

# Twitter Analysis

Brandon Cooper  
William Farmer  
Kevin Gomez

December 13, 2013

# Contents

1	Introduction . . . . .	1
2	Materials and Methods . . . . .	1
2.1	Analysis Overview . . . . .	2
2.2	Data Acquisition and Formatting . . . . .	2
2.3	Data Cleaning . . . . .	3
2.4	Data Entry . . . . .	4
2.4.1	Create Matrix . . . . .	7
2.4.2	Replace . . . . .	8
2.4.3	Followers . . . . .	8
2.4.4	Normalize . . . . .	9
2.5	Analysis . . . . .	9
2.6	Analysis . . . . .	9
3	Conclusion . . . . .	10
3.1	Implications . . . . .	10
3.2	Discussion . . . . .	11
1	Workflow . . . . .	13
2	Eigenvector . . . . .	13
3	User Weight Dataframe . . . . .	13
4	Attachments . . . . .	16
5	Authors . . . . .	16

# List of Figures

1	Simplified Links of our Established Network . . . . .	11
2	All Users of our Established Network . . . . .	12
3	Workflow Diagram . . . . .	13

# List of Equations

1	User Influence . . . . .	2
2	Abbreviated User-Weight Dataframe . . . . .	6
3	Initial Arbitrary Vector . . . . .	9
4	Resulting Eigenvector . . . . .	10

## **Abstract**

This study shows how a ranking system can be applied to Twitter and give a ranking for users based on their influence. Through application of matrix methods, users on Twitter were ranked according to retweets, followers, and favorites. The ranking system was based on the application of the Perron-Frobenius theorem as well as an iterative method. For this particular study, a smaller data set was used due to the complexity of the ranking system as the matrices increase in size. A larger data set would also be hard to model because some users are hardly connected to twitter and don't have much information. Their data might skew the system unless a significantly large data set is analyzed. The first step was to verify the matrix was irreducible. Once the matrix was found to be irreducible, the Perron-Frobenius theorem allowed for the matrix to be manipulated with an iterative method. Traditional power iteration was used for approximation of the dominant eigenvector, and was able to determine an accurate ranking for our users.

# 1 Introduction

Twitter is a free, online social media website with over 550 million users worldwide. It allows a user to create brief messages, or “tweets” (not exceeding 140 characters in length), and share them with their followers. The popularity of this website has greatly altered the way in which information is transferred across the world, as well as the speed at which this is done. With that being said, any one user has the opportunity to become rather influential based on how many other users on Twitter interact with his/her tweets. Because of its clear relevance in the modern world, celebrities, bloggers, and news media sites have taken to sharing information with Twitter. Some especially famous celebrities have several millions of followers, and are obviously incredibly influential in the world of Twitter.

We then seek to rank Twitter users based on how influential they are to other users. In order to do so, one must account for the three most common forms of interaction on the website: followers, favorites, and retweets.

To understand these three interactions, take for example user A and user B. If user A “follows” user B on Twitter, anything that user A tweets will appear on user B’s news feed. This is by far the most common form of interaction. If user A “favorites” a tweet from user B, user A expresses his/her interest in user B’s tweet because of its humor, importance, relevance, etc. Users can see one another’s favorited tweets, so if user B is commonly favorited there is a good chance others will see his/her tweets. Finally, the least common interaction, a “retweet” occurs when user B chooses to allow a tweet from user A to show on his/her page. In doing so, all of the followers of user B see the tweet that user A posted. This broadens the scope of their influence, as many more followers are able to see the tweet without having to search for it. This can quickly allow a tweet from someone with a small amount of followers to reach a large amount of users.

Due to the immense amount of users in Twitter, an analysis of the entire population would be very time consuming and difficult. We therefore limit the scope of our analysis to a random sample of tweets in the Boulder area. However, since these tweets could be retweets of users outside of Boulder, or even the country, our data set will almost certainly include a broad spectrum of users.

# 2 Materials and Methods

Because there was a large amount of data that needed to be analyzed, there were numerous tools that were necessary for an effective study. The tools used include Twitter’s development API as well as programming methods for analysis of the data. The first step to figuring out the ranking was to obtain the data from twitter that needed to be analyzed. Twitter’s development API allowed us to take the necessary data in order to rank the users, which included the tweets of different users as well as many crucial details about the user’s account. For the ranking method used, the only tweets that were needed were the, relatively rare, retweets to allow for a more accurate ranking. The idea behind it was that it took out any data that could be considered an outlier, as well as limit our search to users that were actually important enough to be retweeted.

Once the data from Twitter was acquired through the API, the tweets needed to be processed in a way that the ranking method could then be applied. Doing the work by hand would have been nearly impossible due to the amount of data and the complexity of the ranking methods.<sup>1</sup> In order to analyze the data effectively we needed to use more sophisticated data analytical tools. Python, R and SQL were the three main languages used for the analysis, with Python being predominant due to its all-purpose intent as well as its relatively simple syntax and library support. It also handles Twitter nicely using `requests` and `requests_oauthlib` to manage their OAuth authentication system. R was used primarily because of its support of larger filebacked matrices, but also because it can handle large data sets and has built in tools for visualization. We chose to store the raw data in a SQLite database, which is very easy to use with both Python as well as R, because of its inherent connectability between the different languages as well as its relatively lightweight framework. With those three tools, the ranking method could then be applied to the Twitter data.

In terms of use, Python was used solely to obtain the data from Twitter’s API and create the visualizations. Once the data was stored, a primary ranking algorithm was established to determine edge weights in our network. There were three criteria in our ranking algorithm which were the number of followers, the favorites and the retweets.

---

<sup>1</sup>Our final network had 2715 individual users.

$$Influence = x \cdot (followers) + y \cdot (favorites) + z \cdot (retweets) \quad (1)$$

We decided that because retweets were so rare, they would carry the most weight for our influence score. Followers and favorites were fairly common on Twitter so they didn't carry as much importance. In this study, for the above equation,  $x = 1, y = 1, z = 8$ .<sup>2</sup> Once our parameters were obtained, the ranking algorithm we created was applied using R. The application of our ranking algorithm gave us an adjacency matrix<sup>3</sup>, which was then normalized to give us our stochastic matrix. One theorem that was necessary to verify we could proceed was the Perron-Frobenius theorem which states that a real square matrix with positive entries has a unique largest real eigenvalue and that the corresponding eigenvector has strictly positive components. The Perron-Frobenius theorem also required our matrix be irreducible, which allowed us to proceed with finding the ranks. The last step for obtaining our final ranks was to apply the power iteration method, and thusly obtain an approximation of our dominant eigenvector.

## 2.1 Analysis Overview

There were four main steps involved in analysis.<sup>4</sup>

1. Data Acquisition and Formatting (Python)
2. Data Cleaning (Python)
3. Data Entry (R)
4. Analysis (R)

## 2.2 Data Acquisition and Formatting

The first step of this process was to import the data from Twitter's streaming API.

```

21 ./code/twitterImport.py
22 # Limit to English around Boulder, CO
23 data = {'language': 'en', 'geocode': '40,105,50mi'}
24 # Stream URL
25 response = requests.get('https://stream.twitter.com/1.1/statuses/sample.json',
params=data, auth=auth, stream=True)

```

Twitter provides a handy url for streaming a random subset of all Twitter traffic. In order to avoid overfitting, or selection bias we used this random subset for all data analysis. We limited our search to English posts from Boulder, CO in order to hopefully acquire a densely connected network of related posts.

The next step is to either create the SQLITE database or open a previously existing one. The tables in the database are created with the following python code, and look like Table 1.

<sup>2</sup>Note, it would be trivial to adjust this influence equation and determine a different ranking based on a different numerical meaning of "influence"

<sup>3</sup>It is important to note at this time that the edges were established from (most importantly) retweets and follower links. Every retweet was given an edge weight of  $8 \cdot retweets + nodeweight$ , and every follower was given a weight of  $follower * nodeweight$ . Follower edges were only added if the user that was being followed existed within the established dataset, and retweet edges, by nature had to exist within the dataset.

<sup>4</sup>For a more complete and helpful diagram, please see Appendix 1.

```

29 cursor.execute("""CREATE TABLE users(
30     id TEXT PRIMARY KEY NOT NULL,
31     name TEXT NOT NULL,
32     followers TEXT,
33     favourites_count INT,
34     followers_count INT,
35     friends_count INT)""")
36 cursor.execute("""CREATE TABLE posts(
37     id TEXT PRIMARY KEY NOT NULL,
38     time TEXT,
39     entities TEXT,
40     user TEXT)""")
41 cursor.execute("""CREATE TABLE retweets(
42     id TEXT,
43     end TEXT,
44     count INT)""")

```

users	posts	retweets
id (text)	id (text)	id (text)
name (text)	time (text)	end (text)
followers (text)	entities (text)	count (int)
favourites_count (int)	user (text)	
followers_count (int)		
friends_count (int)		

Table 1: Database Tables

Once our database has been established, we move on to the actual data import process. The code block below shows the steps in the order they are completed. For a more detailed reference, please see the attached source code.

```

61 posts = take(limit, response) # Get posts from stream
62 add_post_users(posts, cursor, connection) # Add posts and users
63 add_mentioned_users(posts, cursor, connection) # Add all mentioned users
64 update_retweet_count(posts, cursor, connection) # Updates Retweets
65 update_user_info(connection, cursor) # Update mentioned profiles if missing

```

After this has been completed we have a full database filled with many unique users and posts.

## 2.3 Data Cleaning

After our initial data has been imported, we need to clean the data. In order for our analysis method to work, we need to acquire an irreducible<sup>5</sup> adjacency matrix.<sup>6</sup> The first step is to establish a new database entirely using the name of the previous database. The structure of this new database is identical to the old, save it is missing the posts table.

After the database is established, we scan our old database and do three things:

1. Find the *most important* user. This is determined by who has the most edges (retweets) connecting them to the other users.
2. Find all users that have an edge between them and the most important user.

<sup>5</sup>Irreducible in this case meaning that all nodes are connected to each other, and the matrix cannot be split into two or more adjacency matrices

<sup>6</sup>Where an adjacency matrix is the matrix representation of our network of users connected by retweets

3. Recursively apply step 2 to obtain the complete network.

Step 1 is accomplished by aggregating the retweet table into absolute scores counting how many times a user was retweeted. The user with the most retweets is defined as our “most important user” and only connected users are added to the clean database.

```

108 ./code/pyRank.py
108 for retweet in cursor.execute("SELECT * FROM retweets"): # Grab all retweets
109     dst = retweet[1] # and create dict
110     score = retweet[2] # of all user scores
111     most_important = dst
112     try:
113         user_scores[dst] += score
114     except KeyError:
115         user_scores[dst] = score
116 for user in user_scores: # Remove any that
117     try: # have less than 1
118         if user_scores[user] > user_scores[most_important]:
119             most_important = user
120     except KeyError:
121         pass

```

All users connected to our important user are then added to the new database by looping through user addition until the network no longer changes size from one iteration to the next.

```

123 ./code/pyRank.py
123 cleaned_scores[most_important] = user_scores[most_important]
124 retweets = [retweet for retweet in cursor.execute("SELECT * FROM retweets").fetchall()]
125 pastsize = 0
126 current_size = len(cleaned_scores)
127 while pastsize != current_size:
128     pastsize = len(cleaned_scores)
129     counter = 0
130     for retweet in retweets: # Grab all add to dict all users that
131         src = retweet[0] # Add to dict
132         dst = retweet[1]
133         score = retweet[2]
134         try:
135             cleaned_scores[dst]
136             cleaned_scores[src] = score
137             retweets.pop(counter)
138             edges.append((src, dst))
139             counter -= 1
140         except KeyError:
141             pass
142         counter += 1
143     current_size = len(cleaned_scores)

```

This then gives us a list of all users connected to the most important user, as well as the most important user itself. We then iterate through the old database and copy all relevant information from the old database to the new one if and only if the user exists in our list of connected/important users. At this point the data has been cleaned and all further analysis will be performed on the clean database.

## 2.4 Data Entry

Now that we have a reliable data source, we can analyze the data. This is accomplished using the R language. R is very good at handling large quantities of data, and as a result is ideal for this situation. The R code



for the analysis is contained in `./code/ranking.R`, and can be run separately from the python program, however this is not recommended.

The first step to importing the data from our previously established clean SQLITE database is to create the user dataframe.<sup>7</sup> As part of this operation, we need to establish a list of users by querying the user table from our clean database.

```

16  setup_users <- function(con) {
17    print("Establishing Users")
18    query      <- dbSendQuery(con, "SELECT * FROM users")
19    userresult <- fetch(query, n=-1)
20    cleared    <- dbClearResult(query)
21    size       <- dim(userresult)[1]
22    users      <- c(userresult[,1])
23    users
24  }

```

Once we have our list of users, we can then construct the dataframe by finding the weight of the user from individual database queries.

```

26  user_weights <- function(con) {
27    print("Establishing Edge Weights")
28    users      <- setup_users(con)
29    size       <- length(users)
30    weights    <- c()
31
32    get_weight <- function(con, user, i) {
33      query    <- dbSendQuery(con, gsub("%1", user,
34                                     "SELECT favourites_count,
35                                     followers_count FROM users
36                                     WHERE id=%1"))
37      result   <- fetch(query, n=-1)
38      cleared  <- dbClearResult(query)
39      weight   <- FAVORT * result[1,1] + FOLLOW * result[1,2]
40      weight
41    }
42    lapply(1:size, function(x) weights <- append(weights,
43                                                get_weight(con,
44                                                            users[x],
45                                                            x)))
46    weights_data <- data.frame(user=users, weight=weights)
47    weights_data
48  }

```

This dataframe construction gives us the following dataframe. Please see Appendix 3 to see a more complete dataframe.

<sup>7</sup>In R, a dataframe is a special datatype that is useful for many applications. For more information, I would recommend <http://stat.ethz.ch/R-manual/R-devel/library/base/html/data.frame.html>

390267816	2349.00
1167483505	33.00
853887943	34.00
325869713	1798.00
898986895	902.00
1328457397	1616.00
1355302483	721.00
1266058160	3682.00
515334031	366.00
230436012	1082.00
371462259	1946.00
298627315	2740.00
449047486	5743.00
630746058	11330.00
1143867078	4359.00
461854939	87.00
707394641	653.00
204178374	328.00
454467056	648.00
1740729596	1406.00
555636619	1183.00
783648271	2189.00
582951657	7693.00
216346618	659.00
1656121652	1304.00
714914388	4165.00
1370674644	2044.00
341134422	1336.00
580051343	6002.00
1451680514	3076.00

(2)

Now that we have our user weights dataframe, we can determine the links between users, and construct yet another dataframe consisting of all follower links between our users.

```

50 get_links <- function(con, users) {
51   print("Establishing User Links")
52   query <- dbSendQuery(con, "SELECT id, followers
53                               FROM users
54                               WHERE followers!= '[]'")
55   result <- fetch(query, n=-1)
56   cleared <- dbClearResult(query)
57   links <- data.frame(from=c(), to=c())
58   lapply(1:dim(result)[1], function(x)
59     links <- rbind(links[1:dim(links)[1],],
60                   result[x,]))
61 }

```

We finally have all the information we need in the correct formats in order to create our matrix. Four separate functions are referenced in order to optimize our code for speed. These functions are described below.

```

71 fill_matrix <- function(con, users, links) {
72     size      <- nrow(users)
73     rankings <- create_matrix(size)
74     print("Adding Retweet Edges")
75     query     <- dbSendQuery(con, "SELECT * FROM retweets")
76     result    <- fetch(query, n=-1)
77     cleared   <- dbClearResult(query)
78     print("    Replacing Data")
79     mclapply(1:dim(result)[1], function(x) rankings <- replace(rankings,
80                                                                result,
81                                                                users,
82                                                                x))
83     print("Adding Follower Edges")
84     query     <- dbSendQuery(con, "SELECT id, followers
85                                   FROM users
86                                   WHERE followers!='[]'")
87     result    <- fetch(query, n=-1)
88     cleared   <- dbClearResult(query)
89     mclapply(1:dim(result)[1], function(x) rankings <- followers(rankings,
90                                                                result,
91                                                                users,
92                                                                x))
93     print("Normalizing")
94     mclapply(1:size, function(x) rankings <- normalize(rankings,x))
95     print("Writing")
96     write.big.matrix(rankings, "output.txt")
97     rankings
98 }

```

### 2.4.1 Create Matrix

```

63 create_matrix <- function(size){
64     # Create our nxn file backed matrix
65     rankings <- big.matrix(size, size, type="double",
66                            backingfile="rankings",
67                            descriptorfile="rankings.desc")
68     rankings
69 }

```

This generates an  $n \times n$  filebacked matrix to be populated later.

### 2.4.2 Replace

```

122 replace <- function(rankings, result, users, row) {
123   # /T
124   # -/-
125   # F/
126   row      <- result[row,]
127   from     <- row[1,1]
128   to       <- row[1,2]
129   i        <- which(users$user == from)
130   j        <- which(users$user == to)
131   c        <- strtoi(row[1,3])
132   weight   <- RTWEET * c + users[i,2]
133   rankings[i,j] <- weight
134   rankings
135 }

```

Replace takes each row in our SQL query and adds a link from retweets in our adjacency matrix in the correct position.

### 2.4.3 Followers

```

100 followers <- function(rankings, result, users, row) {
101   row      <- result[row,]
102   user     <- row[1,1]
103   followers <- strsplit(row[1,2], ", |[^0-9]")
104   follow_count <- length(followers[[1]])
105
106   get_link <- function(rankings, users, to, from) {
107     i      <- which(users$user == from)
108     j      <- which(users$user == to)
109     weight <- FOLLOW + users[i,2]
110     rankings[i,j] <- rankings[i,j] + weight
111     rankings
112   }
113
114   mclapply(1:follow_count, function(x)
115     rankings <- get_link(rankings,
116       users,
117       user,
118       followers[[1]][x]))
119   rankings
120 }

```

Followers takes each row in our SQL query and adds a link from followers in our adjacency matrix in the correct position.

### 2.4.4 Normalize

```

137 normalize <- function(rankings, x) {
138     row      <- rankings[x,]
139     total    <- sum(row)
140     if (total == 0) {
141         total <- 1
142     }
143     rankings[x,] <- row / total
144     rankings
145 }

```

Normalize replaces each row with the row divided by its sum. This gives us a probability matrix.

## 2.5 Analysis

In order to determine the rankings of the users, we first construct an arbitrary vector with length equal to the number of users with unit length 1. Our non-orthonormalized matrix is of the form

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \quad (3)$$

We then pass the vector and matrix to our power method which runs however many iterations are specified. Please see the beginning of this section for more explanation on how the algorithm works.

```

151 power_method <- function(rankings, A, iterations, users) {
152     q = list()
153     q[[1]] <- A
154     for (i in 2:iterations) {
155         q[[i]] <- rankings[,] %*% q[[i-1]]
156     }
157     write.table(q[[iterations]], file="eigenvector.txt")
158     q
159 }

```

These two steps will run as many times as specified, and when complete (or nearly so) they will give us an approximation to our eigenvector. This eigenvector has a direct one-to-one correspondence to our user's relative rank compared to this network.

## 2.6 Analysis

Using the data available we were able to successfully rank the users on their relative influence compared to the selected group. In 4 we only have 26 non-zero values in our approximated eigenvector. Our full results can be found in Appendix 2

$$\left( \underbrace{\begin{pmatrix} 14 \\ 21 \\ 23 \\ 32 \\ 33 \\ 34 \\ 42 \\ 52 \\ 53 \\ 57 \\ 61 \\ 65 \\ 68 \\ 69 \\ 76 \\ 88 \\ 89 \\ 100 \\ 102 \\ 114 \\ 116 \\ 117 \\ 155 \\ 1018 \\ 1457 \\ 2692 \end{pmatrix}}_{\text{Index}} \underbrace{\begin{pmatrix} 6.64949232632148e - 06 \\ 1.09911011888071e - 05 \\ 11.855584902313 \\ 0.000579864962983945 \\ 0.000151203180869463 \\ 5.66269756293866e - 05 \\ 3.43608910353295e - 06 \\ 0.000207441307238603 \\ 0.000596997173695231 \\ 6.64949232632148e - 06 \\ 5.53795798073443 \\ 0.00014002408432437 \\ 0.000579864962983945 \\ 0.000810638056052665 \\ 0.000108830685549892 \\ 11.8555814662239 \\ 0.000105152995730235 \\ 0.000171495791986021 \\ 5.27313420768053e - 05 \\ 3.43608910353295e - 06 \\ 3.43608910353295e - 06 \\ 5.95354325766195e - 05 \\ 22.8520643143014 \\ 0.000596997173695231 \\ 0.000207595719100648 \\ 1.36186831565854e - 05 \end{pmatrix}}_{\text{Value}} \right) \quad (4)$$

If you'll note, the user with the highest score is user 155, which correspondes to Twitter user 27260086. This user's screen name is "justinbieber".

Looking at our user connections we can start to determine what our graph looks like. We have one central user that has many connections leading to him, however we also have many other users that contribute solely to other users, and often aren't related to our "main" user at all. These users are often dissacociated from our main user by at least one degree, and we have up to four degrees of separation when you look at our furthest distant user.

If we look at the entire expanded network we see a similar picture. Our users are clustered around one main user, and it is obvious that the network has one focal point, however when examined closely it becomes apparent that some users have no connections to our main user and instead point to just one sub-user instead.

Given our results these networks make intuitive sense; we have one user that is focused upon, and everyone else is of little to no importance. When looking at Twitter's population this result begins to hold true for a large portion of Twitter users. Most users are either fairly passive or of too little consequence to be highlighted.

### 3 Conclusion

#### 3.1 Implications

Aside from friends being able to look at one another's popularity, our analysis offers assistance in several different fields. Most prominently, advertising companies could utilize the ranking to find the most influential Twitter users and work with them to reach as many people as possible. Since every interaction on Twitter is available to everyone with their development system, anyone could perform similar analysis to ours.

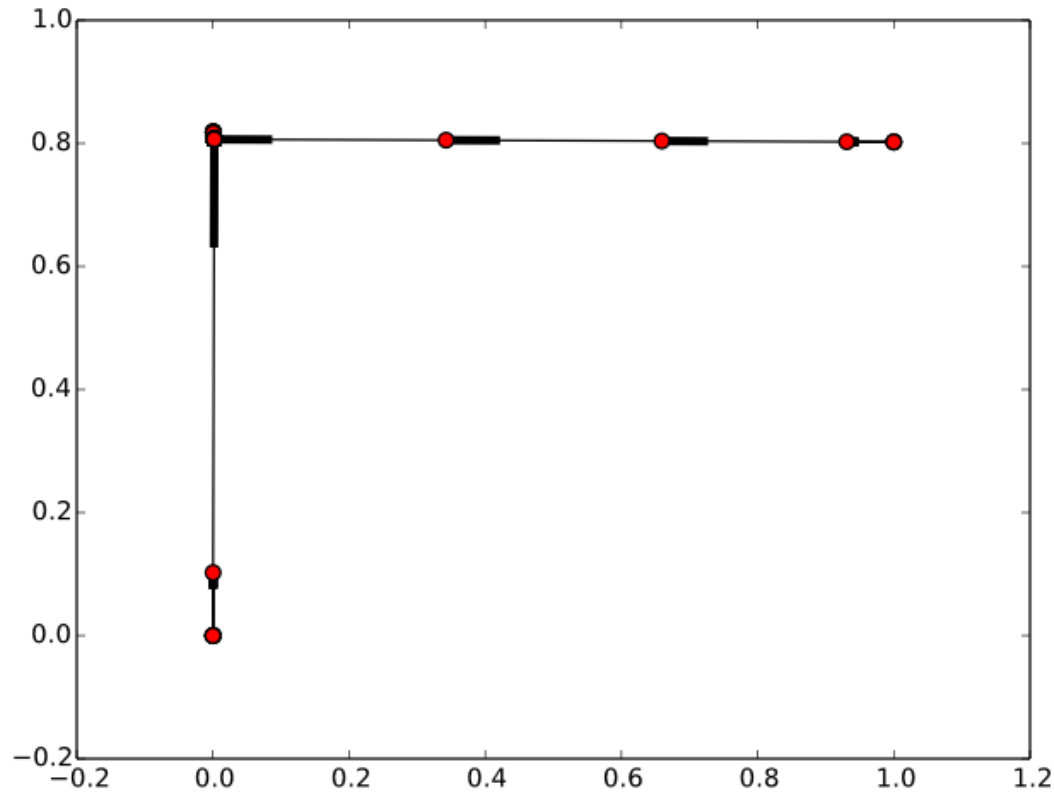


Figure 1: Simplified Links of our Established Network

As a company, Twitter could even find this process useful. Being able to find their most active users would be an incredibly useful tool for analysis of the functionality of the website. While we only sampled a very small subset of users, Twitter could include and compute the ranking with every user in the data set—something even more valuable.

Beyond these uses, an interesting abstract idea is the thought that due to applications of matrix methods, one can literally assign a human a ranking of influence. While not everyone in the world uses Twitter and our definition of influence is not absolute, it is still peculiar that an entire population can be ranked by popularity with ease, despite a large data set.

A simplistic demonstration of this potential service is available at [www.will-farmer.com/twilight](http://www.will-farmer.com/twilight). This website was created by William Farmer, and uses a simplified algorithm to rank the users.

### 3.2 Discussion

After implementing the power method and isolating the final dominant eigenvector, we had an explicit ranking of the users in our data set. An interesting note in this eigenvector was the large amount of users with negligible influence; out of nearly 3000 users, only 26 had an influence value greater than zero. This did not come as too much of a surprise due to the nature of our sample data set. Many of these users likely retweeted one of the more influential people and had no other interactions, diminishing their relative influence.

This clear difference in number of users with appreciable influence and those with almost no influence imply that the vast majority of users on Twitter are not all that influential. Our data suggests that in fact there are very few users who truly hold significant weight. As a matter of fact, we found one of the influential users in our data set was none other than Justin Bieber. While he does not live in Boulder, several people

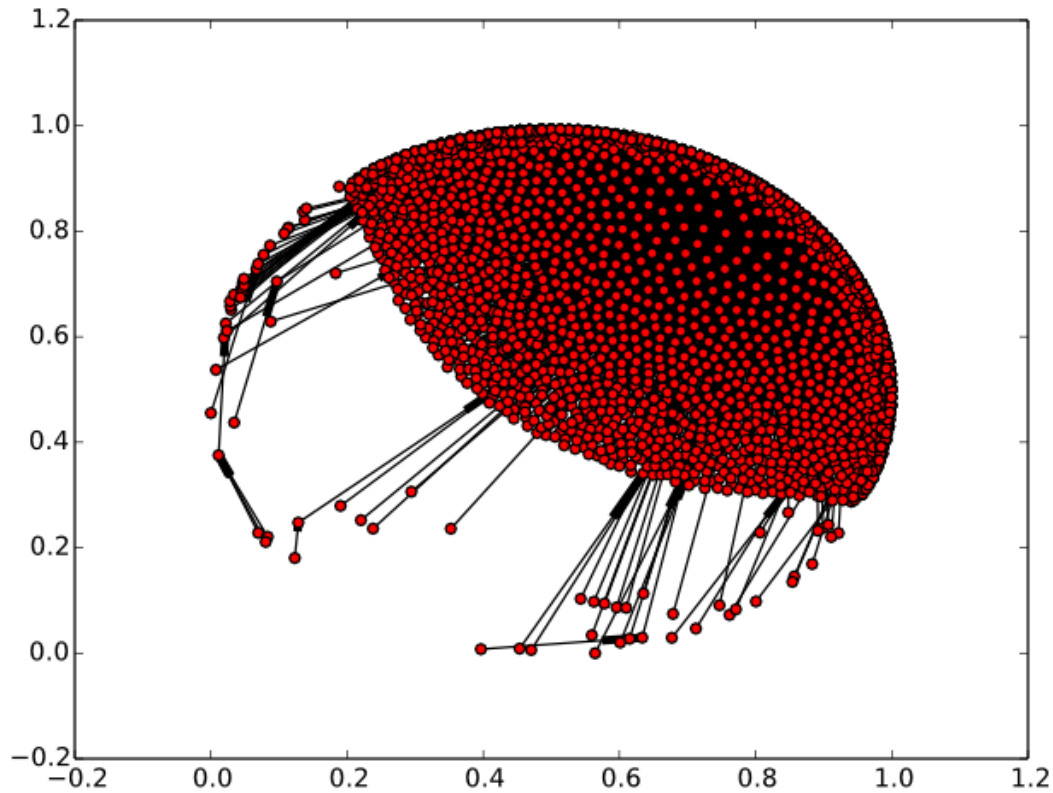


Figure 2: All Users of our Established Network

must have retweeted him and thereby introduced him to our data set. With his some 47 million followers, it does not come as a surprise that he was one of the most influential users on Twitter in our analysis.

This does relay a rather dismal message to almost the entire remainder of Twitter users however—they are just not that important. With the massive amount of followers and retweets that celebrities like Justin Bieber have, it is highly unlikely that a random user's tweets will reach the majority of the users on Twitter. However, this will certainly not keep people from using Twitter. Many users have an account simply to follow celebrities or comedians. Other have one to keep up with their sports teams or local news stations. Having a Twitter account is not always about trying to be the most popular user. Instead, it serves as a means of quick and easy communication for friends and celebrities alike.



# 1 Workflow

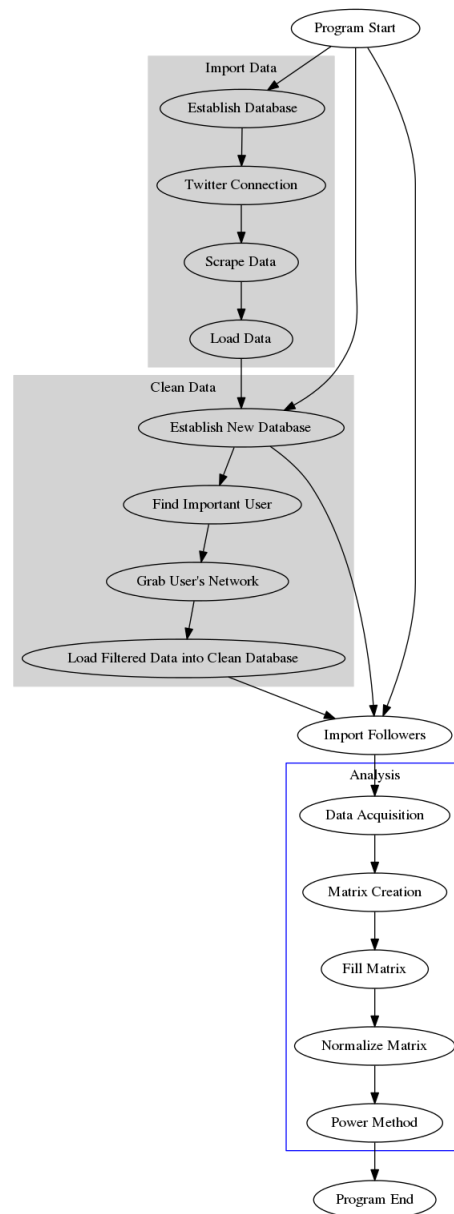




Figure 3: Workflow Diagram

## 2 Eigenvector

Our .csv eigenvector can be found  here.

## 3 User Weight Dataframe

Only users with a weight greater than 10,000 have been shown. For a full listing, please download  User Weights.

Index	User	Weight
14	630746058	11330.00
34	326405936	10767.00
42	624400312	12045.00
51	564366503	13193.00
61	94794703	11244.00
90	186963547	19042.00
112	327089971	16478.00
117	540112867	10525.00
120	411679941	10467.00
122	970302428	35038.00
127	609521711	21402.00
128	100131802	11997.00
145	634976070	28744.00
146	106689852	14948.00
155	27260086	47171703.00
159	341911569	21805.00
179	1159366760	11572.00
192	921812846	11562.00
200	599747318	11904.00
228	229478749	10904.00
261	511016804	14763.00
263	712518012	28174.00
276	452386369	33685.00
279	1663040792	10516.00
293	1546396718	11648.00
322	554499917	23350.00
324	35471715	12698.00
334	342297272	13392.00
389	233757681	14670.00
425	303852627	17833.00
433	339153652	16946.00
436	760144261	13995.00
464	272219232	13412.00
465	415096035	10355.00
493	251328265	13418.00
508	387980396	14130.00
526	281816610	10467.00
527	1049301990	13737.00
534	249763046	19570.00
544	334612319	21648.00
552	1462426686	18719.00
559	1096938487	12634.00
571	884698502	72336.00
601	1191641258	10118.00
645	261267312	12640.00
667	701679588	60758.00
681	344123447	10241.00
689	333962617	17114.00

Index	User	Weight
713	1054943286	17851.00
719	806434044	13066.00
755	136494576	13272.00
769	379477396	12866.00
823	1152077396	16730.00
835	1370034217	10423.00
841	273177183	15853.00
848	134300287	17914.00
857	228034203	17240.00
859	1628657935	11793.00
901	1253343157	10403.00
903	156449730	26229.00
914	425369749	14645.00
919	251670427	29269.00
923	349161702	12209.00
948	423893401	10675.00
950	302028075	10678.00
957	725168408	14443.00
964	98008008	15278.00
976	329264315	24567.00
979	1267870554	86059.00
996	716045551	16734.00
1010	548569720	11015.00
1021	405197592	13711.00
1026	364363813	10473.00
1030	223181418	257578.00
1041	155871002	21075.00
1065	1411164529	19633.00
1073	254761626	11840.00
1075	99368285	11159.00
1079	277459459	13662.00
1082	43992418	20170.00
1090	1231287487	11075.00
1091	241127563	16494.00
1100	711021564	16203.00
1108	465144648	14484.00
1119	489746004	51657.00
1146	31775600	133275.00
1165	502141732	27772.00
1170	946608667	140350.00
1181	1177058718	11267.00
1195	275192872	45550.00
1210	630470363	11932.00
1211	508786354	15920.00
1251	101854971	14403.00
1289	1061740729	24610.00
1294	174465013	13857.00
1300	1011329257	11241.00

Index	User	Weight
1304	860230987	11241.00
1341	551246662	13501.00
1343	40347131	11357.00
1358	549484617	22568.00
1359	574715600	25299.00
1362	336859526	14243.00
1384	794528744	18068.00
1387	224921031	18339.00
1388	124264446	10107.00
1414	126433456	16746.00
1416	475655739	13170.00
1448	157712893	17453.00
1449	1157273696	14516.00
1457	585786517	19082.00
1485	775832820	12645.00
1508	334384978	10991.00
1521	441491566	11127.00
1524	457618957	10121.00
1535	255645831	23107.00
1578	634994358	47900.00
1582	327487739	12857.00
1591	1063040520	39343.00
1596	102961107	27764.00
1608	162800521	11485.00
1612	144684377	25852.00
1619	436346220	10792.00
1698	251323953	10697.00
1702	946589106	12424.00
1777	302081484	10850.00
1789	611843107	13880.00
1796	878402719	26858.00
1824	211530541	13620.00
1853	310640211	27623.00
1857	1664339594	26089.00
1876	336809891	43156.00
1884	465972676	30061.00
1911	28842209	13711.00
1919	439269298	12527.00
1923	1148452152	10959.00
1926	370383777	17931.00
1933	1106839123	11395.00
1966	99172004	13172.00
1983	741571238	16967.00
1985	184832093	11345.00
1986	303138113	38332.00
2026	1160060478	14318.00
2073	351169230	10909.00
2091	135682030	13102.00

Index	User	Weight
2092	768007074	10869.00
2096	478087852	10064.00
2109	569249586	11391.00
2117	1508402305	11422.00
2119	235746080	12850.00
2124	373361072	22091.00
2140	731538752	26125.00
2156	1340799422	10541.00
2197	353451876	29308.00
2238	1591908384	14157.00
2290	188971014	11325.00
2324	110328199	18428.00
2347	306343981	12405.00
2364	790831482	23295.00
2374	1772231389	10633.00
2402	534633467	24961.00
2437	624785810	13959.00
2448	164465678	12599.00
2454	571239776	14849.00
2455	705288404	15074.00
2457	329486211	11534.00
2463	608724448	12165.00
2468	787472516	15487.00
2480	225080589	10576.00
2505	411006479	29723.00
2509	184891657	12073.00
2513	238775866	20700.00
2525	84546155	16784.00
2543	406328788	17930.00
2553	275733498	12932.00
2603	558826929	11854.00
2640	397514527	11000.00
2664	217578833	14439.00
2667	433687416	14952.00
2687	797115074	25631.00
2692	1345120957	25628.00

## 4 Attachments



Report L<sup>A</sup>T<sub>E</sub>X



Report Programs

## 5 Authors

If you have any questions about methods used, are unable to access attached files, or wish to discuss the report more with the authors, please email [willzfarmer@gmail.com](mailto:willzfarmer@gmail.com) or visit [www.will-farmer.com](http://www.will-farmer.com) where this paper will be posted.