Applying trust metrics based on user interactions to recommendation in social networks

Alberto Lumbreras
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: alumbreras@lsi.upc.edu

Ricard Gavaldà
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: gavalda@lsi.upc.edu

***Abstract*—Recommender systems have been strongly researched within the last decade. With the arising and popularization of digital social networks a new field has been opened for social recommendations. Considering the network topology, users interactions, or estimating trust between users are some of the new strategies that recommender systems can take into account in order to adapt their techniques to these new scenarios. We introduce MarkovTrust, a way to infer trust from Twitter interactions and to compute trust between distant users. MarkovTrust is based on Markov chains, which makes it simple to be implemented and computationally efficient. We study the properties of this trust metric and study its application in a recommender system of tweets.**

I. Introduction

Recommender systems have emerged as an effective response to the so-called information overload problem in which users are finding it increasingly difficult to filter the huge amount of information they are exposed to. Finding useful information in the web is another challenge that recommender systems help to solve. With the popularization of social networks, recommender systems had a new scenario with its own particularities that had to be taken into account. Network structure and users interactions can give extra information to be considered by recommender systems. Trust-based recommenders usually combine trust between users with traditional recommender techniques to improve the quality of the recommendations. Computing trust between users is here a necessary step, and several proposals have been made [1] [2]. However, what trust is and how trust propagates across a social network is a problem that does not have a one-for-all solution. As network topology, as well as user behavior, change among networks, every network requires a particular analysis.

Twitter [1] is a social network that allows users to post 140-character text messages (or tweets). Users subscribe (or follow) to other users' publications. Twitter then shows a personalized live feed (or timeline) to every user by aggregating the tweets of people they follow (followees) by time of publication. Also, Twitter users can interact between them by means of forwarding a friend's tweet to their respective followers (retweet) or by mentioning other user to chat. The more people a user follows, the more her timeline shows more tweets and it becomes harder to read them all. This is an interesting problem for a recommender system, that might help users to filter, order or discover really interesting tweets.

We propose a method to compute trust on Twitter. This method estimates trust from user interactions and computes its propagation through a simple algorithm that we call MarkovTrust. MarkovTrust is similar to other trust propagation algorithms proposed in the literature. Our contribution is evaluating how this family of trusts performs on Twitter, and whether it can improve the quality of recommendations.

To evaluate recommendations we propose an architecture for social recommendations. Here we explore different techniques at three different levels: crawling, trust propagation and text mining. Crawling aims to get the optimum set of neighbors for every user, in such a way that the value of items published in this neighborhood is maximized. Trust and trust propagation aim to compute trust between pairs of users and use this information to enhance recommendations with social information rather than just using content-based recommendations. Text mining focuses on how to encode tweets and how to compute similarities to use in a content-based recommendation of tweets.

II. Related work

There have been some proposals to compute trust on the web. EigenTrust [3] considers trust as a function of successful interactions (e.g. a file download from a node) in a peer-to-peer network. A direct trust matrix is created with this information from every pair of nodes. Trust propagation is computed by calculating its eigenvector matrix. This gives a local metric of trust from every pair of nodes.

Advocato (http://advogato.org) is based on Levien's trust metrics [4]. Within Advocato, each user has a single rating calculated from the perspective of a group of *seed* nodes. Trust has three levels (apprentice, journeyer and master) and its propagation is computed using a network flow model. The computed trust is global, thus assigning a single trust value to every node which is trusted similarly from every node in the network. Advocato is designed to make public key certification more attack-resistant.

In [5], a method similar to Advocato is proposed, but using spreading activation strategies instead of maximum flow to compute a local trust value instead of a global one. Appleseed

[1] www.twitter.com

is also a group trust metric. This means that trust is computed from a set of seed nodes and therefore, for a given node, its trust is shared within a group. Like EigenTrust, Appleseed is based on finding the principal eigenvector of a trust matrix. This is the reason why it cannot deliver an absolute trust value as trust is normalized, but a ranking of nodes according to their trust.

Richardson et al. [6] aim to compute the belief a user may have in another user's statement. They propagate trust by finding paths from the source to every node which holds an opinion on the statement. They concatenate trust along the paths and finally aggregate trusts along the found paths.

All these methods normalize trust before computing trust propagation. This make trusts from different users incomparable, as a user with more trusted nodes will assign a lower level of trust to each of them (trust is limited and shared among peers). Golbeck proposes TidalTrust [1] to avoid this problem by keeping the original trust scale while trust propagation is computed. TidalTrust proposes using FOAF meta-data to compute trust for recommendations. It extends the FOAF ontology adding a trust level that ranges from 1 to 10. Trust is explicitly annotated by users, and then propagated by the TidalTrust algorithm.

Song et al. [7] propose an information flow model based on Markov chains to predict user behavior based on early adopters behaviours. They aim to solve the following question: "If one user or multiple users access an item, who else is likely to follow these early adopters and access this item next?." They designed a general model and a topic-sensitive model, and consider timestamps to deal with the temporal notion of adoption.

In this paper we propose a method similar to that of EigenTrust adapting the computation of direct trust from a P2P network to the Twitter social network. As EigenTrust, we will infer direct trust analyzing the interactions between users. Inferring trust from interactions in Twitter has an additional problem of fetching these interactions. There are several types of interactions in Twitter and thus this translation from interactions into trust will have to be carefully analyzed.

In order to summarize, we address here two main questions:

(a) how can we adapt existing trust metrics to Twitter?
(b) does trust information help in recommending tweets?

For the first question we will adapt EigenTrust to our scenario and evaluate it in a similar way to that of TidalTrust, as it is also a recommender system scenario. For the second question, we will develop a basic recommender system framework for social networks. Once we have a set of candidate tweets for a given user, we will test whether adding trust information improves classical recommendation methods.

III. Trust on Twitter

If Twitter had an explicit feedback mechanism, that is, the possibility for users to rate other users (or their tweets), we could create a trust metric that considers the positive feedback towards other users. The more positive feedback a user u has given over a user v, the higher confidence on the assertion that user a trusts user b. If Twitter had a FOAF-like tagging like the WOT (Web Of Trust) proposed by Golbeck [1] where users explicitly annotate trusts on other users, we could use (not considering computationally costs for this scenario) their TidalTrust algorithm to propagate trust.

But we do not have such a feedback or annotation mechanism on Twitter. Instead, we have to analyze the interactions between users to indirectly estimate how much a user trusts another one. Using the Twitter public API we can get information on this interactions. We propose using these interactions and translating them into a measure of direct measure of estimated trust.

Twitter users interact in three ways: (a) following other user tweets (b) retweeting other user's tweet (c) mentioning another user or (d) favoriting another users' tweet. The question here is how these interactions can be cast to a measure of direct trust.

- The action of a *following* b means that a wants to receive all the tweets published by b. A user can start following someone for many reasons. The user might have been recommended by Twitter or by a friend. A common case is that b has been retweeted or mentioned many times by a person that a is already following, and a eventually decided to follow b because a thought that b publications are interesting. Following is not a clear indicator of trust. Users follow users (apart from cases of personal friendship) because they think they will be interested on what they post. However, if after a period of time this profile was not as interesting as expected, users can stop following (unfollow) others. Therefore, we cannot infer much information from a following b.
- The action of a *retweeting* a tweet t from user b means that a found the content of t (or the link it refers to) interesting, and a expects her friends to like the content of t as well. If a tends to retweet b a lot, it can be inferred that a trusts b at some level. It is reasonable to think that the more a retweets b, the more confident we can be that a trusts b. Our trust model is gradual, that is, we do not compute the probability of a trust relation, but its strength. Therefore we can say that the more a retweets b the stronger is her trust on b.
- The action of a *mentioning* b on a tweet t means that a wants b to read the tweet. It can be a single tweet or a whole set of crossed mentions between a and b (e.g. a discussion). Since mentions can be used in so many ways (e.g. expression of disagreement or notification) we cannot be sure, without further analysis, whether a mention expresses trust, distrust, or none of them. However, we think mentions usually express some kind of trust relationships (people tend to relate to those people who think like them) so we will consider a mention as a positive indicator of trust.
- The action of a *favoriting* a tweet t from b may have two meanings. One possibility is that a uses favorites as a "read it later", usually a tweet with some link to external content. Another possibility is that a wants to

keep this tweet because she liked it (e.g. it was funny, or has some content that a wants to access easily in the future). Therefore, we can say that favorites mostly express a trust relation. However, as Twitter does not provide in the API the time when a user favorites some tweet we do not to consider favorites.

Mentions and retweets express trust, but they might not express trust with the same strength. We are sure about retweets, but mentions contain more ambiguity. As we will see later, we propose weighting these interactions to fine tune the trust model.

Some Twitter users have specialized profiles on some topic (e.g. politics or machine learning) and some others have general profiles and post about many topics. An ideal computation of trust would detect the topics of every tweet involved in a mention or a retweet, and would increase trust only on those topics. Though the problem they tackle is different, such a topic-based influence is considered for instance in [7]. For simplicity's sake, we compute trust in a general way, considering that if a trusts b on a topic T, a trusts b on any topic.

We consider users trust their friends proportionally to the number of interactions. Given a user u, she shares her total trust between every user she has interacted with (mentioned or retweeted). For instance, if user a had 10 interactions shared amongst user b (3 retweets), user c (5 retweets) and user d (2 retweets), her trust t_{ab} is 0.3, trust t_{ac} is 0.5, and trust t_{ad} is 0.2. Note a user a can interact with a user b even if a does not follow b. For instance, if user b published a tweet t and another user which is followed by a retweets t, t will be published on a's timeline. As mentioned before, we will consider weighting interactions according to whether it is a mention, or retweet. It can be formally expressed by the next formula:

$$t_{ij} = \frac{wN_{ij}^{(m)} + (1-w)N_{ij}^{(rt)}}{N_i} \tag{1}$$

where N_{ij} is number of interactions from i to j, N_i is the number of total interactions originated by a, and w is the weight assigned to retweets against mentions. Our intuition is that w should have a value between 0.5 and 1. We chose $w = 0.5$. Superindexes denote mention interactions (m) or retweet interactions (rt).

Note that by assigning a limited amount of trust to share among friends, we are losing the real magnitude of trust. Instead, what we get is an ordered list of users by their assigned trust.

A. Trust propagation

So far we have discussed how to infer trust strength between two users (a and b) when a has directly interacted with b. However, most users have never interacted between each other. Let c be a user followed by b and not by a. Let t_{uv} be the trust of user u on user v. The goal of a trust propagation model is to compute t_{ac} from trust t_{ab} and t_{bc}.

Note that equation 1 can be seen as transition probabilities of a Markov chain if t_{ij} is seen as the probability of an interaction from i to j. If the network is interpreted as a Markov chain, we can apply a *random walk* model considering trusts as probabilities and, in general, considering trust t_{ab} as the probability for a to reach b. In Markov models, the *t-step distribution* is the distribution after taking t steps from the starting distribution. It is denoted by $\Pi^t = \Pi^0 P^t$ where Π^0 denotes the initial probability distribution over states and P^t is the transition matrix P raised to the t-th power. As we are considering trust from every user, the initial probability is set to 1 for every user and P is then a vector of ones. Thus we have $\Pi^t = P^t$. In order to avoid confusion between *steps* and *trust*, we denote steps as s. If considering trust as transition probabilities, the probability of walking from node i to node j after s steps can be expressed as:

$$t_{ij}^{(s)} = \begin{cases} t_{ij} & \text{if } s = 1 \\ \sum_k t_{ik} t_{k,j}^{(s-1)} & \text{if } s \geq 1 \end{cases} \tag{2}$$

where the probability of i reaching j after s steps is seen as the probability of taking a single step to some vertex k and then taking $s-1$ steps to j. Note that the model takes into account every path from i to j and with a maximum of s steps.

However, instead of having a fixed number of steps s, we want to consider that users have an horizon of trust of one step, two steps and so on up to s steps. Moreover, we want to apply a decay probability to tune the importance of closer neighbors. In a matrix form, we can express this as:

$$T^s = \frac{1}{s} \sum_{n=1}^{s} \alpha_n \prod_{m=1}^{n} P^m \tag{3}$$

where P is the initial transition matrix with components t_{ij} and α is a decay factor which value will be discussed in section VI. Our intuition is that $s = 3$ is a good choice. Inferring trust further than this would lead to a lower accuracy on trust. To avoid dead end nodes, for those users with no outcoming interactions we artificially assign an equal amount of interactions to very other user in the network.

To reduce computing cost we truncate the transition matrix Q by deleting every trust value below a threshold. We will explain this truncation later. If P has $n \times n$ dimensions, with a naive computation it takes a time of $O(sn^3)$.

B. Temporal dynamics of trust

Users change their preferences over time. It can be caused by a personal shifting on her interests or because new topics appear and old topics become obsolete. In the first case, most changes occur slowly and gradually, as are more attached to the user's personal interests. In the former case, changes occur very fast (e.g. breaking news that the user is interested on).

Another interesting aspect is temporal dynamics on the trust relationships. Over time, Twitter users start following some new users and unfollow some of their old friends. Twitter users can do so to optimize their twitter feed with information or people they are really interested on.

Ideally, these dynamics should be considered by a recommender system. But it is not the main goal of this system to deal with this kind of dynamics. However, and as the trust model drives the crawling, which is a critical part of the system, we apply a simple decay model to capture some of the trust dynamics. Our model ages the old interactions so that the newer prevail when profiling the user. We apply a forgetting factor to old interactions of a user every time she creates a new one. New retweets imply a decay on old retweets and new mentions imply a decay on old mentions. Let $\vec{v_u}$ be the vector of interactions from user u. Let $\vec{u_a}$ a new vector with one element set to one and the rest set to zero. The n-th element corresponds to the user that received the interaction from u. The decay can then be expressed as:

$$\vec{v_u} = \lambda\vec{u} + (1-\lambda)\vec{v_u} \tag{4}$$

where $\vec{b}$ is a vector with all zeros but a one in the position of user b. The more dynamic the trust is, the higher value λ should have to age old interactions. Our intuition is that trust does not change dramatically and a $\lambda = 0.1$ will suffice.

Note that this decay is applied to the transitions matrix, not directly into the trust matrix. The former will be computed from the first.

IV. Limitations of our setting

As researchers, analyzing a network such as Twitter brings some limitations. First, we do not have access to the whole network. Twitter freely provides a public API with strong limitations for developers in order not to overload their network. Using this API limits the extension and frequency of the crawling that our system can make to fetch either tweets or interactions between users. Our crawler has to fetch Twitter data without reaching the API limits. As a consequence, there is a limitation of the users of our system (target or seed users) and a limitation on how many neighbors of these users we crawl.

Recommender systems can be tested online or offline. Online approaches test the algorithms on real time, giving us a real feedback on how our system behaves out of the lab. However, online tests are not trivial to design and they are costly as any modification in the algorithms will have to be tested again. Offline tests can use past behavior of users to use it as test set. This allows us to run the algorithms over an over again, but the results will not be as accurate. Within this approach we assume that past behavior is a good model of present behavior, which is not true specially when dealing with rapidly changing contexts such as Twitter. What was interesting one week ago might have lost its information value today. Because of time limitation, we followed this offline approach on our tests.

These factors prevent us from getting to strong conclusions over the results of our work. We will therefore limit ourselves to discuss the obtained results to get an intuition of the quality of our model and its possible enhancements.

V. Architecture

To test the MarkovTrust on Twitter we designed a system that crawls users and fetches their interactions. After a period of crawling, we will be able to see how MarkovTrust performs and whether it can serve as a basis for recommending tweets.

The system is divided in two modules: a crawler and a recommender. The crawler updates a list of most influential neighbors for every target user and fetch their tweets. It stores these tweets for the recommender to use them as its item database. The recommender learns a model for every user from the user tweets, retweets, and mentions and later makes personalized recommendations of tweets for every target user. The next sections describe these two modules in more detail.

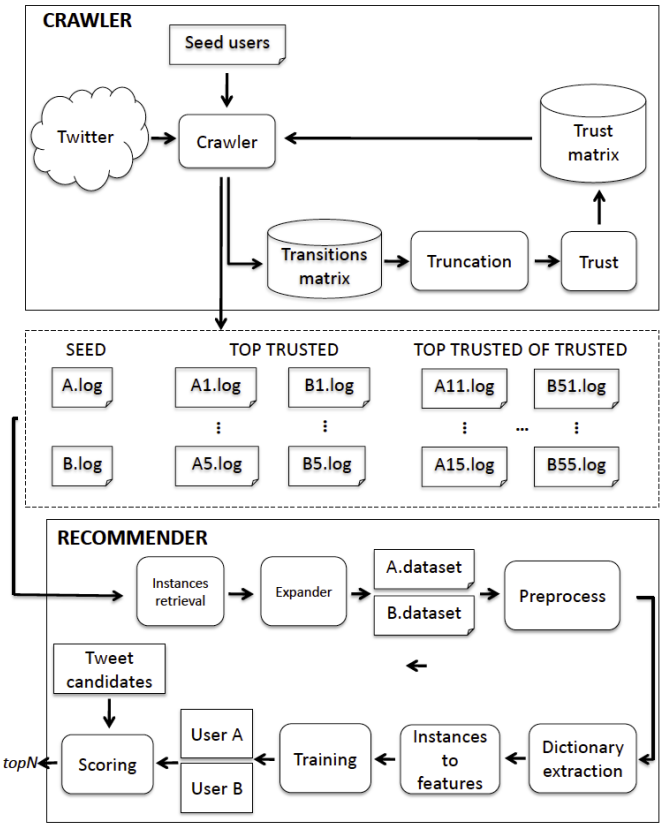

Fig. 1. Architecture of the system considering two seed users A and B

A. Crawler

At the first cycle, the crawler has only a collection of target users that have registered in the system. These users are those who will receive recommendations from the system. They serve as seed nodes for the crawler to start looking for trusted users in the neighborhood. In the next step, the algorithm iterates through every user to get their last published tweets. Every tweet is then parsed looking for interactions (mentions and retweets) and the interaction matrix is then updated. When there are no more users left, the interaction matrix is truncated by leaving only the n-top interacted users of every target

user. Using the resulting matrix as the transition matrix, trust between every pair of remaining users is recomputed. From the final trust matrix, we get a new list of m-top trusted users that will be crawled in the next cycle. The system then sleeps for some hours in order not to overload the Twitter API.

1) Crawling: The aim of the crawler is to get a collection of tweets that maximizes the probability of being liked by the seed users. If trust is well computed and is significant to what users like, a good crawling would improve the possibility of making good recommendations as the items to choose from will be an interesting subset for the user. In other words, crawling should maximize the signal to noise ratio.

The selection of which paths to crawl can follow two general strategies. The first one is a similarity-based strategy. We can rank the neighbors of the seed users by how similar they are to the target seed user, and select the n-top similar. A variation of this strategy would be a more elaborated collaborative filtering strategy that recommends users that the seed user might like. However, both techniques require fetching the candidates' profiles *a priori* to create profiles and compute similarity between users. This option is discarded for the intensive (and not allowed) use of Twitter API required. A second major strategy is that of a trust-based crawling and is the one chosen for our system as it only needs to analyze the output interactions from target users, which can be done by a single query that retrieves the last tweets of a user. New users are selected by ranking the neighbors of every seed user by how much the seed user trust them. To compute this trust we only need to parse the profile of the seed user and count the retweets and mentions to every other user. From this we can compute a trust matrix as explained in Section III.

To summarize, the crawling strategy follows the next rule: For every seed user crawl her profile and the profiles of her "top trusted" users, and for every user in the "top trusted" set crawl the profiles of her "top trusted". We chose the size of the "top trusted" sets to be 5. Therefore, the system crawls the top trusted neighbors up to a distance of two steps. We consider this distance to be good enough to build an interesting set of tweets to recommend. Crawling further will increase the noise, making our recommendations achieve a possibly better recall but far less precision.

Crawled profiles are stored in separate files (one file per user) to be processed later. The information stored for every tweet is: Type, user, tweet ID, timestamp, tweet, referred user and trust. Type can be "normal" (a simple tweet with no interaction with other users), "mention", or "retweet". Referred user is an optional field that remains empty for normal tweets and for mentions and retweets, contains the user which our target user interacted with. If more than one user is contained in a tweet only the first one is considered. Trust is the computed trust of the author of the tweet on the referred user, if any. When the trust system has not computed any trust for this reference user this value is set to -1.

2) Pruning: As the number of iterations grows and new users are discovered the interactions matrix also grows. Computing trust propagation at three steps has a cost $O(3n^3)$, which is polynomial in the number of users n but costly in practice if we let n grow arbitrarily. An option is to limit n so that the size of the matrix stabilizes at some feasible size. As seen in Equation 3, the dimensions considered are given by the dimensions of the transition matrix Q. In order to truncate Q, we delete those users in the matrix with a low level of interactions. More specifically, for every user u in the network we keep the edges to the n-top trusted users and delete the rest of the edges. Once we have delete the less trusted edges, those users with no incoming edges are deleted from the matrix. We chose $n = 100$.

3) Updating trust: After the transition matrix is updated and the less trusted users have been deleted from the matrix, the trust matrix T^s is recomputed. This matrix contains the trust index between every pair of seen users, considering that trust can be propagated up to two steps.

At every cycle the crawler reads the set of n-top trusted users for every seed user and fetches their lasts statuses. Note that, as Algorithm 1 shows, the system updates the transitions matrix for every node up to a distance of s. The new trust matrix is then computed from this transition matrix. From this trust matrix, the crawler will start a new cycle fetching the last tweets from the target users users (that are always the same), their top trusted nodes, the top trusted of the top trusted nodes, and so on up to a distance of s. This way, the crawler gets a collection of tweets from the top trusted users in the neighborhood of every target user.

Algorithm 1: Crawling cycle

```
while True do
    S ⟵ SeedUsers() ∪ TopTrustedUsers()
    foreach s ∈ S do
        statuses ⟵ GetLastUpdates(s)
        UpdateInteracctionsMatrix(s, statuses)
    end
    TruncateInteractionsMatrix()
    UpdateTrustMatrix()
    UpdateTopTrustedUsers()
end
```

B. Recommender

The aim of the recommendations module is to periodically (e.g. every day) get a rank of m-top items for every final user. For this, it analyzes final users' publications to create a user profile. Then, for every final user, it gets a list of tweet candidates to be recommended. This list is extracted from tweets published by its neighbors. From this list, a scoring function predicts a score for every candidate tweet. The highest m-top items are then showed to the user.

In the next section we explain this process in more detail.

1) Instance retrieval: First, a file is created for every target user with a set of positive and negative instances. As Twitter has no explicit mechanism of rating tweets, we consider a binary rating where retweets are tagged as positive. The

question now is how to get or identify negative tweets, that is, tweets the user is not interested in. We follow a similar reasoning to that of [8] where, in a web search scenario: they consider that if user u clicked through result 7 but not through result 6, he must have seen 6 but not felt interested in it. Here, we consider that given a user u and a user v and given two contiguous publications $v(i-1), v(i)$ from user v, if user u retweets $v(i)$ but not $v(i-1)$ then it means that user liked $v(i)$ and not $v(i-1)$. We are making two assumptions here: first, that user u read $v(i-1)$. This might be not true if $v(i)$ and $v(i-1)$ are very separated on time. In this case, when u checks his feed he might see $v(i)$ but not $v(i-1)$. However, we assume that users can read most of their feed. The second assumption is that the first assumption holds even when u is not subscribed (follower) of user v, that is, a tweet from v has been propagated through the network until reaching the feed of user u. Actually, that is why our interaction model does not consider direct friends but any user, as possible interactions are not limited to friends (followees).

To get positive and negative instances we read user u tweets and look for retweets. For every retweet, we check the author of the original tweet. If the author was between the n-top trusted nodes at the moment of the retweet, the crawled should have captured the original tweet beforehand and maybe the previous ones. Then, we add the original tweet as positive example and the previous one as a negative example. We use the original tweet because sometimes users modify the original one to add their opinion, write it in a personal way, or shorten some words to fit the tweet in the 140 characters allowed. If the original tweet of a retweet is not found in the logs, the example is discarded. Note that we could use Twitter API to get this tweets but this would increase the number of calls to the API.

2) Tweet Expander: Text similarity is a important pillar when recommending text items. Traditional methods to compute text similarity are based on bag-of-word representation of texts, to compute afterwards a similarity function such as cosine between the two vectors of words. When dealing with short texts the probability of a word coincidences falls and these methods give inaccurate results.

The aim of query expansion is to add more words to the tweet so that the word occurrences between tweets increase. We follow the work on [9] and apply a similar method to expand tweets. The idea of the method is to query a search engine with our tweet and append the words contained in the results to the tweet. First, we clean the tweet from any artifact that could tighten the search (url, hashtags, "RT", and usernames). Second, we query Bing search API. Every result contains some fields such as URL and description. Description is the snippet of text that is shown for every result. We get the snippets of text of the first 200 results and add them to the original text of the tweet. Note that the search engine might not find 200 results for our tweet query. In fact, given the extension and complexity of most of tweets in comparison to a traditional user query, Bing finds no results or just about five or ten in the best cases. The main risk of tweet expansion is getting much more noise (unrelated results) than signal (related results).

3) Preprocessing: The aim of preprocessing is normalizing the text (tokenization, filtering stop words and stemming) and cleaning it from non-text artifacts (numbers, url). As url can be informative they will be added later as a boolean feature that says whether a tweet contains a url or not.

4) Dictionary extraction: The aim of dictionary extraction is to create a dictionary of words that will act as word features. A common technique is to use most frequent words in the set of documents. This avoids having an excessively large set of word features and would not only slow down but make our classifier more inaccurate (the curse of dimensionality). However, as our dataset for every user is relatively small we add every word to the dictionary.

We create two dictionaries. The first one is a simple bag-of-words and the second is a term frequency weighted bag-of-words. In the second case, for every word in the dictionary we compute the log ratio between its occurrences in the positive examples and the occurrences in the negative examples. The formula is

$$tf = log \frac{POS_w}{NEG_w} \tag{5}$$

the *tf* value is 0 when the word is equally used in positive and negative examples. It will be positive when predominant in positive examples and negative when predominant in negative examples. It can be easily learnt from any classifier that words with $tf(w) = 0$ are non discriminant with no need of eliminating the word feature.

5) Instance to features: In this step we convert the instances into sets of features so that a traditional classification algorithm can work with them. We extract the following features: *trust*, *url*, *words* which we describe below.

trust is a real value [0,1] that indicates the trust from the target user to the user who made the original tweet (at the moment of the retweet).

url is a binary variable that indicates whether the tweet contains any url or not.

word is a set of features that expresses the content of the tweet. This information can be codified in different ways. The basic form is a boolean bag of words (one if the word is in the tweet and 0 if it is not) with words taken from the previously extracted dictionary. We can express the bag of words with the *tf* value of every word, as explained in section V-B4. We can also collapse these bag of words by computing the distance from the tweet to the corpus using the *tf* dictionary. We will use the first and the third encodings denoted as *bow* and *tf*.

6) Training and scoring: Finally the system is trained over all the past tweets and is ready to score a new tweet candidate. These candidates are taken from the top trusted neighbors. If we want the system to recommend the n-top tweets every day, candidate tweets will be those tweets in the neighborhood that have been published today (or after the last recommendation to the user was made).

Note that ten retweets talking about some new topic must be more important than one hundred tweets talking about another

topic two months ago. To give some adaptivity to the system in terms of concept drift, a decay factor should be applied to past tweets. For now, we are not considering any decay factor for tweets.

VI. Experiments

To test the system we selected 20 seed users from our twitter network. Their interests range from politics to professional coaching. Their main language is Catalan, Spanish, or English. We crawled these users and their neighborhood for six months. After processing users' logs we got an average of 314 instances which are balanced (around 50%-50% of retweets and non-retweets).

A. Validation of the trust model

We have no direct way of testing the accuracy of our trust propagation model. The ideal scenario would be having a full annotated network where every user has rated every other user. As we do not have such a network, we follow a similar approach to that of Golbeck [1] to see how our trust model does on direct neighbors. Later, we will test how trust information affects the accuracy of recommendations on this network. Golbeck did the following process for every user: For each neighbor n_i of the user (source), a list of common neighbors of the user and n_i was compiled. For each of those neighbors, the difference between the source rating and n_i rating of the neighbor (i.e. computed trust) was recorded as a measure of accuracy. We call this difference Δ. A smaller Δ means higher accuracy. But unlike Golbeck, we are not interested on the absolute value of trust but on the resulting ranking of users ordered by their given trust. We would like to know whether a common neighbor is similarly (say, highly) trusted by both users, without caring for what "high" means for each user. For instance, let a be a user whose most trusted friend is b, Let b be a user whose most trusted friend is c, and let c be also a friend of a. We would like our trust model to place c between the most trusted users of a. A solution can be to compare rankings of a and b. However, this can only be done when a and b rank the same users. To solve this we normalized the rankings of our users following the next normalization formula for every item in the original ranking:

$$normalized\_rank(r) = \frac{r-1}{R-1} \tag{6}$$

where r is the ordinal value of the item and R is the length of the rank. The normalized ranking has all its values into [0,1].

To analyze the impact of trust decay we compare behaviors of delta when no decay is used and when an exponential and a linear decay is applied. The formula for the exponential decay used is:

$$\alpha_n = c\frac{1}{1.5^n} \tag{7}$$

where c is a normalizing factor to make the sum of α's to be 1. And the formula for the linear decay is:

$$\alpha_n = c(n_{max} - n + 1) \tag{8}$$

TABLE I
Relation of trust rank and delta

Ranking (0-10)	Δ No decay	Δ Linear decay	Δ Exp.decay
(0-1)	1.14	0.90	0.87
(1-2)	1.10	1.22	1.09
(2-3)	0.96	1.05	1.05
(3-4)	1.07	1.18	1.20
(4-5)	1.09	0.81	1.01
(5-6)	1.13	1.11	0.92
(6-7)	0.91	1.08	1.16
(7-8)	0.90	1.08	0.99
(8-9)	1.3	1.34	1.34
(9-10)	1.26	1.11	1.16

Table I shows the average values of Δ at every position of the ranking of trust. The trust rank and Δ have been normalized to a [0-10] range. We would expect a smaller Δ in the top users, i.e. more agreement among common neighbors. However, we see no clear correlation between neighbors ratings and Δ. This might be caused either by our model being wrong or by transitivity not holding in this scenario.

If a user a trusts a user b, this can have a double meaning: First, it can reflect a general interest on what b publishes. As direct trust is computed by analyzing the interactions between users this is, by definition, a characteristic of our model; A second characteristic is that of transitivity when endorsing other users. If a trusts b and b trusts c, a would trust c if a) transitivity holds and b) the propagation model is appropriate. The general validity of our trust model will be further discussed in the next section. As for transitivity, the lack of correlation shown in the table seems to indicate that there is no apparent trust transitivity occurring in Twitter.

It would be interesting to know whether we do not see apparent transitivity on trust because trust is not transitive on Twitter, or because our trust model does not capture it appropriately. If the original interactions are transitive (a highly interacting with b and b highly interacting with c implies a highly interacting with c) we would infer that our trust model does not keep transitivity. One could even say that a model based on transitivity, such as MarkovTrust and most of trust models, might not be appropriate in such scenario.

Table II is similar to table I but uses the stored interactions instead of the computed trust. The table shows that there is a slight correlation but that it is opposite to the correlation we expected. It seems that, in terms of interactions, users agree less about those with whom they interact more. This might be an effect of our selective crawling or due to a lack of more data to make the patterns statistically sound. Anyway, no transitivity is apparent in the interactions.

Even if there is no evidence of transitivity, we observe a low Δ at every position of the rank. That is, either considering interactions or inferred trust, common neighbors are similarly trusted (or interacted).

TABLE II
RELATION OF INTERACTIONS RANK AND DELTA

Ranking [0-10]	Δ
$[0-1)$	1.36
$[1-2)$	0.75
$[2-3)$	1.14
$[3-4)$	0.83
$[4-5)$	0.78
$[5-6)$	0.52
$[6-7)$	1.06
$[7-8)$	0.53
$[8-9)$	0.57
$[9-10)$	0.17

TABLE III
ACCURACY OF RETWEET PREDICTION

	Cl.	Exp	Enc	Acc	Rec	Prec	F1	AUC
Trust	NB	yes	bow	50.76	59.43	52.07	55.51	50.93
			tf	52.38	58.99	52.52	55.57	52.58
		no	bow	48.79	53.01	47.76	50.25	49.88
			tf	51.97	50.49	50.60	50.44	51.51
	SVM	yes	bow	45.84	61.00	30.22	40.42	50.53
			tf	47.63	51.36	46.95	49.06	47.94
		no	bow	46.25	68.42	32.47	44.04	50.00
			tf	47.79	46.38	48.44	47.39	47.32
Averages				48.93	56.13	45.13	49.08	50.09
No trust	NB	yes	bow	47.02	57.58	48.90	52.89	49.56
			tf	51.13	49.56	54.81	52.05	52.22
		no	bow	49.28	47.98	50.76	49.33	50.54
			tf	46.12	45.18	45.84	45.51	46.28
	SVM	yes	bow	45.22	55.83	27.05	36.44	50.06
			tf	49.23	49.36	51.47	59.39	49.80
		no	bow	43.40	34.87	17.59	23.38	50.22
			tf	47.11	41.55	48.87	44.91	47.32
Averages				47.31	47.73	43.16	45.49	49.50

B. Influence of trust on recommendations

We would like to test whether our trust model is good enough to enhance the performance of a recommender system in Twitter. In order to test this, consider the recommender system explained in section V. Given a set of retweets a user has made (positive instances) and another set not retweeted by the user (negative instances), our recommender system tries to learn a model to predict whether a non-seen tweet will be retweeted by the user. As we are applying offline experimentations and we do not have access to future tweets, we split this collection of tweets on training set (75%) and test set (25%). As these test tweets are past tweets, we know whether the user retweeted them or not. In other words, the instances are all tagged. We compared models in a search space as shown in III. We tested both SVM (RBF kernel with $C = 1.0$ and γ set to the inverse of the number of features) and Naive Bayes classifiers combined with the multiple techniques explained in section V: Trust, tweet expansion and encoding of word features. Metrics used are accuracy, recall, precision, F1, and AUC.

For intuition, table III shows the averages of the models with and without trust for different metrics. Though the small size of the dataset does not allow to draw rigorously sound conclusions, we observe that models using trust information seem to perform better, and in particular recall seems to be most benefited without compromising precision. However, note that we are using a class-balanced set so that the baselines are all 50%. These poor results of our recommender are probably due to the difficulty of the task at hand and the simplicity of our model. Using more sophisticated attributes would probably enhance these results.

VII. CONCLUSIONS

In this paper we presented MarkovTrust, a model to estimate trust between users based on users interactions. We applied this trust model to Twitter and studied some of its properties. To test the utility of MarkovTrust we developed a recommender system framework that first crawls the Twitter network following the top trusted neighbors of target users, and then makes tweet recommendations based both on past retweets and estimated trust. Though not statistically sound due to lack of data, experiments show that such a trust model does not respect the idea of transitivity of trust. However, recommendations are enhanced when using MarkovTrust which makes us think that the model has some good properties for trust-based recommender systems. We think further study of these properties, as well as the properties and patterns in micro-blogging social networks, can shed interesting results in the future when applied to more complete recommender systems. Likewise, further analysis should be done on the marginal contribution of mentions and retweets to the accuracy of trust.

ACKNOWLEDGEMENTS

This research work has been supported by the Spanish Ministry of Education and Science (projects TIN2008-06582-C03-01 SESAAME and TIN2011-27479-C04-03, BASMATI), by the EU PASCAL2 Network of Excellence, and by Generalitat de Catalunya (2009-SGR-1428).

REFERENCES

[1] J. Golbeck, "Computing and applying trust in web-based social networks," Ph.D. dissertation, University of Maryland at College Park, 2005.

[2] P. Massa and P. Avesani, "Trust-aware recommender systems," in *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 2007, pp. 17–24.

[3] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigentrust algorithm for reputation management in P2P networks," in *Proceedings of the twelfth international conference on World Wide Web - WWW '03*. New York, New York, USA: ACM Press, 2003, p. 640.

[4] R. Levien, "Attack-resistant trust metrics for public key certification," *Proceedings of the 7th conference on USENIX Security Symposium*, vol. 7, 1998.

[5] C.-N. Ziegler and G. Lausen, "Spreading activation models for trust propagation," *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04)*, 2004.

[6] M. Richardson, R. Agrawal, and P. Domingos, "Trust Management for the Semantic Web," *Proceedings of the Second International Semantic Web Conference*, 2003.

[7] X. Song, B. L. Tseng, C.-Y. Lin, and M.-T. Sun, "Personalized recommendation driven by information flow," *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '06*, p. 509, 2006.

[8] T. Joachims, "Optimizing Search Engines using Clickthrough Data," *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[9] M. Sahami and T. D. Heilman, "A web-based kernel function for measuring the similarity of short text snippets," *Proceedings of the 15th international conference on World Wide Web WWW 06*, vol. pages, p. 377, 2006.