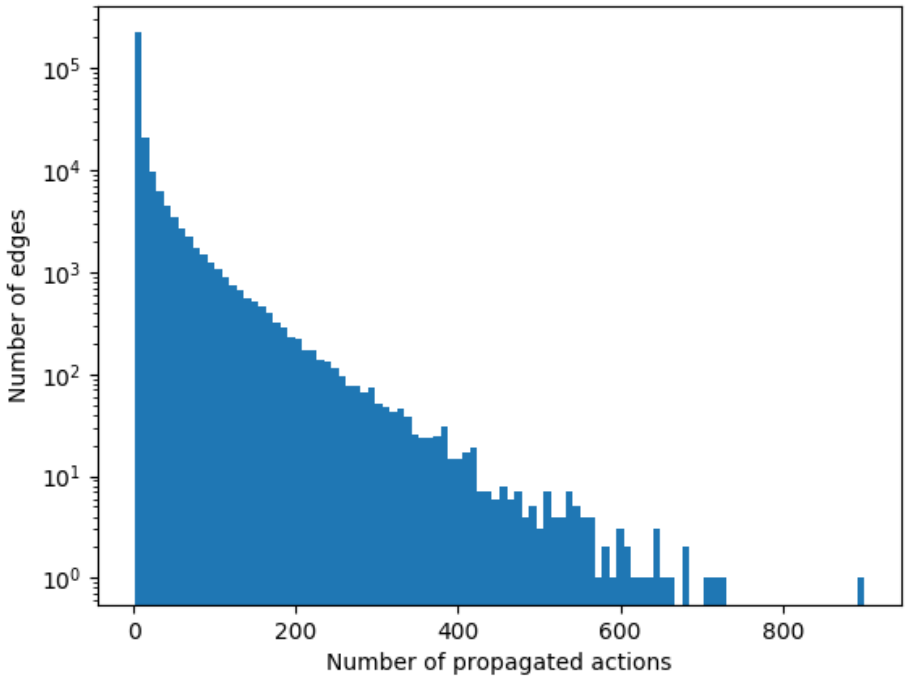
In this section we will present some statistics that are created through the learning influence probability algorithm. In the first phase of the algorithm [3] we learn some characteristics of the graph based on which we can calculate the probability for the various models.

One of the characteristics we learn in phase 1 is the number of propagated actions for each edge of the graph. Lets assume that user u follows user v. If the connection between those two users was present before any of the users performed the action, and if user v performed the action before user u (we remind the reader that the flow of information is from the user being followed towards the follower) then we assume that the action propagated from user v to user u. Of course with this definition it is possible that an action may propagate through various edges towards one user.



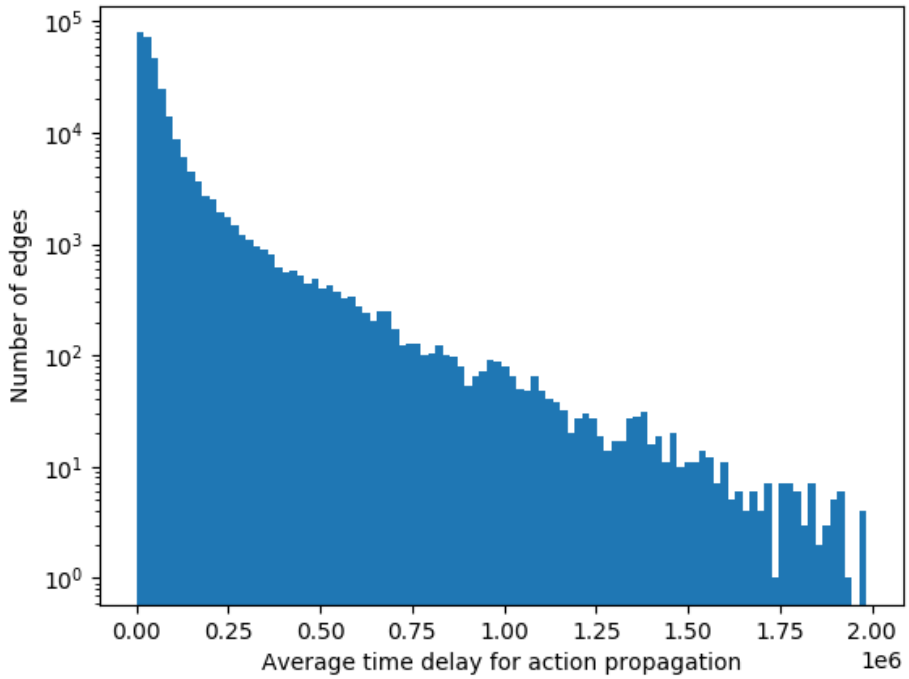
Σχήμα 4.4: Digg's propagated actions in phase 1 of the learning algorithm.

In phase 1 we keep track of the time it takes for an action to propagated from one user to the other. In this manner we are able to compute the average time it takes for an action to propagate through each edge. In phase two we consider as propagated actions only those whose propagation took less than this average time delay. So the following diagram displays a similar but reduced distribution.



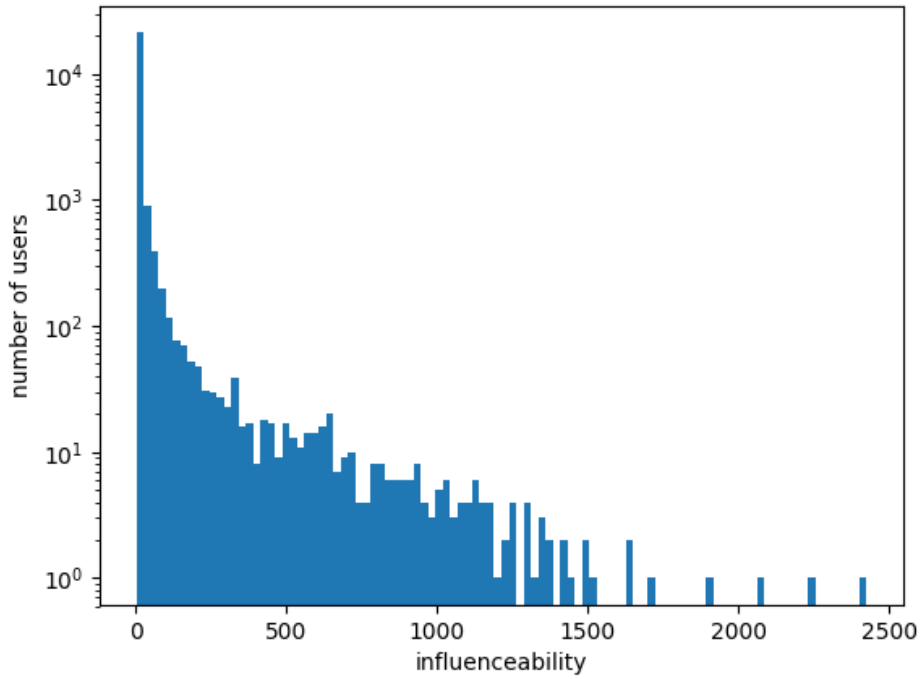
Σχήμα 4.5: Digg's propagated actions phase 2, filtered with average time delay.

The average time delay distribution is displayed in the following diagram.



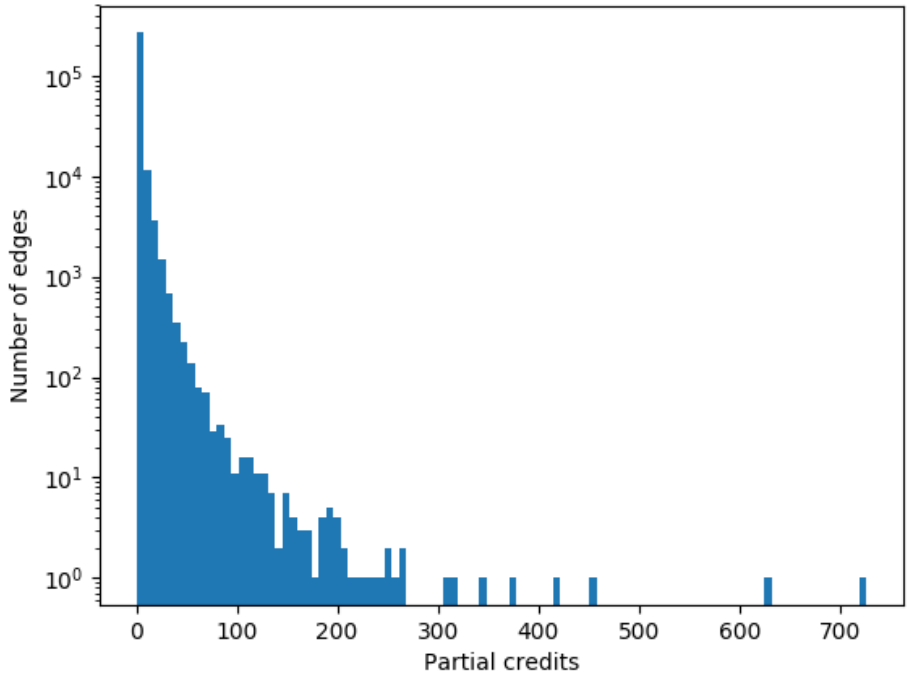
Σχήμα 4.6: Average time delay

The influenceability dignifies the amount of actions for which we have evidence that the user was influence by its neighbors. We would expect that filtering out the users with small influenceability we would get better predictions since our calculations would gain a better statistical significance.



Σχήμα 4.7: Influenceability

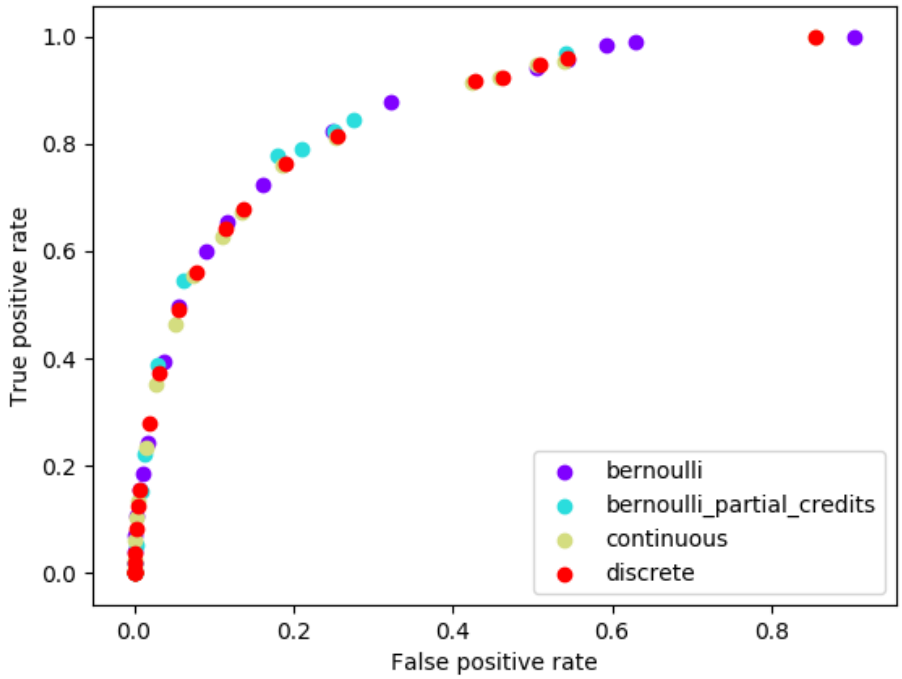
The last component learned by phase 1 of the learning algorithm is the partial credits for each edge. The partial credit values are generated by attributing to all active incoming edges for an activated user the same weight.



Σχήμα 4.8: Partial Credits

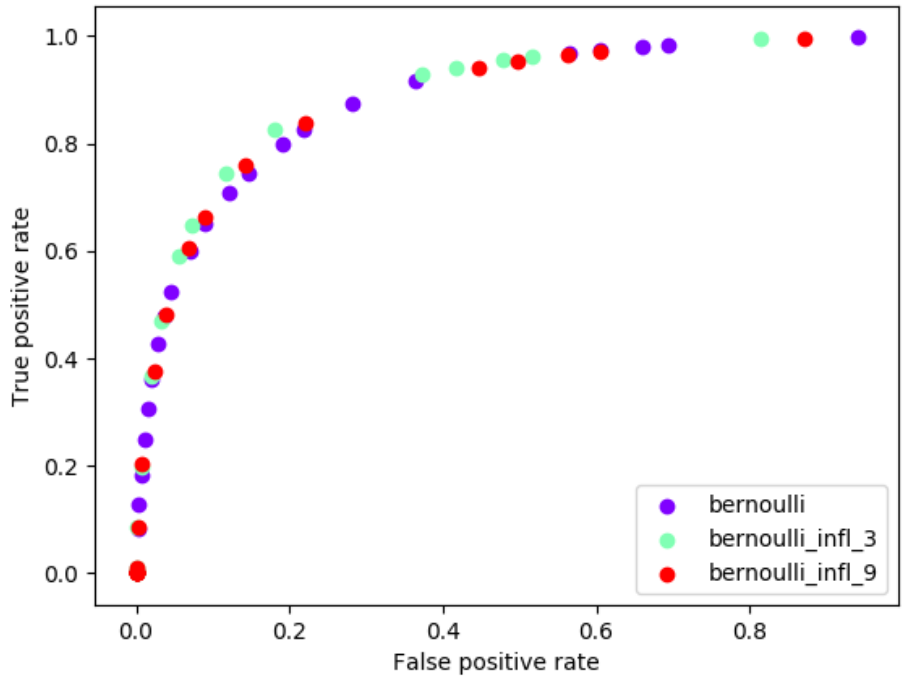
Based on the data learned on phase 1 of the algorithm for the digg dataset we run the evaluator algorithm that produces the confusion matrix. With the confusion matrix we can produce the roc curve for different thresholds for the probability. The following diagram is the basic bernoulli probability [3.6], evaluated on approximately $5 * 10^5$ actions.

This plot shows the predictive strength of the proposed model. Our plot is quite far from the diagonal. We don't seem to get the same results as displayed in the original paper. The time consious models do not display any significant improvement when compared to the static models.



Σχήμα 4.9: Roc curves for Digg dataset. These are produced running the evaluation algorithm over 50000 actions.

The following plot shows the roc curves for the static bernoulli model for different values of influenceability. We can see that filtering out the less influenceable nodes gives us a slightly better model. The evaluation was done over $1.8 * 10^5$ actions.

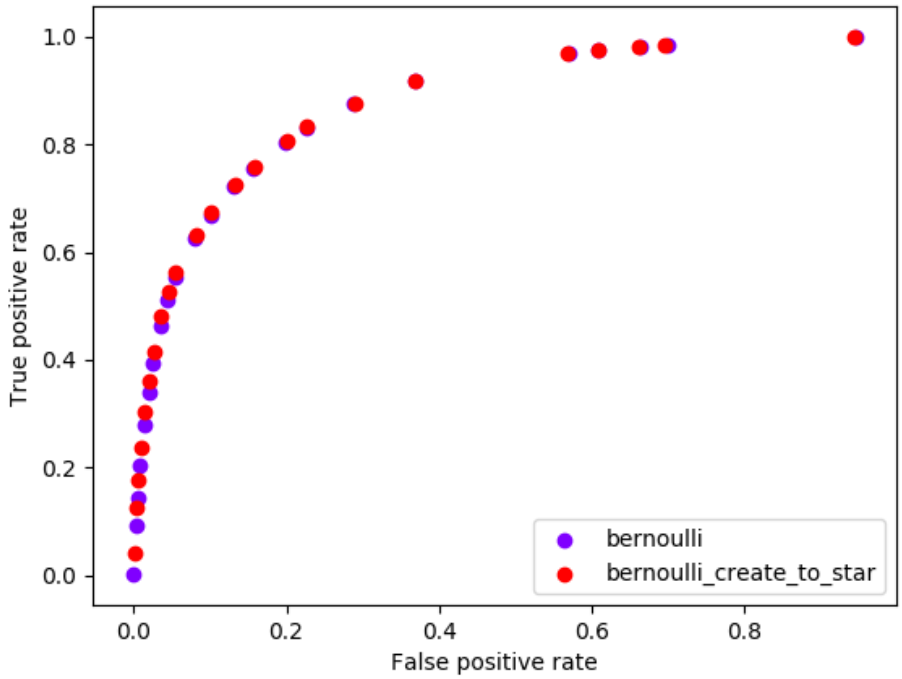


Σχήμα 4.10: Roc curves produced by the evaluation algorithm for different values of influenceability.

4.2.2 Utilize the different type of actions

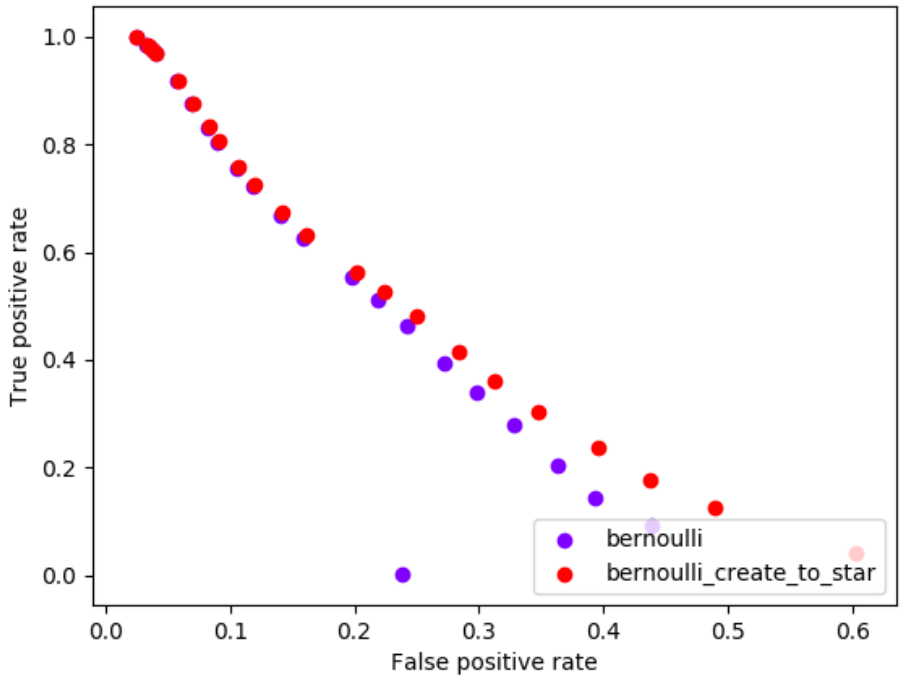
Thinking of how the influence is spread within a social network there is a possibility that some types of action influence differently each user. The action of someone creating/publishing an article in a social network will have a different effect on his/hers social connections from the action of upvoting an article from the same user. By modifying the algorithm of phase 1 slightly we can attribute different probabilities in the ”create” to ”like” action and to ”like” to ”like” action.

In the following plot we can see that there is no obvious improvement when comparing the static Bernoulli model and a static Bernoulli model which attributes different probabilities in different types of actions.



Σχήμα 4.11: Roc curves comparing Bernoulli model and Bernoulli for different types of action.

A note that we need to make here is the fact that roc curves might not be the ideal when evaluating unbalanced datasets. In our case the number of negative examples are far greater than the positive. The reason for this is that for each user that performs an action we scan all of his/her neighbors to check whether they performed the action themselves. In the following plot we are displaying the precision-recall curve for the same evaluation. We can see that there is a clear improvement when distinguishing the influence probabilities for the different types of actions. Another conclusion we may draw from this plot is that there is quite big room for improvement something that is not quite obvious from the roc curves.



Σχήμα 4.12: Precision recall curves comparing Bernoulli model and Bernoulli for different types of action.

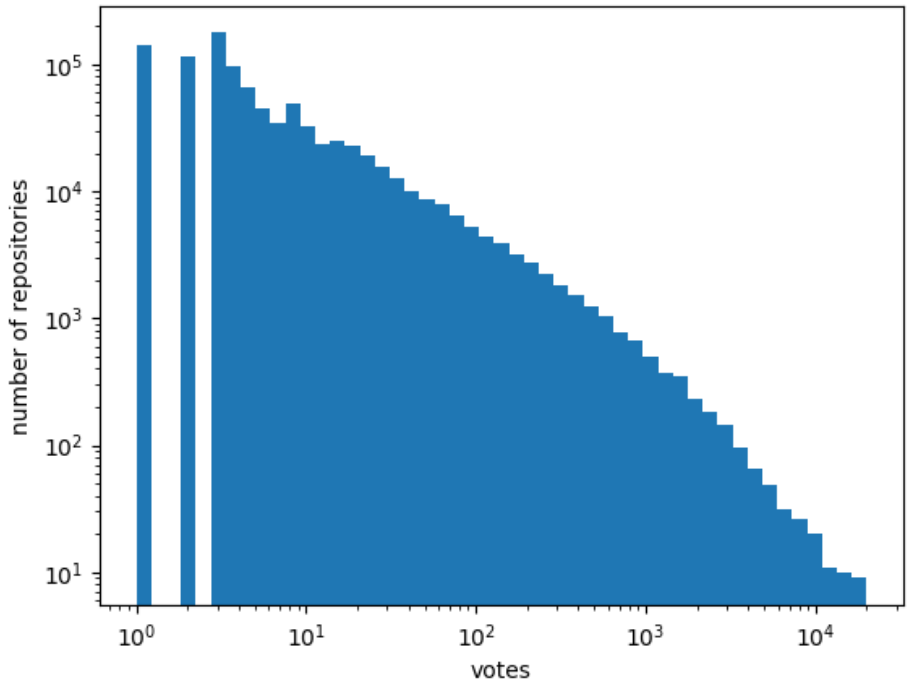
4.3 Github archive

Through the github archive site we can freely download the events that happened every hour from 12/02/2011. Among these events exists a ”FollowEvent” which gives us information regarding the connections in the social graph. There are many reasons that limit our knowledge of the graph though. The first and most obvious is that we do not have the events from the initial release of github. The second is that we don’t have information regarding the breaking of a connection between users. And the third is that in the dataset provided by the github archive we can find following events until the 11/12/2013.

Except for the social-graph the second dataset we must somehow acquire, in order to be able to learn the influence probabilities, is the action log. The actions we used in order to construct the action log were the ”CreateEvent” and ”WatchEvent” which represent the creation of a new repository and the action of giving a star to a repository. In github though there are many create events that are not followed by any watch/star event. So we excluded these create events from our action log.

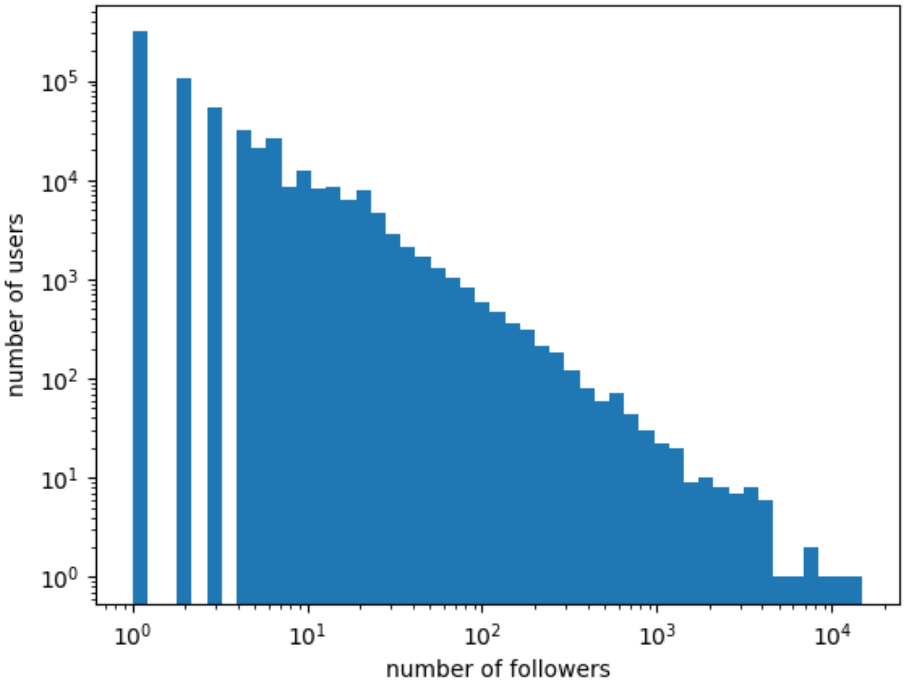
We stored these events in two mysql tables, one regarding the connections between users along with the date on which the events happened, and a second table to hold the create and watch actions. Regarding the social graph we ended up with $3 * 10^6$ connections and $8.5 * 10^5$ nodes. In the action log we ended up with $12 * 10^6$ *watch* events and $6.8 * 10^5$ related *create* events.

In the following figure we can see the number of votes distribution over the number of repositories.



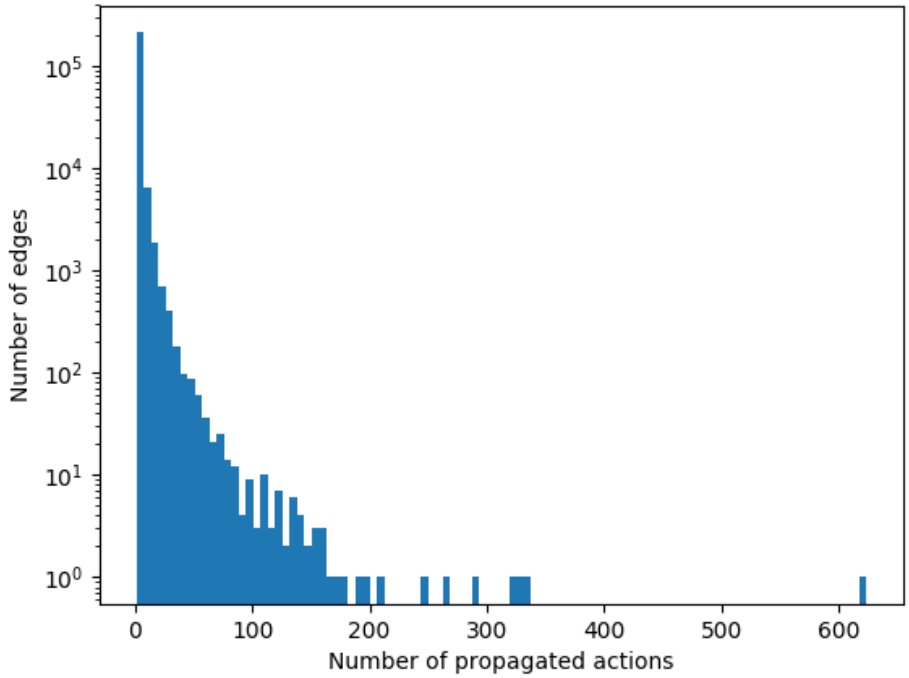
Σχήμα 4.13: Votes distribution from github archive dataset

As we can see in the following histogram the github social graph exhibits a very clear power law degree distribution which is found in social networks and other ”scale-free” [11] networks, as these are commonly referred to through the literature.

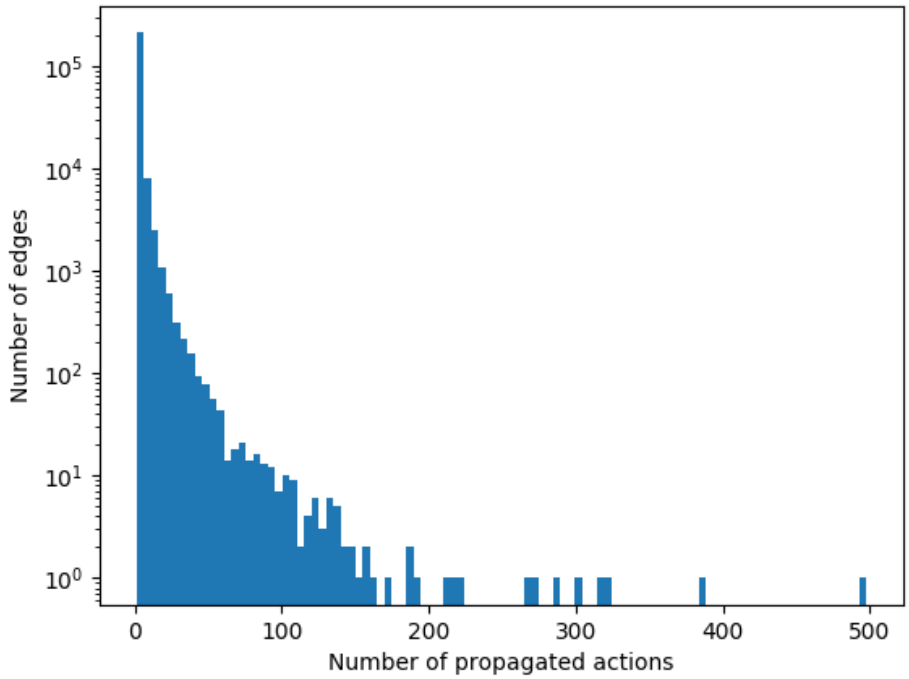


Σχήμα 4.14: Social link distribution in github network

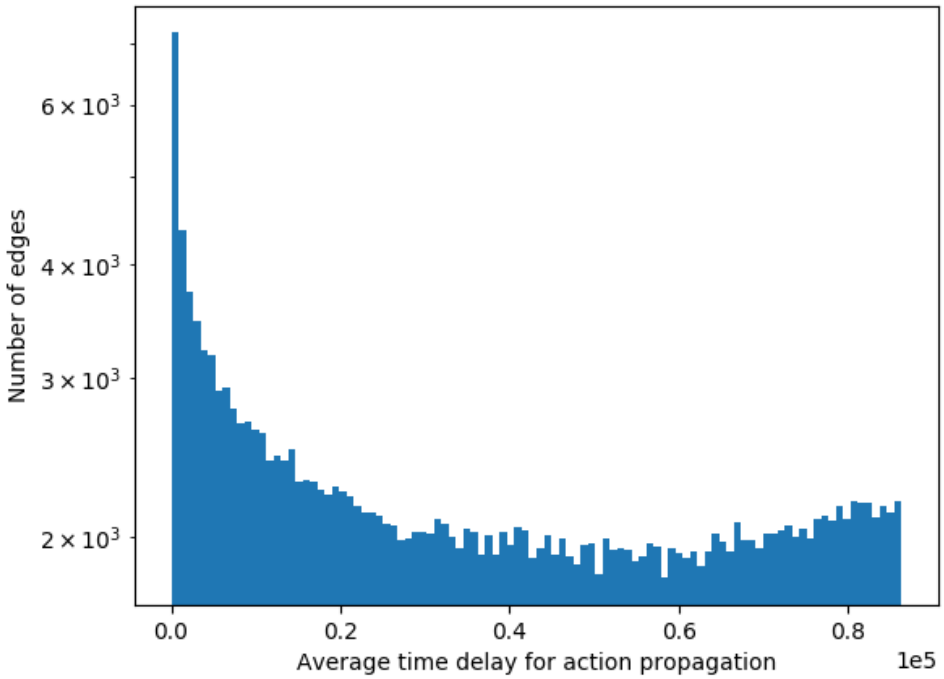
In the following histogram picturing the propagated actions found in the first phase of the learning algorithm we can see that we have a somewhat worse action log, when compared to the digg network. Even though we have a quite larger action log it is a lot sparser when we look at the amount of propagated actions. Of course we expect this to have an effect on the predictive power of our algorithm. The reason for this could be either something we overlooked in the preprocessing of our dataset or simply the fact that the technical and scientific nature of this networks users makes them quite less prone to peer influence!



Σχήμα 4.15: Propagated actions found in phase 1 of the learning algorithm in github

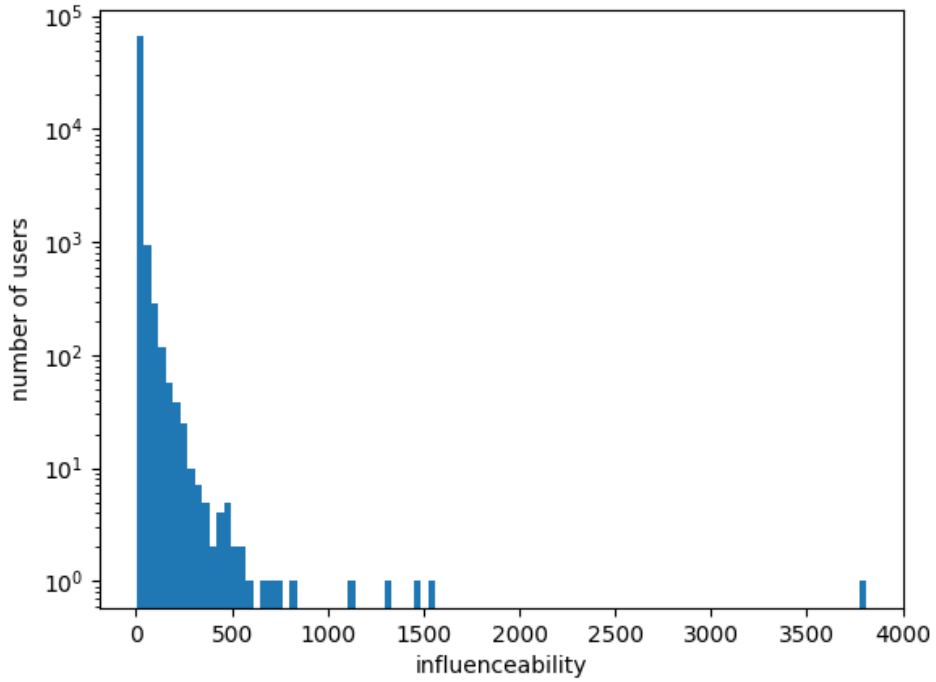


Σχήμα 4.16: Propagated actions found in phase 2 of the learning algorithm, filtered with average time delay

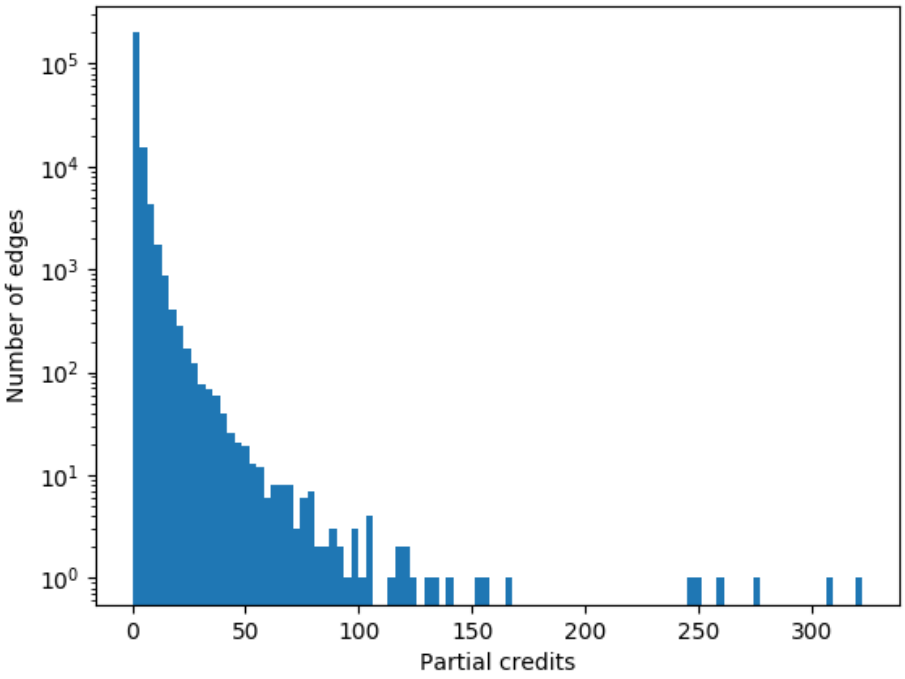


Σχήμα 4.17: Average time delay in github.

As would expect from the propagated actions diagram the influenceability in github network is somewhat diminished. We remind the reader that influenceability represents the amount of actions for which we have an indication that a user was influenced by one or more of its neighbors.

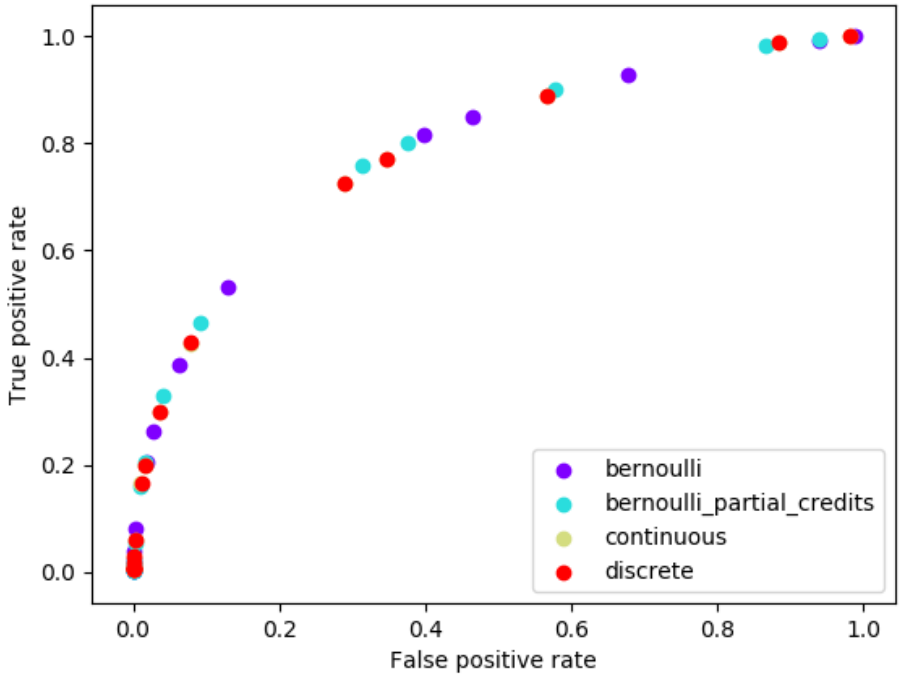


Σχήμα 4.18: Influenceability in github.



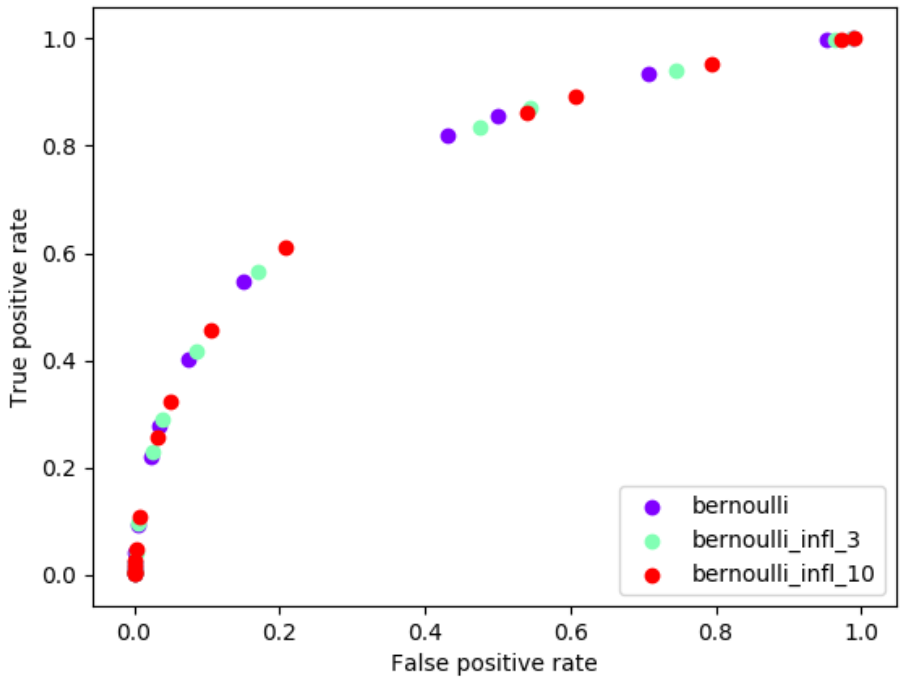
Σχήμα 4.19: Partial credits in github

The result of the evaluation algoritms is depicting the fact that influence is not so strong in this network. The various models here also do not exhibit any different behavior. The predictability of all the models is almost equal in contradiction to the results of the original paper.



Σχήμα 4.20: Roc curves for github archive

In the following plots we can see the roc curves produced by the evaluation algorithm for different values of influenceability. There is not obvious improvement in the github social network.



Σχήμα 4.21: Roc curves for github archive for different influenceceability values

4.4 Flickr Api

Another source of data that I explored in the context of my thesis was flickr api, a description of which can be found here `https://www.flickr.com/services/api/`. The size of the flickr social graph is enormous, having more users than the population of earth. The amount of users I found in my first tries where more than 20 billion. To acquire this number I started from a randomly selected user with a large amount of public contacts, and crawled though the graph following the contacts relationship.

The problem with this approach was that for each of the contacts a separate api request was required. Taking into consideration that the flickr api has limitations for the amount of requests it allows from a specific user, the approach of having a complete knowledge of the public contacts graph was not possible in a logical amount of time. The other limitation was that the contacts endpoint did not provide us with the time that the connection among the users was created, which is a requirement for the learning process in order to be able to determine whether an action propagated over a social edge. Nonetheless we could restrict our calculations on actions performed after reconstructing the social graph.

Of course not all of these users were active flickr users. Meaning they don't actively participate in creating social connections, creating new content or performing the action of upvoting a new photo. So I came up with another approach that I started but not completed since this would also require a fair amount of time to get a good portion of the active flickr social graph and performed actions.

The logic I used was getting newly uploaded photos through $getRecent$ endpoint[1] as well as their owner. I would then filter out the photos by using only the ones that get an amount of likes above a certain threshold for a given timeframe. For each of those photos we would proceed and perform some requests to collect both a part of the active social network and the action log for the flickr platform. The next step would be to get the public list of the owners contacts through $getPublicList$ endpoint[2]. The $getPublicList$ endpoint however returns a list of the users that the user for whom we performed the request follows. Currently there is not endpoint to easily get the followers of a user in flickr api. We would then proceed on monitoring the favorites performed for the specific photo. For each of the favorites we would add it to the action log. Each favorite contains the id of the user that performed the action as well as the time that the action happened. For each of the users that performed the action we would add the users she/he follows, assuming that the connection between them was created at least at the time the photo was created.

[1] `https://www.flickr.com/services/api/flickr.photos.getRecent.html`

[2] `https://www.flickr.com/services/api/flickr.contacts.getPublicList.html`

Note that $crawlPhoto()$ method can be spawned into an independent non blocking process. The step of adding the contacts of the owner of the photo also may be avoided.

Algorithm 7 Flickr crawler

```
1: procedure crawl(threshold)
2:     while crawlingPeriod > elapsedTime do
3:         recentPhotos = flickr.photos.getRecent()
4:         for photo in recentPhotos do
5:             favorites = flickr.photos.getFavorites(photo)
6:             if favorites['total'] ≥ threshold then
7:                 crawlPhoto(photo, timeframe, photo['createdAt'])
```

Algorithm 8 Crawl photo

```
1: procedure crawlPhoto(photo, timeframe, photoCreatedAt)
2:     contacts = flickr.contacts.getPublicList(photo['owner'])
3:     for contact in contacts do
4:         add edge (contact, photo['owner']) to graph
5:         edge['time'] = photoCreatedAt
6:     sleep(timeframe)
7:     favorites = flickr.photos.getFavorites(photo)
8:     for action in favorites do
9:         add action to actionLog
10:        contacts = flickr.contacts.getPublicList(action['username'])
11:        for contact in contacts do
12:            add edge (contact, action['username']) to graph
13:            edge['time'] = photoCreatedAt
```

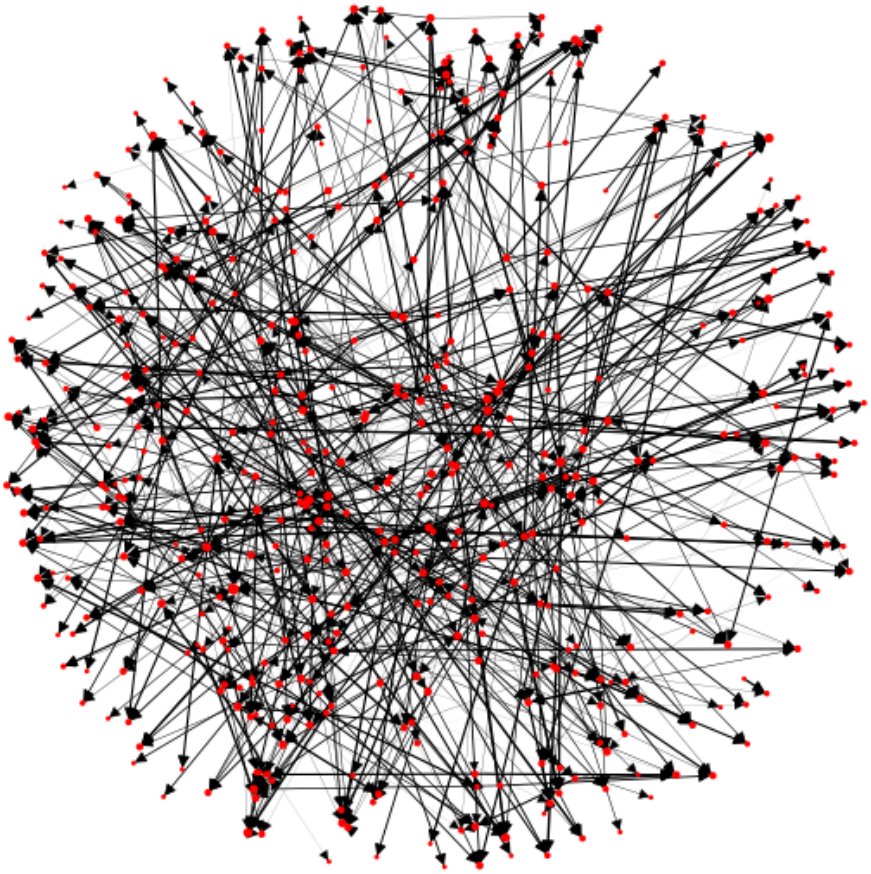
Κεφάλαιο 5

Experimental evaluation of influence maximization algorithms

In order to run the influence maximization algorithm we need to somehow acquire a social graph and influence probabilities for each of the edges. This can be done using either a real social graph with learned probabilities, a real social graph with artificial probabilities, or a constructed random graph. In the following sections we will present some of these approaches and our results.

5.1 Influence maximization on random graphs

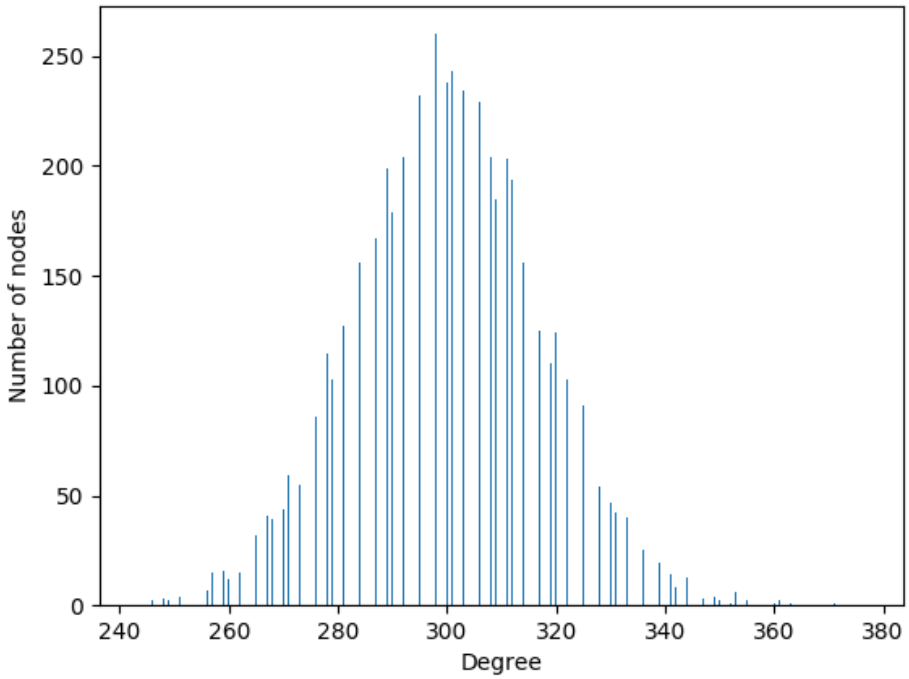
The most obvious approach when constructing a random graph would be to start totally at random. Add a number of nodes into the graph and for each of the pairs connect or not depending on some predefined probability. Depending on the threshold we could result to a completely unconnected graph, a graph with multiple components or a fully connected graph. The degree distribution in this case would be a normal distribution around the mean value being $connection\ probability * number\ of\ nodes$.



Σχήμα 5.1: Directed random graph, with 500 nodes and 10^3 edges.

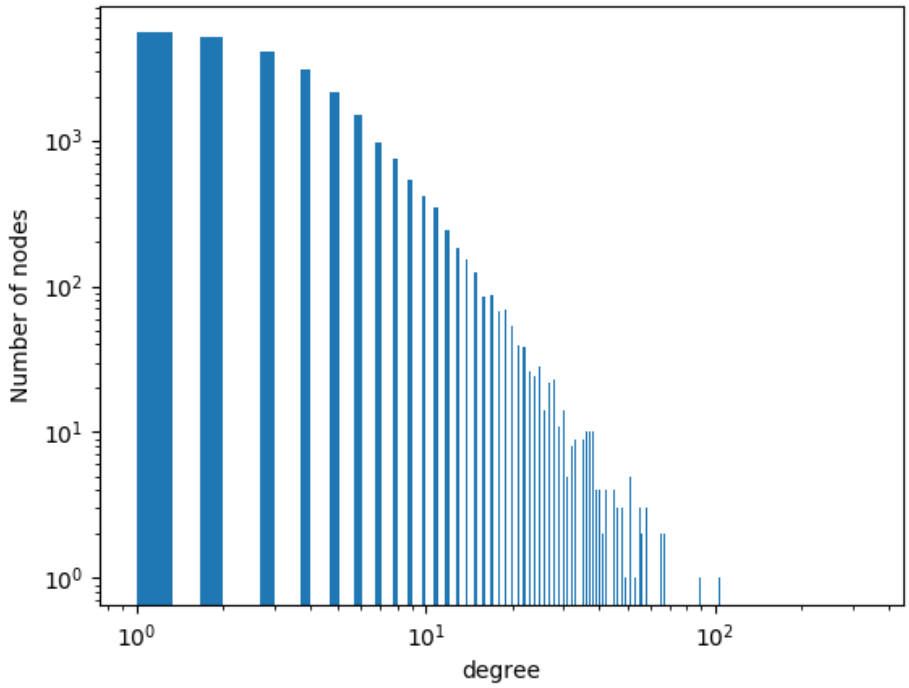
The second approach, which displays a degree distribution much closer to a real network dynamics. This is generated by a preferential attachment logic. Each new node added to the network has a greater probability to attach to ”popular” nodes. This probability is proportional to the degree of the node, k_j in the following equation represents the degree of node j.

$$p_i = \frac{k_i}{\sum_j k_j} \tag{5.1}$$



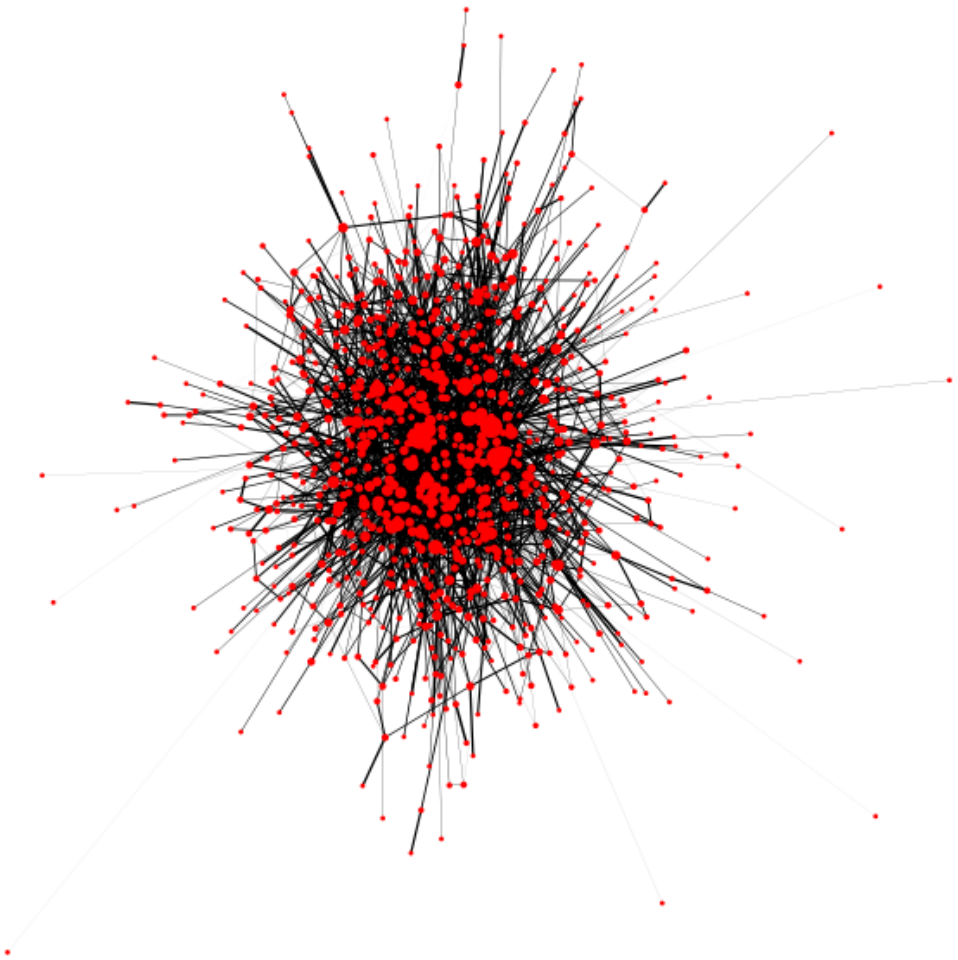
Σχήμα 5.2: Random graph degree distribution. In this example the link probability was 0.03 and the number of nodes is 10^4

In the following image we can see a $Barabasi-Albert$ graph with 1000 nodes and 2000 edges. The different structure of the two graphs is visible. The size of each node represents the degree and the thickness of each edge represent the assigned probability.



Σχήμα 5.3: Barabasi Albert graph. This graph consists of $5 * 10^4$ nodes.

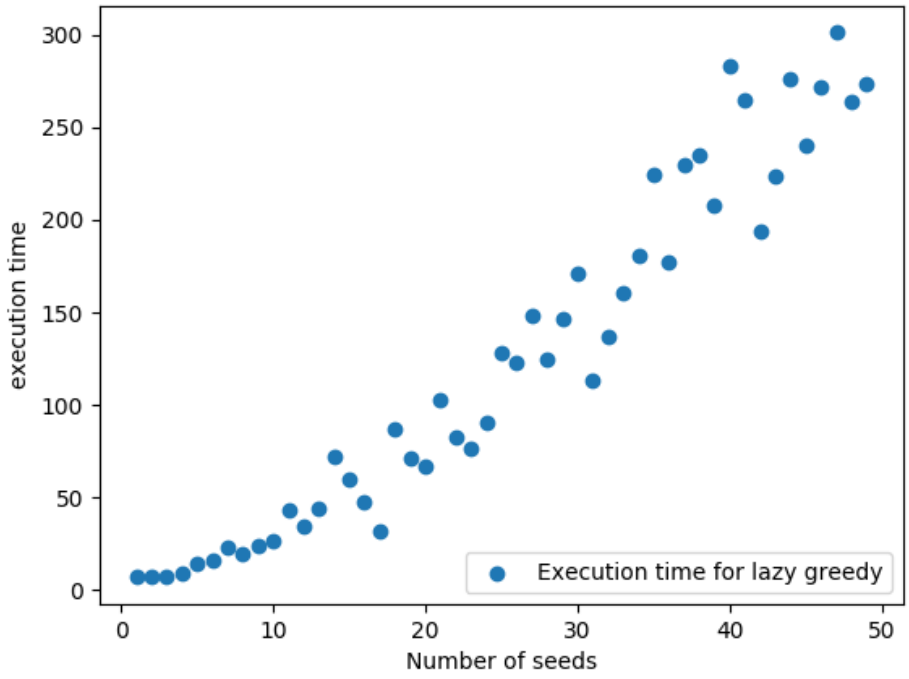
In the following plots we are demonstrating the execution time for different number of seed nodes for the lazy greedy algorithm. The graph we are running the algorithm upon is created by the barabasi albert algorithm with 500 nodes and randomly assigned probabilities.



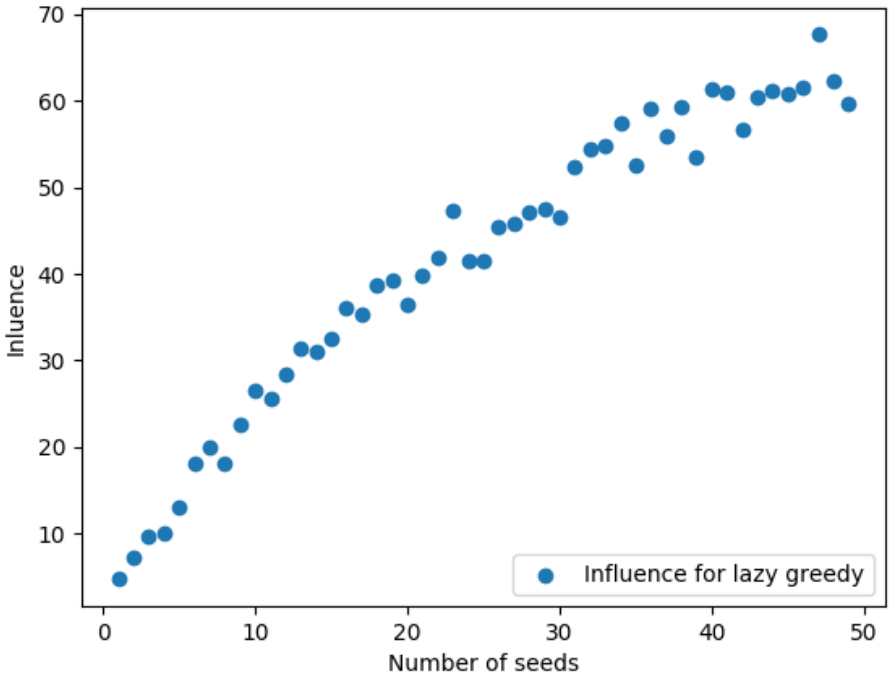
Σχήμα 5.4: Barabasi Albert graph.

5.2 Digg and Gihub social graphs

Based on the learning process we described on a chapter 4 we were able to calculate the influence probabilities for big portion of the social graph's edges. In digg's social graph which consists of $1.7 * 10^6$ edges we got influence probabilities greater that zero for $2.9 * 10^5$ edges. The github graph we have constructed based on the follow events of the github archive dataset consists of $3.1 * 10^6$ edges with $2.2 * 10^5$ of them with non zero probability. In order to be able to run the influence maximization algorithm in a reasonable amount of time we would need to further reduce the size of the graph as we have seen through the various runs on the artificial graphs in the previous section.



Σχήμα 5.5: Execution time for lazy greedy algorithm on barabasi-albert graph.



Σχήμα 5.6: Influence for the seeds that are returned from the Lazy greedy algorithm on the same graph, calculated with simulations over linear threshold model.

Κεφάλαιο 6

Discussion

In this thesis we explored two main problems regarding influence in social networks. The one is the influence maximization problem and the main approximation algorithms that can be found throughout the literature. In the same context we also present two different models for the information/influence diffusion in the social networks Independent Cascade and Linear Threshold models. Under both of these models the influence function remains submodular which ensures the approximation efficiency of the greedy algorithm. We provide a robust python implementation for aforementioned models, the greedy algorithm, as well as an optimized version (lazy-greedy) which exploits the submodular property of the influence function to reduce computation steps.

The second problem regarding influence, is how we can learn these influence probabilities that we take for granted in the first problem, in a real social network. We proceed in the implementation of the algorithms proposed in the literature which we use to calculate the influence probabilities and evaluate the accuracy of the various proposed models. We have implemented minor modifications to the algorithms to adapt them to a directed network instead of the undirected initial implementation. We also create a ruby gem to parse github archive events into mysql tables, digestible from the learning and evaluating algorithms, producing a new social network dataset.

Running the learning algorithms over the action logs of both digg and github social networks we managed to confirm the findings in literature regarding the predictability of the proposed models. We did not detect any improvement though using time conscious models, which are a lot more expensive computationally compared to the static models. We found a minor improvement in the digg social network when restricting our evaluation only on users with influenceability above a certain threshold, a finding that is not reproducible on github dataset.

We proposed a novice approach in calculating influence probabilities by distinguishing the effect of different types of actions. The new model did not exhibit any improvement

in roc curves but there was a significant improvement in precision recall curves which might be a better evaluation method for this specific problem due to the fact that our dataset is unbalanced.

Regarding the influence maximization we managed to run the algorithms on randomly generated graphs with randomly assigned probabilities. The size our real social graphs is quite big to run this algorithms over them. We would probably need a performance improvement on our models for influence diffusion like *maximum influence in-arborescence*, since with our existing models we need to go through the whole network to get an estimation of the influence.

Βιβλιογραφία

[1] W. Chen, C. Wang, and Y. Wang. *Scalable influence maximization for prevalent viral marketing in large scale socialnetworks.* InProceedings of the 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2010.

[2] Amit Goyal, Francesco Bonchi, Laks V. S. Lakshmanan, *Learning influence probabilities in social networks*.

[3] G.L. Nemhauser, L.A. Wolsey, M.L. Fisher, *An analysis of approximations for maximizing submodular set functions*.

[4] J. Goldenberg, B. Libai, E. Muller. *Talk of the Network: A Complex Systems Look at the Underlying Process of Word-of-Mouth.* MarketingLetters12:3(2001),211-223

[5] J. Goldenber g, B. Libai, E. Muller. *Using Complex Systems Analysis to Advance Marketing Theory Development.* Academy of Marketing Science Review 2001

[6] D. Kempe, Jon Kleinberg, E. Tardos *Maximizing the Spread of Influence through a Social Network* Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2003. p. 137–146.

[7] Minoux M. (1978) *Accelerated greedy algorithms for maximizing submodular set functions.* In: Stoer J. (eds) Optimization Techniques. Lecture Notes in Control and Information Sciences, vol 7. Springer, Berlin, Heidelberg

[8] *Cost-effective Outbreak Detection in Networks.* J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, N. Glance. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2007. Best student paper award.

[9] *Social Dynamics of Digg*. Hogg, T. and Lerman, K. (2012) EPJ Data Science 1(5). doi:10.1140/epjds5

[10] *Digg 2009 data set*, `https://www.isi.edu/~lerman/downloads/digg2009.html`

[11] *Emergence of scaling in random networks*, Barabasi, Albert-Laszlo; Albert, Reka. (1999), Science. 286 (5439): 509–512.

[12] *The Spread of Obesity in a Large Social Network over 32 Years* Nicholas A. Christakis, James H. Fowler (2007) N Engl J Med 2007; 357:370-379

[13] *Investigating the growth paths of young technology-based firms: a process approach* Hugo, O, Garnsey, E University of Cambridge, Working Paper, No. 02/01, July 2002, 29 pp.. Centre for Technology Management Working Papers. 02.