TweetRank
An adaptation of the PageRank algorithm to Twitter world

V. Hallberg, A. Hjalmarsson, J. Puigcerver,
C. Rydberg and J. Stjernberg
{victorha,alehja,joanpip,chrryd,stjer}@kth.se

May 14, 2012

Abstract

PageRank is an algorithm presented by Larry Page and Sergey Brin that allows to estimate the importance of a web page using the hyperlinks among them. This algorithm was originally the core of Google's search engine and proved to work very well in the web structure. Our task was to adapt the PageRank to work in the world of Twitter, instead of the web. This document presents the mathematical basis to compute, what we called, the *TweetRank*, an algorithm to compute an approximation to *TweetRank* based on Monte Carlo methods, and a simple search engine designed as a demonstration of our work.

Contents

1 Introduction

Page and Brin defined the importance of a web page as the probability that at a certain time [7], a person that randomly clicks on links will be visiting that web page. This assumption, called *random surfing*, works really well on the World Wide Web since the navigation is driven by hyperlinks among web pages.

Formally, the PageRank is the largest real eigenvector of the matrix G defined as it follows:

$$G = \alpha R + (1 - \alpha) L \tag{1}$$

Each element $G_{i,j}$ represents the probability of visiting the web page j given that the user is in the web page i. $G_{i,j}$ is defined as the weighted summation of $L_{i,j}$ and $R_{i,j}$.

$L_{i,j}$ represents the probability of visiting the web page j following one of the links of the web page i, and $R_{i,j}$ represents the probability of randomly visiting the web page j from web page i. $R_{i,j}$ is usually set to $\frac{1}{|W|}$ and it represents the probability of accessing the web page j by a random access.

In the world of Twitter, considering only direct references or hyperlinks among tweets it is not enough powerful to describe the importance of a Tweet, because the connectivity in Twitter is sparser than the Web. Some authors realised about this problem and presented alternatives to PageRank for ranking tweets in [4, 6]. Other authors [4, 5, 9] have described how to use PageRank to rank users based on the followers, the replies, etc. None of them have proposed a pure PageRank-based algorithm to rank tweets. S. Ravikumar et al. presented in [8] an approach that first computes the PageRank for users, and then distributes it among tweets. This is similar to what we do to avoid the sparsity of Twitter, however our approach embeds this into the PageRank computation itself and not as a separate step.

In the next section, we present some basic concepts about Twitter that will be used to define the TweetRank.

2 Twitter concepts

Twitter[1] is a social network that allows users to send and read short messages called *tweets*. There are many ways of interaction in Twitter: users can *follow* other users and are notified about the posts from these *followed* users, users can *mention* other users on their tweets, they can re-post a tweet from an other user (*retweet*) or they can *reply* to a certain tweet. Additionally, a tweet can mention several *hashtags*, which are usually used to designate the topic that the tweet is talking about (for instance: *#politics*, *#Twitter* or *#ChampionsLeague*). Given that concepts[2], we define the following matrices representing the different relationships.

The matrix RT describes the *retweet* relationship among different tweets.

$$RT_{i,j} = \begin{cases} 1 & \text{if tweet } i \text{ retweets } j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

[1] `http://www.twitter.com`

[2] `http://support.twitter.com/entries/13920-frequently-asked-questions`

Notice that RT is a $|T| \times |T|$ matrix (T is the set of tweets), where each row has at most one non-zero element (since a tweet i can only retweet a single tweet j) and the relationship is asymmetric (if a tweet i retweets j, then j does not retweet i).

The matrix RP describes the *reply* relationship among different tweets, with the same properties of RT.

$$RP_{i,j} = \begin{cases} 1 & \text{if tweet } i \text{ replies } j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The matrix MN describes the *mention* relationship among tweets and users. MN is a $|T| \times |U|$, where U is the set of users.

$$MN_{i,j} = \begin{cases} 1 & \text{if tweet } i \text{ mentions user } j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Notice that $m_i = \sum_{j \in U} MN_{i,j}$ is the number of mentions in the tweet i.

The matrix FW describes the *following* relationship among users. FW is a $|U| \times |U|$ matrix.

$$FW_{i,j} = \begin{cases} 1 & \text{if user } i \text{ follows user } j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Notice that $f_i = \sum_{j \in U} FW_{i,j}$ is the number of users followed (*friends*) by user i.

The matrix HT describes the relationship among tweets and hashtags. FW is a $|T| \times |HS|$ matrix, where HS is the set of hashtags.

$$HT_{i,j} = \begin{cases} 1 & \text{if tweet } i \text{ includes hashtag } j \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Notice that $th_i = \sum_{j \in HS} HT_{i,j}$ is the number of hashtags included in tweet i (row sum). Similarly, $ht_j = \sum_{i \in T} HT_{i,j}$ is the number of tweets that used the hashtag j (column sum).

3 TweetRank definition

TweetRank also defines the relevance of a tweet as the largest real eigenvector of a matrix G', as PageRank did. However, our matrix will take into account not only the direct relationships among tweets (retweets and replies). The idea is that tweet j can be accessed from tweet i by:

- $R_{i,j} = \frac{1}{|T|}$, random access probability.
- $L_{i,j}$, probability of accessing tweet j following a retweet or reply on tweet i.
- $M_{i,j}$, probability of accessing tweet j following a mention on tweet i.
- $F_{i,j}$, probability of accessing tweet j accessing the owner of tweet j from the friends of i's owner, and then randomly choosing j.
- $H_{i,j}$, probability of accessing tweet j accessing a hashtag used by j from the hashtags used by i, and accessing randomly choosing i.

Given that, elements in G' are defined as (with $\alpha+\beta+\gamma+\delta+\epsilon=1$):

$$G'_{i,j} = \alpha R_{i,j} + \beta L_{i,j} + \gamma M_{i,j} + \delta F_{i,j} + \epsilon H_{i,j} \tag{7}$$

$L_{i,j}$ approximation

$$L_{i,j} = RT_{i,j} + RP_{i,j} \tag{8}$$

The previous equation uses the result of equations 2 and 3. Note that a tweet can be either a retweet, a reply or none of them (new post). So, $RT_{i,j} + RP_{i,j}$ is equal to 0 or 1, and there are at most one non-zero element in the i-th row of matrix L.

$M_{i,j}$ approximation

First, probability of visiting the profile of a user j from a mention in a tweet i is defined as $FM_{i,j}$. Remember that m_i was defined as the number of mentions in tweet i. This definition uses the result of equation 4. Then $M_{i,j}$ is defined in 10.

$$FM_{i,j} = \begin{cases} \frac{MN_{i,j}}{m_i} & \text{if } m_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$M_{i,j} = \frac{FM_{i,u_j}}{|t \in T : u_t = u_j|} \tag{10}$$

Note that $|t \in T : u_t = u_j|$ is just the number of tweets from user u_j. This means that the user visits a random user profile mentioned by a tweet, and then visist a random tweet of that user.

$F_{i,j}$ approximation

In the first place, probability of visiting the profile of user j accessing the following list of user i is defined as $FF_{i,j}$. Notice that f_i is the number of users followed by user i. Using this probability, $F_{i,j}$ is defined in 12.

$$FF_{i,j} = \begin{cases} \frac{FWi,j}{f_i} & \text{if } f_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$F_{i,j} = \frac{FF_{u_i,u_j}}{|t \in T : u_t = u_j|} \tag{12}$$

Note that $|t \in T : u_t = u_j|$ is just the number of tweets of user u_j. This means that the user visited a random user profile followed by the owner of a tweet, and the visited a random tweet of that user.

$H_{i,j}$ approximation

First, let's define the probability that a tweet j is visited from tweet i, using the hashtag k. Remember that ht_k denotes the number of tweets that use the hashtag k and th_i denotes the number of hashtags that are included in tweet i.

$$HP_{i,j,k} = \begin{cases} \left(\frac{HT_{j,k}}{ht_k}\right) \cdot \left(\frac{HT_{i,k}}{th_i}\right) & \text{if } ht_k > 0 \wedge th_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

Note that $HP_{i,j,k}$ will be greater than zero if, and only if, both i and j included the hashtag k.

Then, $H_{i,j}$ is defined as:

$$H_{i,j} = \sum_{k \in HS} HP_{i,j,k} \tag{14}$$

3.1 Weighting using *hashtags*

An extension of the previous definition of G' could take into account some measure of similarity between users. Our idea is that users that share common interests would rank higher tweets from users with similar interests. We propose to use the *hashtag* concept on Twitter to measure what a user talks about. However, as we will discuss later, the assumption behind this approach might be not true in many cases as we will discuss later.

Hashtag similarity

The matrix HT' represents the relation among users and hashtags, that is, wich hashtags are used (and how many times) by each user. HT' is defined as it follows.

$$HT'_{i,j} = \text{Number of times that user } i \text{ uses the hashtag } j \tag{15}$$

Each row i in the matrix HT' represents a feature vector $\vec{h}_i$ of the user i. The similarity between two users i and j is defined as the cosine of the angle formed by the vectors $\vec{h}_i$ and $\vec{h}_j$.

$$d_{i,j} = \frac{\vec{h}_i \cdot \vec{h}_j}{|\vec{h}_i| \cdot |\vec{h}_j|} \tag{16}$$

G' extension to G''

Given $Z = \beta' L + \gamma' M + \delta' F + \epsilon' H$ (being $\beta' = \frac{\beta}{1-\alpha}$, $\gamma' = \frac{\gamma}{1-\alpha}$, $\delta' = \frac{\delta}{1-\alpha}, \epsilon' = \frac{\epsilon}{1-\alpha}$), G' can be expressed as:

$$G' = \alpha R + \beta L + \gamma M + \delta F + \epsilon H = \alpha R + (1-\alpha) Z \tag{17}$$

Given that, we extended the definition of G' and Z to take into account the similarity between two users defined before.

$$Z'_{i,j} = \frac{d_{u_i,u_j} Z_{i,j}}{\sum_{j \in T} d_{u_i,u_j} Z_{i,j}} \tag{18}$$

Note that this definition presents a little problem with users that have orthogonal feature vectors or zero-vectors: the distance between them will be 0, and the $Z'_{i,j}$ element will be 0 as well. A constant value $k > 0$ to d_{u_i,u_j} is added to ensure that as fas as $Z_{i,j}$ is different to 0, $Z''_{i,j}$ will be different to zero as well.

$$Z''_{i,j} = \frac{(d_{u_i,u_j} + k)Z_{i,j}}{\sum_{j \in T} (d_{u_i,u_j} + k)Z_{i,j}} \tag{19}$$

Thus, the weighted-probability of visiting the tweet j from tweet i would be expressed by the matrix G'' defined as:

$$G'' = \alpha R + (1 - \alpha)Z'' \tag{20}$$

Disadvantages of similarity weighting

This extension presents some disadvantages. The first one is that adding a new parameter k adds more complexity to our system and makes harder to select the optimal values for the model parameters. Choosing a value for k too high would vanish the effect of the weighting and a value too small would make even sparser the graph matrix of Twitter, which was already much sparse.

The second problem is given by the assumption which this approach relies on: the fact that users would rank higher tweets from other users with similar interests. For example, suppose two users that use to post comments about politics: it seems unlikely that a conservative person would give an high score to tweets from progressive users. Even if they share the same hashtag, the opinion expressed in both tweets will be very different.

Because of these two reasons we did not implement this extension in our TweetRank computation. However, notice that an adaptation of algorithm 1 can be easily implemented to take into account that score. The only thing that has to be changed is how the visit counter is updated.

4 Implementation

4.1 Overview

We implemented a small search engine that uses TweetRank to rank the tweets crawled from Twitter. The search engine has three main components: the crawler module, the *ranker* module, and the Solr module. Figure 1 represents the communication among the different components.

The crawler continuously fetches tweets from Twitter users using the Twitter API, saves the data to disk as a backup, parses it to extract the required information for our algorithms, and sends the parsed data to the ranker and the Solr module. The crawler sends the data to Solr using XML its XML POST request specification and it sends data to the ranker using a protocol designed by us. The crawler was designed to run in multiple threads or multiple machines.

4.2 TweetRank

Algorithm 1 is used to compute the TweetRank score and it is an adaptation of the classic Monte Carlo complete path stopping at dangling nodes method. The graph from Twitter G is

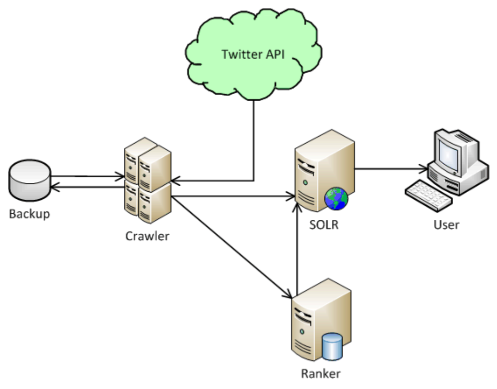


Figure 1: Overview of our search engine.

just the set of relationships described in section 3. Algorith 1 includes some predefined functions that are not described here, however we thought that their name is descriptive enough.

Algorithm 1 Compute TweetRank using MC complete path stopping at dangling nodes

```
Input: Probabilities $\alpha, \beta, \gamma, \delta, \epsilon$ such that $\alpha + \beta + \gamma + \delta + \epsilon = 1$, Twitter graph $G$ and $M \in \mathbb{N}$ such that $T \times M \in O(T^2)$.
Output: $\pi$, TweetRank of tweets in $T(G)$.
  $V_t \leftarrow 0, \forall t \in T$ {Visit counter for tweets}
  $CP_0 \leftarrow \alpha$ {Cumulative probabilities for each action}
  for all $t \in T(G)$ do
    for $m = 1$ to $M$ do
      $ct \leftarrow t$
      $stop \leftarrow False$
      while $\neg stop$ do
        $CP_1 \leftarrow CP_0 + \beta \cdot hasRetweetOrReply(ct)$
        $CP_2 \leftarrow CP_1 + \gamma \cdot hasMentions(ct)$
        $CP_3 \leftarrow CP_2 + \delta \cdot hasFriends(user(ct))$
        $CP_4 \leftarrow CP_3 + \epsilon \cdot hasHashtags(ct)$
        $r \leftarrow UniformRandomNumber(0, CP_4)$ {This ensures that all rows sums 1.0}
        if $r \leq CP_0$ then
          $ct \leftarrow jumpToRandomTweet()$
          $stop \leftarrow True$
        else if $r \leq CP_1$ then
          $ct \leftarrow jumpToRetweetOrReply(ct)$
        else if $r \leq CP_2$ then
          $ct \leftarrow jumpToMentionTweet(ct)$
        else if $r \leq CP_3$ then
          $ct \leftarrow jumpToFriendTweet(ct)$
        else
          $ct \leftarrow jumpToHashtagTweet(ct)$
        end if
        $V_{ct} \leftarrow V_{ct} + 1$
      end while
    end for
  end for
  $\pi_t \leftarrow Normalize(V)$ {TweetRank as the normalized visit vector}
```

Observe that the running time of algorithm 1 is non-deterministic, since it depends on the structure of the graph itself and the α parameter. Assuming random access memory, and no dangling nodes in graph, the previous algorithm has an expected running time bounded by $O(|T| \times M \times \mathbb{E}[|w|])$, where $\mathbb{E}[|w|]$ is the expected length of the random walk. The expected length of the random walk is obtained from the following equation:

$$\mathbb{E}[|w|] = \sum_{k=1}^{\infty} P(|w| = k) \cdot k = \sum_{k=1}^{\infty} (1-\alpha)^{k-1} \cdot \alpha \cdot k = \frac{1}{\alpha} \tag{21}$$

Given that $|T| \times M$ must be $O(|T|^2)$ to ensure a good approximation of TweetRank, the expected running time of algorithm 1 is:

$$O(|T| \times M \times \mathbb{E}[|w|]) = O(|T|^2 \times \frac{1}{\alpha}) = O(|T|^2) \tag{22}$$

Note that equation 22 is an upper bound on the expected running time if the graph contains dangling nodes. On the other side, the assumption of random access memory might be a problem for a large index which does not fit in the main memory of a single machine. However, implementing TweetRank on a large-scale distributed system is not the approach of this work and we will maintain this assumption to keep the analysis simple.

4.3 Solr - Incomplete!

Solr 3.6 were used as the query interface for the TweetRank. It receives the Tweets from the Crawler by XML POST requests and indexes them. To improve the usability we also make use of the built-in tf-idf scoring that is offered by Solr/Lucene.

The TweetRank scores which are calculated at the ranker component which then prints the results as key value pairs in a plain text formatted file. This file is then used by Solr as an "External file field" which we combine with the other ranking attributes. The ranking attributes that we base our weighting on are the following:

- **Hashtag** - If a hashtag H is mentioned in the tweet T and found in the query Q, we give additional score to T for the query Q.
- **Username** - If the username of the author of tweet T is found in query Q, we give additional score to T for the query Q.

In addition to the the mentions above, we give an additional score to tweets that have been tweeted recently. The function we used for this additional date score is following:

$$\frac{a}{m * x + b} \tag{23}$$

where we have defined x as the age of the tweet in milliseconds, $m = 3.16E^{-11}$, $a = 0.08$ and $b = 0.05$.

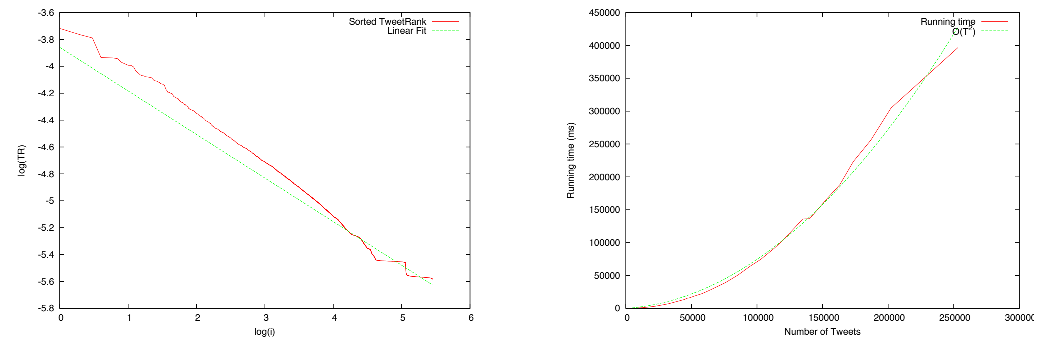
5 Experiments

Measuring the quality of the TweetRank score is difficult because there are no reference corpora to be tested on, the only thing that we did was to try it together with Solr and check that the results quality was good.

We also plotted the sorted list of TweetRank values in log-log scale and checked that it follows closely the power law, as it was stated by [1]. Figure 2(a) shows this fact.

Figure 2(b) shows the number of tweets versus the running time. We see that a quadratic curve fits very well with the experimental data, which confirms our analysis of the expected running time described in equation 22 (section 4.2). These results were obtained on a machine and parameters specified in table 1.

It's important to observe that our model has many parameters to be adjusted. Design a good set of experiments to choose the best values would require a lot of time and also lot of time to be tested. For estimating the optimal values for the probabilities used by our algorithm, the simplest solution would be trying different combinations of their values and try to find the one that produces better results. However, this would require a tremendous amount of time and,



(a) Sorted TweetRank, loglog scale (b) Number of tweets vs. running time

Figure 2: Experiment results

Processor	Intel Core i5, 2 cores at 2.4GHz, 64 bits
Memory	Main: 4GB DDR3, Cache L3: 3MB, Cache L2: 256KB
OS	Mac OS X 10.6.8 Snow Leopard
Compiler	javac 1.6.0_31, 64 bits, no optimization parameters
Others	8 threads, $\alpha = 0.2, \beta = 0.4, \gamma = 0.2, \delta = 0.04, \epsilon = 0.16, \zeta = 0.2, M = T/100$

Table 1: Specification of the machine used for experimentation

even more important, would require a labelled dataset to measure the precision and recall of the search engine.

An other approach, would be monitor those probabilities on real users. For example, installing a plug-in on the web browser that registers the activity of the user (when the user clicks on a retweet or reply, on a mentioned user, on the friends list, etc). To preserve anonymity, this would be a plug-in installed in volunteer's browsers and the data collected would not contain any personal information (it is not important for us which tweet is viewing the user, but how he or she got to it). Using this data, one could estimate those probabilities and use the estimated value to compute the TweetRank.

6 Conclusion

We adapted the definition of PageRank to Twitter context where direct references among tweets (retweets or replies) are not the usual way to surf on Twitter. We proposed to take into account also mentions, followers and hashtags to determine the importance of a tweet. We presented an extension of the previous model which considers the similarity between users in terms of common hashtags.

We have disscussed several during the previous sections the advantages and disadvantages of our model and the algorithm used to compute the TweetRank. However, the main disadvantage to our current implementation is that each time a new tweet is added, a new computation for the whole set of tweets is required. And this approach does not scale in a dynamic environment like Twitter where millions of tweets are published each second. Computing PageRank on a dynamic network is a well-studied problem. Our model could be adapted to the approaches described in [2, 3] easily, but that is more over the purpose of this work.

References

[1] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM J. Numer. Anal.*, 45(2):890–904, Feb. 2007.

[2] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *Proc. VLDB Endow.*, 4(3):173–184, Dec. 2010.

[3] P. Desikan, N. Pathak, J. Srivastava, and V. Kumar. Incremental page rank computation on evolving graphs. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 1094–1095, New York, NY, USA, 2005. ACM.

[4] Y. Duan, L. Jiang, T. Qin, M. Zhou, and H.-Y. Shum. An empirical study on learning to rank of tweets. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 295–303, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[5] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM.

[6] R. Nagmoti, A. Teredesai, and M. D. Cock. Ranking approaches for microblog search. In *Web Intelligence*, pages 153–157, 2010.

[7] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[8] S. Ravikumar, R. Balakrishnan, and S. Kambhampati. Ranking tweets considering trust and relevance. *CoRR*, abs/1204.0156, 2012.

[9] M. J. Welch, U. Schonfeld, D. He, and J. Cho. Topical semantics of twitter links. In *WSDM*, pages 327–336, 2011.