**A REPORT ON TWITTER DATA ANALYTICS: RANKING TWEETS**

**By**

**Sukanya Patra Nikita Dwivedi**

**CSE-2nd year CSE-2nd year**

**IIIT Bhubaneswar VNIT Nagpur**

**Under the Guidance of**

**Prof. R.B. Keskar**

**Department of Computer Science and Engineering, Visvesvaraya National Institute of Technology, Nagpur-440010**

**2015-16**

**Table of Contents**

Acknowledgement

Abstract

1. Introduction.....................................................................1

2. Background.....................................................................5

3. Methodology...................................................................6

4. Conclusion and Result....................................................13

5. Future Work....................................................................14

**Acknowledgement**

We avail this opportunity to extend our greatest gratitude to our project guide Dr. R.B. Keskar, Assistant Professor, Department of Computer Science and Engineering, VNIT Nagpur for his invaluable guidance and morale support which enabled us to complete this project. His ideas and suggestions helped us in doing a lot of research.

We wish to pay our sincere thanks to Dr. Manish P. Kurhekar, Assistant Professor, Department of Computer Science and Engineering, VNIT Nagpur for his support and invaluable advice throughout this project.

**Abstract**

Microblogging is one of the important sources for disseminating real-time unbiased information quickly and widely in the current age. Twitter, as a microblogging tool has gained so much popularity in the recent years that some people regard it as the news outlet of the 21st century. The 310 million-strong active user community communicates through 140-character messages called “Tweets”, and this character limit sparks creativity as users try to pack substantial meaning in a short space. At the same time, users also post tweets which can be effectively regarded as unimportant, or spam, for the other users. As a consequence, real time twitter data is a mixture of both relevant and irrelevant information.

The biggest challenge is to filter out ‘important’ or ‘relevant’ tweets in the flood of tweets. In this report, we propose a ranking algorithm to rank tweets on trending topics made in a time span of 24 hours, by considering the retweets of each tweet over a hierarchy of three levels of followers of the original author who retweeted the tweet.

**Introduction**

Twitter is one of the most popular microblogging services which enables users to exchange short 140-character updates, comments or thoughts called "tweets". In 2013, it was one of the ten [most-visited websites](https://en.wikipedia.org/wiki/List_of_most_popular_websites) and has been described as "the [SMS](https://en.wikipedia.org/wiki/Short_Message_Service) of the Internet". As of March 2016, Twitter has more than [310 million monthly active users](https://en.wikipedia.org/wiki/List_of_virtual_communities_with_more_than_100_million_active_users)

Unlike other social networking sites, twitter users follow others or are followed, which requires no reciprocation. It makes connections in real-time based on dynamic interests and topics, rather than a static social graph. A user can follow any other user, and the user being followed need not follow back. So, it is a one-way following mechanism which is why it is very powerful and provides a richness that can't be gotten from two-way friend requests. Depending on the security and privacy settings, the tweets can be publicly read by everyone or limited to those approved followers only.

Twitter’s strength is real-time. No other social platform comes even close. Social media data is often difficult to obtain, with most social media sites restricting access to their data. Twitter's policies lie opposite to this. The twitter streaming API allows anyone to retrieve atmost 40% of all live-tweets by providing some parameters.

The world's best content is inside Twitter. Whether it is news, entertainment, sports, human interest, celebrity gossip, technology, art, politics, product launch or advertisement, twitter has it all in real time. People use twitter in many ways, some as a newsfeed by following prominent people or networks, some as a pseudo-chatroom by limiting their followers and whom they follow to close friends and family, and some as a microblog for updating people about the work they are doing and their personal lives. Moreover, it is open for public consumption, has well-documented API, rich developer tooling and appeals to users from every walk of life.

**Ranking tweets is important** Twitter folks are hyperactively engaged. Hence, the ever-flowing real-time information on twitter is noisy, ambiguous and large. Also, a lot of content generated by tweets is irrelevant. When an event of sizeable magnitude and impact occurs, thousands of tweets are posted per hour. Due to the large amount of content generated on twitter, it is hard to identify the tweets with credible information manually.

The purpose of this report is to rank trending tweets based on retweets by followers at three levels of hierarchy of the tweeter by assigning an influence factor to each level of followers.

**Trends on twitter** One of the main features of twitter is a list of top 10 terms called trending topics that users see on the sidebar of their twitter homepage. A trend on Twitter is a hashtag-driven topic that is immediately popular at a particular time. Trending topics on Twitter show readers what the most popular conversation topics on the microblogging site are right now. According to twitter, trends are *topics that are popular now, rather than topics that have been popular for a while or on a daily basis, to help you discover the hottest emerging topics of discussion on Twitter that matter most to you.* Trending topics are determined by an algorithm that monitors hot subjects based on who one follows and where one is located. However, many national and international topics are seen by everyone regardless. It often reflects what's hot in the news, from the death of Michael Jackson to Britain exiting from the European Union. Significant world events, international sports results, and news about popular celebrities are among the items that commonly "trend" on the Twitter site.

**Retweets** Common practice of responding to a tweet is by retweeting. A retweet is a repost of a tweet posted by another user, i.e., a retweet means the user considers that the message might be of interest to others. When a user retweets, the new tweet copies the original one in it. Furthermore, the retweet attaches an ***RT*** and the ***@username*** of the user who posted the original tweet ***Text of the original tweet*,**  a retweet on that tweet would look this way

RT @username: Text of original tweet.

Moreover, retweets can be further retweeted by others, what creates a retweet of level 2, e.g.,

RT @username2: Text of original tweet.

Similarly, retweets can go deeper into 3rd level, 4th level and so on. Hence, the retweet mechanism allows the users to spread information of their choice beyond the reach of the original user's followers. It is considered to be an essential mechanism for information diffusion on twitter. Retweeting is most useful when the retweeter has a large network and occupies structural holes, or gaps in network connectivity between different communities.

**Twitter APIs**  Twitter provides two public APIs to access its data namely, REST API and Streaming API. They use JSON data format for responses.To access the twitter API, a Twitter account was created and the credentials(i.e. API key, API secret, Access token and Access token secret) were obtained from the twitter developer site. There are a number of libraries written in different languages which are used to access the twitter APIs.

**Streaming API:** The Twitter Streaming API provides high-throughput near-realtime access to a fraction of Twitter's global stream of data, i.e., it can help access to tweets as they are posted . Twitter provides several streaming endpoints. Public streams constitute streams of public data flowing in twitter, suitable for focussing on specific users or topics, user streams give access to single user stream and site streams give access to multi-user version of single user stream. A persistent HTTP connection is required to connect to the streaming API.

**Rest API:** The Twitter REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and information about a user. Unlike streaming API, the Rest API is rate-limited. It can give 100% access to ongoing streams of tweets based on the search terms. Due to the constraint provided by the API, it is likely to miss out some crucial tweets containing the search terms. It gives a historical tweets posted in a week's duration.

**Problem Statement** Create a system which takes in a continuous stream of raw tweets, filters out the retweets of the tweets tweeted in the last 24 hours, stores the relevant information about each tweet in a database; and ranks them on the basis of the number of retweets, the *influence factor of the retweeter*, and the retweeter level.

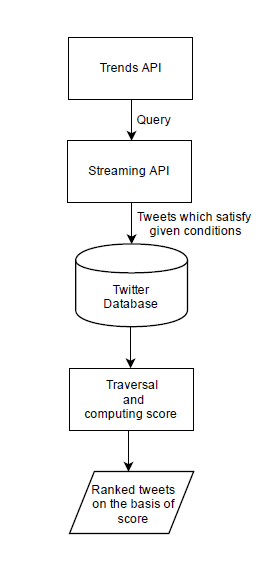
The main aspects of this problem are:

1. Collecting tweets and filtering them out to identify retweets.
2. Processing these tweets by grouping the retweets of the same tweet and identifying the *retweeter level*.
3. Storing this information in a graph database, and calculating a unique score for each tweet.

Influence factor of a retweeter: The number of followers of the retweeter.

Retweeter level: Indicates whether the retweeter is a direct follower or not of the original author of the tweet.

**Project Activity Diagram**



**Background**

With Twitter being such a hot trend right now, research firms have been anxious to study how people are using the social platform, and analyze trends. Some research firms such as Pear Analytics, have found out that 40% of the tweets in twitter are pointless babble, with another 37.55% being conversational. Due the large amount of tweets being posted daily, ranking strategies have become extremely important for retrieving information quickly.

* Twitter shows top Tweets that many other Twitter users have engaged with and thought were useful.
* Srijith et al. proposed a method of ranking of tweets considering trustworthiness and content based popularity. They modelled a microblog ecosystem at a three layer graph consisting of users, tweets and web pages.

**Tweet search**

* Websites such as Twazzup rank real time tweets by posting time.
* Chirps ranks tweets by retweet count and posting time as well.
* Twitority and tweetfind rank tweets by author authority.
* Bing and CrowdEye rank tweets by posting time and content relevance.

**Tweet recommendation**

* Hong et al. (2011) recommend tweets based on popularity of messages based on number of future retweets.
* Chen et al. (2010) investigate how to select interesting URLs linked from Twitter and recommend the top ranked ones to users. Their recommender takes three dimensions into account: the source, the content topic, and social voting.
* Duan et al. in their paper proposed a new ranking strategy which uses the content relevance of a tweet, the account authority and tweet-specific features such as whether a URL link is included in the tweet.
* Yan et al. proposed a graph-theoretic model for tweet recommendation which ranks tweets and their authors simultaneously using several networks.

Recently, many approaches for blog search and forum search have been developed, which include *learning to rank* methods and link-based method. Our work devises a method to rank tweets and filter out the most important ones. It has turned out that the number of followers of a tweeter alone cannot measure the rank of a tweet. A user can accumulate hundreds of thousands of followers in a day or so. At the same time, it is not difficult for spammers to create large quantities of retweets. So, we have considered followers and retweeters at different levels of hierarchy to filter out valuable information from a sea of tweets.

**Methodology**

**Construction of the Dataset**

**The Streaming API:** This API allows authenticated users to collect tweets in real time, based on various parameters like search terms, language(s), locations etc., without imposing any rate limit restrictions(unlike the REST API). To obtain relevant tweets, we used the Trends API, with WOEID=1, which returns the top 10 trending topics worldwide. The Trends API was repeatedly polled after collecting every 5000 tweets(approximately after every three hours), and the search query was updated to continue the flow of relevant tweets.

Approximately, around one hundred thousand tweets were collected over a time span of two days for further processing.

**Tweepy:** Tweepy is an open-source, well-documented Python library which enables Python to communicate with Twitter and use its API. It supports accessing Twitter using OAuth, with proper authentication credentials.

In Tweepy, an instance of **tweepy.Stream** establishes a streaming session and routes messages to the **StreamListener** instance, from where they were filtered and stored in the database for further processing.

**Trends API:** The Trends API is a Twitter REST API which provides the top ten trending topics of a particular country, and for WOEID=1, we get the top ten trending topics worldwide. This API is rate limited to 15 requests in a 15 minute time interval. Also, it is updated after every 5 minutes, so two requests with a time gap of less than 5 minutes will return the same result.

**Filtering the tweets:** The Streaming API response is a JSON object for each tweet, containing information about the tweeter, if the tweet is a retweet/reply, the content (including hash tags, mentions, URLs etc.) and the location at which the tweet was tweeted.

In order to track retweets, we had to filter out all those tweets from the Streaming API which

1. Had the ‘in\_reply\_to\_user\_id’ field set as ‘True’, this indicates that this tweet is a reply, and is of no use to us.
2. Did not have the ‘retweeted\_status’ field, this means that this tweet is not a retweet but an original tweet, and was thus discarded as we are only interested in collecting retweets.
3. The ‘created\_at’ fields for both the original tweet(embedded in the JSON object) and the retweet were checked, and if the time difference was greater than 24 hours, the tweet was discarded. This was done to make sure that the retweets which are collected are of recent tweets only.

From amongst the filtered tweets, the following fields were extracted and stored in a csv file –

**Table 1: Fields in the csv file**

|  |  |
| --- | --- |
| Original Tweet ID | ID (integer) of the original tweet |
| Tweet Text | The content of the tweet |
| User ID | The unique id of the original author of the tweet |
| User Screen Name | The ‘user handle’ of the original author |
| User’s followers | Number of followers of the user |
| Retweeter ID | The unique id of the retweeter of the tweet(uniquely identified by the tweet id/text) |
| Retweeter Screen Name | The retweeter’s user handle |
| Retweeter’s followers | Number of followers of the retweeter |

**Table 2: Descriptive statistics of the Twitter dataset(after filtering)**

|  |  |
| --- | --- |
| Start Date | 13/07/2016 |
| End Date | 15/07/2016 |
| Total number of tweets | 6000 |
| Database(.csv) file size | 1.4 MB |

**Retweeter level:** The above csv file was sorted on the basis of ‘Original Tweet ID’, so as to group the retweets of the same tweet together. Now, the next challenge was to compute the retweeter level:

**Level 1:** Retweeter is a direct follower of the original author of the tweet

**Level 2:** Retweeter is a follower of any of the level 1 retweeter

**Level 3:** Retweeter is a follower of a level 2 retweeter

**Level 0:** Retweeter does not fall into any of the first three levels. It was observed that the number of retweeters decreases rapidly as the level increases. So, any retweeter belonging to this category is a fourth level retweeter (very unlikely), or has retweeted this particular tweet because he/she follows the hashtags embedded in the original tweet.

Identifying the retweeter level: The Twitter REST API method ‘GET friendships/show’ returns detailed information about the relationship between any two arbitrary users.

The retweeters of the same tweet were grouped together, and their ids and the id of the original user were passed as parameters to the above method; and hence the retweeters were categorised as first/second/third/zero level followers.

**GET friendships/show:** This REST API is rate limited to 180 requests in a 15 minute time window. After exceeding 180 requests, we had to wait for 15 minutes before sending in a new request. Fortunately, Tweepy has an in-built exception handler to handle the ‘Rate limit exceeded’ exception, and on catching that exception, the code paused for 15 minutes before making a new request.

**Pseudocode:**

open database for input

open new\_database for output

for each row in database, do:

for each tweet, do:

obtain tweet text, tweeter

store all retweeter IDs in retweeters

for each ID in retweeters, do:

call api.show\_friendship(tweeter ID, retweeter ID)

if retweeter follows tweeter

store retweeter ID in first\_level\_followers

append row,1 to new\_database

remove retweeter from retweeters

endif

endfor

for each ID in first\_level\_followers, do:

for each id in retweeters, do:

call api.show\_friendship(tweeter ID, retweeter ID)

if retweeter follows tweeter

store retweeter ID in second\_level\_followers

append row,2,ID to new\_database

remove retweeter from retweeters

endif

endfor

endfor

for each ID in second\_level\_followers, do:

for each id in retweeters, do:

call api.show\_friendship(tweeter ID, retweeter ID)

if retweeter follows tweeter

store retweeter ID in third\_level\_followers

append row,3,ID to new\_database

remove retweeter from retweeters

endif

endfor

endfor

for each ID in retweeter, do:

append row,0 to new\_database

endfor

endfor

endfor

close database

close new\_database

This processed data was then passed on to the Neo4j database for applying the ranking algorithm.

**Graph Database:** A graph database stores data in the form of graph structures, with nodes, edges and properties to represent data. Each node in the graph represents an entity, and each edge defines a connection or relationship between any two nodes. A node can be uniquely identified by its properties (usually stored in the form of key/value pairs) and a set of incoming and outgoing edges. Edges can be uniquely identified by their properties, and their starting and ending nodes.

**Reasons for using Graph Database**: Data mined from a social network (here, Twitter) is intuitively graphical, as there are a lot of interconnections. Using a graph database proves fruitful while storing and/or retrieving data, due to the usage of various graph traversal algorithms. This would have proved to be very difficult to accomplish in relational databases, as representing and processing many relationships would be expensive.

**Neo4j:** Neo4j is the world's leading open source [graph database](https://en.wikipedia.org/wiki/Graph_database) management system, developed by Neo Technology, Inc. It is an [ACID](https://en.wikipedia.org/wiki/ACID)(Atomicity, Consistency, Isolation, Durability) -compliant transactional database with native graph storage and processing. It is a NoSQL database and is written in Java. Neo4j employs a REST service interface and provides an admin console hosted in a web browser. A single server instance can handle a graph of billions of nodes and relationships. It is simple and intuitive to use. It can perform millions of traversal steps on richly-connected data within seconds.

**Cypher** is the query language used by Neo4j. It employs a neat way of expressing nodes and their relationships. It is a declarative pattern-matching language whose syntax is in human readable format.

Apart from having all the graph database properties mentioned above, edges and nodes can be ‘labelled’, which narrows down the searches.

There are many Neo4j clients available, for several programming languages including Java and Python. For Python, the Neo4j community has contributed a range of driver options, which include Py2neo, Neo4j Rest Client, Bulbflow etc. In our project, we have chosen to work with Py2neo as it is intuitive and easy to use, and readily supports embedding CYPHER Query Language in the Python code, which makes querying easier.

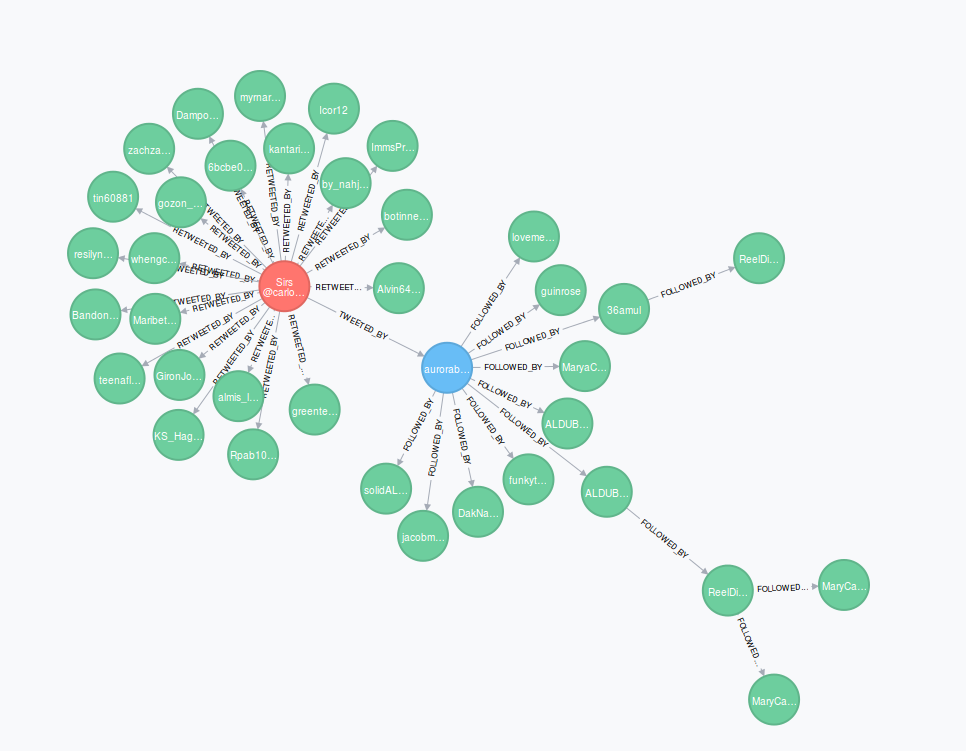
**Py2neo:** Py2neo is a client library and toolkit for working with Neo4j from Python. The core library has no external dependencies, has a cleaner API and integrates Cypher in a much cleaner and more intuitive way.

**Reason for storing data in csv file instead of directly migrating to neo4j:** If we had directly migrated the data to Neo4j, the difficulty of categorising retweeters into various levels would have increased multifold, and we would have had to change the database too often.

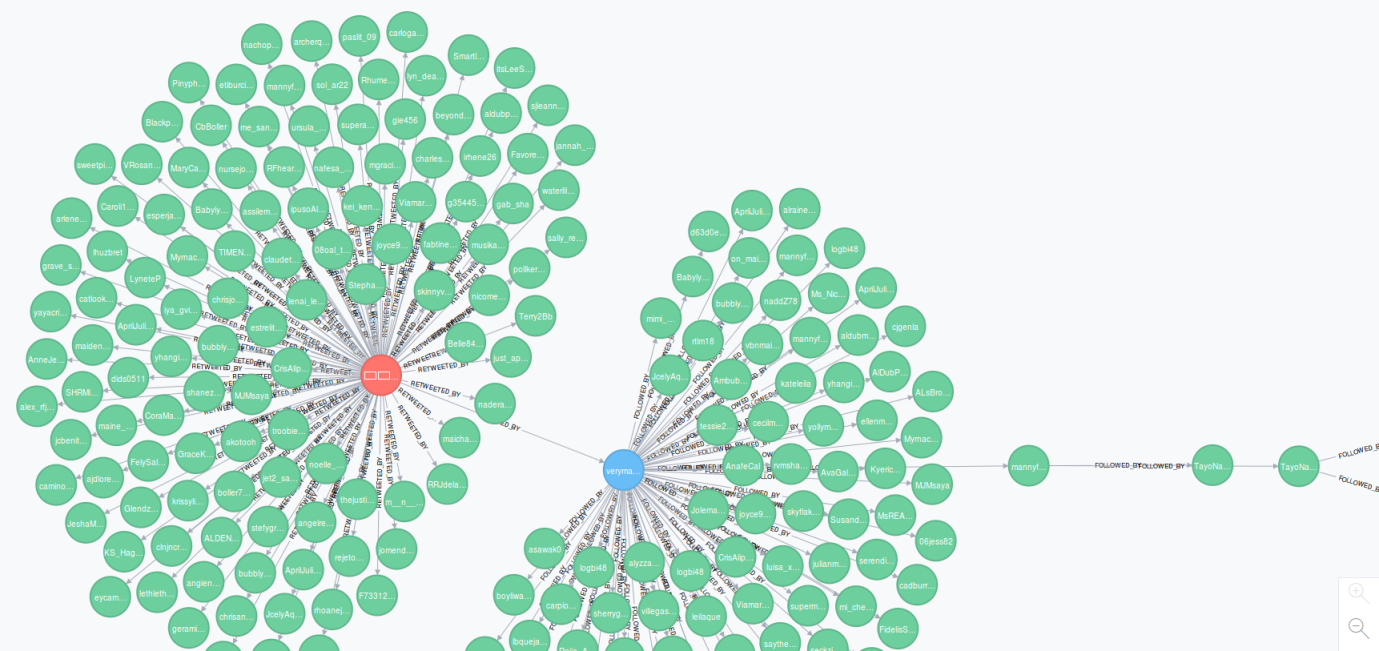
Therefore, to avoid making multiple changes to the graph, we have restricted the processing of the data to the csv file(s).

**The Neo4j tweet database**: The format of the database is as follows

|  |  |  |
| --- | --- | --- |
| Node/Relationship | Label | Property keys |
| Node | Tweet | Id, text |
| Node | Original\_User | Name, id, tweet id, followers |
| Node | Retweeter | Name, id, tweet id, followers |
| Relationship | TWEETED\_BY | - |
| Relationship | FOLLOWED\_BY | Level |
| Relationship | RETWEETED\_BY | - |



**Figure 1: Screenshot displaying a tweet(the red node), its author(the blue node) and its retweeters(the green nodes) at various levels**



**Figure 2: Screenshot displaying another tweet(the red node), its author(the blue node) and its retweeters(the green nodes) at various levels. This tweet seems to be more important than the one in figure 1, as it is surrounded by more green nodes. This prediction can be confirmed using our ranking algorithm :)**

**The Ranking Algorithm** It was observed that the probability of a tweet being retweeted by a first level or a zero level follower is very high, and decreases very rapidly as the level goes up. In other words, the tweet spreads more in the first and zeroth level, and accordingly, the tweet graphs were found to be much clustered in these two levels, and became scarce at the second and third levels.

These observations led us to assign the following influence factors to each level, so as to give equal weightage to each level –

Influence factor for level1 (IF1) = 0.20

Influence factor for level2 (IF2) = 0.30

Influence factor for level3 (IF3) = 0.40

Influence factor for level0 (IF0) = 0.10

**Graph Traversal**

**Breadth first traversal:** The Neo4j graph database was queried and all the tweets and their original authors were obtained. From this Original\_User node, we traversed all the first level followers who retweeted this tweet, summed up the *influence factors of each of these retweeters* (the influence factor of a retweeter is the number of his first level followers), and multiplied this sum with *the influence factor of level 1*. Now, each of these first level retweeters were taken, and their followers and retweeters were traversed in a similar fashion. The same steps were followed to obtain the sum of the influence factors of the second and third level followers.

To obtain the sum of IFs for the 0th level, the tweet node was taken and all the nodes with the relationship ‘RETWEETED\_BY’ with the tweet node were traversed, and their Ifs were summed up.

Influence factor of a retweeter (IFR) = Number of first level followers of the retweeter

Let Ri denote the sum of IFs of retweeters in level i

Score of a tweet = ∑(Ri \* IFi)

Where IFi is the influence factor of level i

The tweets were then sorted on the basis of their score in the reverse order, and ranks were assigned appropriately.

**Pseudocode:**

set influence\_factor1 to 0.4

set influence\_factor2 to 0.3

set influence\_factor3 to 0.2

set influence\_factor0 to 0.1

for each node of the graph, do:

obtain tweet text

obtain original tweeter

obtain and store all the follower IDs of the original tweeter

initialise level1\_followers to 0

initialise level2\_followers to 0

initialise level3\_followers to 0

initialise level0\_followers to 0

for each follower ID of original tweeter, do://1st level

obtain and store follower ids of follower ID in id1

add number of followers of ID to level1\_followers

endfor

for each ID in id1, do://2nd level

obtain and store follower ids of ID in id2

add number of followers of ID to level2\_followers

endfor

for each ID in id2, do://3rd level

add number of followers of ID to level3\_followers

endfor

obtain and store remaining retweeters

for each remaining retweeter, do:

add number of followers of retweeter to level0\_followers

endfor

set score to level1\_followers\*influence\_factor1 + level2\_followers\*influence\_factor2 + level3\_followers\*influence\_factor3 + level0\_followers\*influence\_factor0

store score in scores

print score

endfor

sort scores in descending order to obtain rank of each tweet

**Conclusion and Result**

In this project, an algorithm to find important tweets was proposed and implemented.

The following observations were made:

* Some real-time trending tweets which we could see on twitter weren’t present in our database. This was because of the streaming API limitation as it returns atmost 40% of the real-time twitter data.
* The time taken to process the tweets took more time as compared to obtaining tweets from twitter due to the difference between streaming and REST API; the streaming API provides unrestricted access unlike the REST API which imposes rate limits.

The results obtained for the database containing 6000 tweets can be tabulated as follows:

|  |  |  |
| --- | --- | --- |
| Rank | Tweet Text | User screen name |
| 1 | Bernie Sanders supporters are planning a "fart-in" at the Democratic NationalConvention | IndyUSA |
| 2 | [#TheFlash](https://twitter.com/hashtag/TheFlash?src=hash) star Keiynan Lonsdale on set today shooting his 1st public scenes as Kid Flash | Canadagraphs |
| 3 | We Applaud [@jasoninthehouse](https://twitter.com/jasoninthehouse) for grilling [@LorettaLynch](https://twitter.com/LorettaLynch) at the [#LynchHearing](https://twitter.com/hashtag/LynchHearing?src=hash) [#RETWEET](https://twitter.com/hashtag/RETWEET?src=hash) Join [http://Bikers4Trump.com](https://t.co/AGuHKH0S7K) | Bikers4Trump |
| 4 | Supporting Gary Sinise Foundation [@GarySinise](https://twitter.com/GarySinise) on [#PrimeDay](https://twitter.com/hashtag/PrimeDay?src=hash) - Serving Heroes, Honor & Need [http://smile.amazon.com](https://t.co/ZszspzcXBI) | Celtic\_Norse |
| 5 | Today is another history in the making. We are proud of you ALDEN RICHARDS and MAINE MENDOZA! [#ALDUBImagineYouAndMe](https://twitter.com/hashtag/ALDUBImagineYouAndMe?src=hash) | ShowbizBanter |

**Future Work**

In this project we devised an algorithm to rank tweets by assigning a score to them based on retweeter hierarchy.

We propose the following enhancements and features that can be added to this project work:

**Expanding the idea of infuence factor of a retweeter**

Instead of only considering the first level followers of a retweeter, we can crawl the followers upto a particular depth to get a more comprehensive idea of the actual influence of the retweeter. This will also help us filter out spam accounts.

**Including more factors in the ranking algorithm**

Other factors like the URLs embedded in the tweets, @mentions can be assigned a certain priority and included in the final score of the tweet.

**Combining multiple datasets**

We can collect data in more than one file for several days and combine them in Neo4j. So, tweets trending for several consecutive days can be identified and ranked better.

**Who to follow recommendations**

We can recommend users who they can follow based on their interests. Interests can be identified from the kind of tweets they retweet.