University of Texas Rio Grande Valley

ScholarWorks @ UTRGV

Theses and Dissertations - UTB/UTPA

5-2015

Use of a simulated directional social network to compare measures of user influence

Jose G. Villarreal Jr.
University of Texas-Pan American

Follow this and additional works at: https://scholarworks.utrgv.edu/leg_etd

Part of the Computer Sciences Commons

Recommended Citation

Villarreal Jr., Jose G., "Use of a simulated directional social network to compare measures of user influence" (2015). *Theses and Dissertations - UTB/UTPA*. 197.
https://scholarworks.utrgv.edu/leg_etd/197

This Thesis is brought to you for free and open access by ScholarWorks @ UTRGV. It has been accepted for inclusion in Theses and Dissertations - UTB/UTPA by an authorized administrator of ScholarWorks @ UTRGV. For more information, please contact justin.white@utrgv.edu, william.flores01@utrgv.edu.

USE OF A SIMULATED DIRECTIONAL SOCIAL NETWORK TO COMPARE MEASURES OF USER INFLUENCE

A Thesis

by

JOSE G VILLARREAL JR

Submitted to the Graduate School of
The University of Texas-Pan American
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2015

Major Subject: Computer Science

USE OF A SIMULATED

DIRECTIONAL SOCIAL NETWORK

TO COMPARE MEASURES OF USER INFLUENCE

A Thesis
By
JOSE G VILLARREAL JR

COMMITTEE MEMBERS

Dr. Christine Reilly
Chair of Committee

Dr. Laura Grabowski
Committee Member

Dr. Bin Fu
Committee Member

Dr. Xian Lian
Committee Member

May 2015

Copyright 2015 Jose Villarreal Jr
All Rights Reserved

ABSTRACT

Villarreal, Jose G., Use of a Simulated Directional Social Network to Compare Measures of User Influence. Master of Science (MS), May 2015, 120 pp, 22 references.

This paper proposes a new method for measuring user influence in directional social networks, derived from the works of Reilly et al. and Cha et al. The method being proposed in this paper considers an element from each of the two works. The first is the ratio of ‘messages forwarded’ over ‘messages posted’. The second element is the size of the audience.

The second part of this study entails modeling and simulating an online social network. Using a data sample from the Twitter network to implement the simulation, it is going to allow us to compare the methods that are used to measure influence. The behaviors modeled include the act of gaining a follower, the act of creating a message, and the act of forwarding a message. These are the three behaviors we are using to compute influence.

DEDICATION

The completion of my graduate studies would not have been possible without the support of the following individuals: my family: mom, my two sisters, my ‘Bug Bug’, and Noah– for all your love and support. Shamika Y. Satchell – without your support, I would not have been able to make this dream a reality – I’m forever grateful to you! Alex – for believing me in.

ACKNOWLEDGEMENTS

I would like to thank Dr. Christine Reilly, my thesis advisor, for allowing me the opportunity to work with her and learn from her. I am grateful for her advice, support, encouragement, patience and guidance throughout the entire process. She challenged me and she pushed me out of my comfort zone.

I would also like to thank my Thesis committee: Dr. Bin Fu, Dr. Xian Lian and Dr. Laura Grabowski. In 2004, I took my very first computer science class with Dr. Grabowski. Her participation in the committee means a lot to me. I would also like to thank the Computer Science faculty for all their support through this journey. I would also like to thank the Math department for sharing the LaTex template that made the creation of this document easier.

I am not a man of many friends, but in the last three years, I've met some pretty amazing people who have offered me nothing but support and encouragement: Erricka "EJJ" Jones, Adam 'A-Jay-Bay" Bailey, Claudia, Santiago, & my Chase family in Houston. I would also like to thank Mike Samford for giving me that one opportunity.

TABLE OF CONTENTS

Page

‘CHAPTER I

INTRODUCTION

Identifying influential users in society has been a scientific topic of study and interest since the late 19th Century. There are many ideas and theories as far as what influences social behaviors. Gustave Le Bon published his theories of behaviors in groups, based on the concept of the group mind: the crowd takes over the will of the person. Others before Le Bon had already promoted such ideas. Norman Triplett conducted the first experiment to investigate social influence process in 1895 [Pratkanis 2007]. The research continues today. One of the primary focuses of this type of research is to learn how people influence each other.

The growth of online social networks has been nothing short of phenomenal. Some networks have managed to establish themselves with the public (Twitter, Facebook, Instagram). Other networks didn’t gain popularity or lost their popularity over time (blogs, MySpace). The use of social networks has grown in very little time and it’s evident that social networks are not going away anytime soon. Social networks have broken communication barriers. Social circles are no longer restricted to a small community of people as they once were. Social networks are now a form of socializing and there is interest in studying and analyzing such interactions, including identifying influential users.

Why study social influence in online social networks? For one, the amount of data available is abundant. The technology available today makes it possible to store every interaction that has occurred within each network. We have complete data sets. It may be challenging to acquire the data, but the data exists. There is opportunity to study social influence at many levels, from small groups to world-wide interactions. In contrast to when Norman Triplett conducted his experiment, technology today makes it feasible to compute/automate data collected from studies, rather than doing it by paper and pen, making it prone to less errors, and it yields away from the incomplete and inaccurate data social scientists once had to work with. Online social networks and technology make it feasible to work with large data sets. What once took years to calculate and analyze can now be computed in a matter of minutes, hours, or days. As previously mentioned, social circles are no longer bound by barriers. Social influence is not hindered – it's at full blossom. The opportunity is golden not only for social scientists but also for computer scientists.

For this study, we will be focusing on a Twitter-like network to identify influential users. Twitter's structure disperses conversation throughout a network of interconnected actors rather than constraining conversation within bounded spaces or groups, many people may talk about a particular topic at once, such that others have a sense of being surrounded by a conversation, despite perhaps not being an active contributor. The stream of messages provided by Twitter allows individuals to be peripherally aware without directly participating [Boyd 2010]. Twitter has an API (application programming interface) available publicly by default, from which Twitter data can be collected. Other social networks have APIs available, however, they are private by default.

This thesis is organized as follows: Chapter 2 provides a descriptive overview of the Twitter Social Network. Chapter 3 elaborates on work previously done on identifying influential users as well as work previously done on modeling and simulating online social networks. In Chapter 4, the influence methods that will be implemented for this study are described in detail.

All functions and computations are defined. Chapter 5 is a detailed description of the simulation of the social network. All computations used in the simulation are defined. Chapter 6 provides details of the implementation of both the influence calculations and the social network simulation. In Chapter 7 we conclude.

CHAPTER II

THE TWITTER SOCIAL NETWORK

Twitter, a microblogging service, is currently one of the most used social networks worldwide. Launched in 2007, it currently has over 288 million active users monthly. More than 500 million messages are posted per day. One is able to socialize either via web-browser at home on a computer, or via the Twitter mobile application on a tablet or smartphone. The platform is supported across multiple operating systems, making the network available to virtually everyone. 80% of active users interact via mobile devices. 70% of its user accounts are non-US users [About Twitter 2015].

In addition to interpersonal communication, Twitter is increasingly used as a source of real-time information. Televised events cause massive real-time spikes in global Twitter activity; entertainment news and events also result in high tweet volumes. Public figures and celebrities from the Pope to Lady Gaga attract enormous numbers of followers, and a photo of Barack and Michelle Obama, posted immediately after Obama's re-election as President of the United States in November 2012, rapidly became the single most retweeted message in the history of Twitter. Disasters such as Hurricane Sandy, and tragedies like the shooting spree of a gunman at the Sandy Hook Elementary School in Connecticut (both in the autumn of 2012) show their immediate aftereffects on the platform, as users report their experiences and search for information, often as events are unfolding - a dynamic that makes Twitter seemingly irresistible to the mass media. Such moments demonstrate how deeply embedded the service has become into the everyday lives of its users around the world. Increasingly, when noteworthy events occur—both on a global and a local level—there will be Twitter users who share the news. Twitter remains a space for mundane expressiveness and interaction: millions of private users chat with their friends and share photos or URLs via Twitter at any one point, using the service as a journal of their thoughts and everyday activities [Weller 2014].

To post messages in Twitter, one has to subscribe to the service. Without a subscription, one is still able to view messages and content. However, interacting with users is not possible. Upon signing up, one creates a profile and chooses a user name. User names are unique, and it is by this user name that one becomes known in the network. In messages, one is addressed by this user name. Users have the option to upload a picture to their profile. Once a member, one is able to search for friends and family that are members of the network.

Twitter participants at first were constrained to expressing themselves in 140 characters. As participants embraced the technology and its affordances, a series of conventions emerged that allowed users to add structure to tweets. For example, users developed ways to reference other users, converged on labels to indicate topics, and devised language to propagate messages [Boyd 2010].

In Twitter, users build a community by following other users. This is how a relationship between two users is established. Users have the option to 'follow' other user. 'Following' someone allows a user to keep up with the communication activities of that individual. They are not only able to view the messages or content this user posts but also have the ability to forward this content to other users. The relationship can be reciprocated at will, meaning the user can (but does not have to) follow right back. At any time, a user is able to view how many followers they have and who those followers are. Users have the option the set their profile to private. Users set to private can be followed by request only. Users also have the ability to block a follower. When a follower is blocked, they are no longer able to view a user’s profile or any of the user’s content.

Messages propagated in Twitter are called Tweets (hence the name of the network). Messages are limited to 140 characters (the limit was imposed to allow SMS compatibility). Tweets can be general messages, they can be directed at a specific user, a.k.a. Mentions (these are identified by the symbol “@” followed by the user’s user name), and as of recent, they can be aimed at specific trending topics (these are identified by the use of the symbol “#”, also known as hashtag). Users are also allowed to Tweet pictures and URL’s of websites (URL’s are shortened to adhere to the 140 character limit). Unfortunately, Twitter also is falls victim to spam. Spam tweets have increased in Twitter as the popularity of Twitter grows. The Twitter Support Team suspends any user reported to be a spammer. Still unreported spam tweets can creep in [Kwak 2010]. Although people can interact with Twitter directly through the website, there are many third party applications available, ranging from mobile to desktop Twitter clients [Boyd 2010].

Twitter participants began using the @user syntax to refer to specific users (e.g., @username) to address one another. This convention serves multiple purposes in Twitter, including directing messages to specific people as though sending the message to them (also known as @replies), and to obliquely reference another user (e.g. “I saw @oprah’s show today”) [Boyd 2010]. In the Twitter platform, these are known as Mentions. Cha et al. use Mentions as an element for measuring user influence.

Tweets can be forwarded by users to other users in the network. In Twitter, this is called a Retweet. Boyd et al. state retweeting is the Twitter-equivalent of email forwarding: users post messages originally posted by others. Retweeting brings new people into a particular thread, inviting them to engage without directly addressing them [Boyd 2010]. Retweeting is a common convention in Twitter. When a user finds an interesting tweet written by another user and wants to share it with her followers, [he/she] can retweet the tweet by copying the message. This can be done in one of two ways. First, one can retweet by preceding it with RT and addressing the original author with @. For example, “RT @userA: my experience with the new #iPad is great!” Second, Twitter also enables users to retweet easily with one-click [Suh 2010]. As Reilly et al. point out, the retweeting process can be illustrated by considering two hypothetical users, Alice and Bob. Bob follows Alice and reads her tweets. When Bob reads a tweet that he finds particularly interesting, he may choose to retweet that tweet. Now Alice’s tweet will be sent to all of Bob’s followers. Messages are spread through Twitter by cascades of retweets. Because most users allow their tweets to be publicly broadcast, Twitter is an appealing social network to use for research purposes. [Reilly 2014]. Unlike @replies and hashtags, the conventions for retweeting are hugely inconsistent [Boyd 2010].

The central feature of Twitter, which users see when they log in, is a stream of tweets posted by those that they follow, listed in reverse chronological order. Participants have different strategies for deciding who they follow—some follow thousands, while others follow few; some follow only those that they know personally, while others follow celebrities and strangers that they find interesting [Boyd 2010]. By default, when a user launches Twitter (either via browser or the mobile app), they are routed to this stream of tweets. Users who are not following anyone will see only advertisement/promotional Tweets in this section.

For this study, we will be using data from the Twitter network to model and simulate user influence. Twitter is ideal because of its current immense popularity. As mentioned prior, Twitter has a public API that allows its public user data to be collected. We have the opportunity to study social influence from a directional relationship perspective. In the data one can collect from the API, there are several elements that can be considered in computing user influence: Tweets, Retweets, number of Followers, Mentions, user's geographical location, age of account, among other elements. This same data can be used to simulate and model a directed social network. The simulation is useful because collecting data from the Twitter API is restrictive. To amass a large data set, one would have to be collecting data for a very long period of time. Even then, there is no guarantee that the data one has collected is a complete set.

The Twitter API offers Twitter data at no cost. One can obtain historical data from the REST API. The Streaming API is used to process Tweets in real time. To obtain data, one must first obtain permission to access the API and agree to abide by the Terms of Use, and Developer Policy and Agreement. Twitter imposes API Rate Limits, which control the data a user or application can request/collect. The current rate limits are divided into 15 minute intervals. Depending on what data is being requested, the Rate Limits chart indicates how many requests can be made every 15 minutes. For example, the GET followers/list, which returns a collection of user objects for users following a specified user, allows up to 30 requests (via application access) in a given 15 minute interval. Each result returns a group of 20 users. In a 15 minute interval, one would be able to gather a list of up to 300 followers for a specified user. Users or applications that exceed (abuse) the rate limits will be blacklisted. Once blacklisted, the Twitter API will not respond to requests [Twitter API 2015].

These limitations affect the ability of third-party developers, users, and researchers to exploit or innovate upon the platform [Weller 2014]. Though historical data is available for purchase from third party vendors such as Gnip, the data is expensive. Cha et al. contacted Twitter administrators and they were allowed to collect data via 58 servers. Their IP addresses were whitelisted. They were able to collect data for over 54 million users, including over 1,750,000,000 tweets [Cha 2010].

CHAPTER III

RELATED WORK

A. Related Work on Social Influence

A lot of research has been done on the social conventions of Twitter. Ideas and perspectives as far as who the influential users are differ.

Boyd et al. believe that people retweet for diverse motivations. The goal of their paper was to describe and map out the various conventions of retweeting, and to provide a framework for examining retweeting practices. At the time their study was conducted, Retweeting was not yet a stabilized practice: participants didn't know how retweets were supposed to work. Those that did retweet were not representative of all types of Twitter users. They believed that as Twitter and other technologies begin providing features to support practices like retweeting, it is crucial to understand the diversity of behaviors taking place. As more scholars begin examining Twitter, it is important to have a grounded understanding of the core practices. Though Twitter was not universally adopted at the time the paper was written, Twitter supported an active community with its own set of unique practices that are valuable to examine. Retweeting can be understood both as a form of information diffusion and as a means of participating in a diffuse conversation. Spreading tweets is not simply to get messages out to new audiences, but also to validate and engage with others. Retweeting could be both a productive communicative tool and a selfish act of attention seekers. [Boyd 2010]

Razis et al., in studying the diffusion of information on Twitter, proposed a method for measuring user importance and influence. They state the measurement should not depend merely on the number of “Followers” of a user, even if that number is big enough and the user’s tweets are received by a large number of other users (followers). In case that the number of “Following” is larger, then the user could be characterized as a “passive” one. A passive user, as previously defined by Romero et al. is one who does not forward content in the network. According to Razis et al., these type of users are regarded as those who are keen on viewing or being informed through tweets rather than composing new ones. They believe a more suitable factor is the ratio of “Followers to Following” (FtF ratio). However, this factor alone is not enough. They also take into account a user’s Tweet Creation Rate (TCR), which is calculated based on a user’s last 100 Tweets according to the Twitter API. They believe a user with a high TCR rate has more impact within the network. In their method, influence is dynamic. It depends a user’s most recent activity. Razis et al. also support the notion that there is value in retweets. They state that retweets allows users to repost a received tweet to their Followers. This results in viewing the tweets of users who are not being directly followed. This leads to the diffusion of information to users not targeted (to the followers of their followers). The same process can be repeatedly take place for new viewers of the message. [Razis 2014]

Romero et al. set out to analyze information propagation within Twitter. They focused on collecting tweets, from the Twitter API, that contain the string “http”. They found that the majority of users act as passive consumers, meaning that they do not forward content to the network. They refer to these users as passive users. They found that on the average, Twitter users retweeted one in 318 URLs, a relatively low value. A Uniform Resource Locator (URL) is used to specify specific addresses on the World Wide Web. They are convinced that users who receive information either chose to ignore it or never see it: Twitter users are generally passive. While they state that obtained an estimated 22 million tweets containing URLs, and 15 million of those were unique URLs, they do not mention a total count of user. They propose that in order for users to become influential they must first overcome user passivity. They describe a second element needed for influence, dedication. Dedication is measured by the amount of attention a user pays to a given one as compared to everyone else. In their model, a user's influence score considers the number of people they influence, their passivity as well as their dedication. They also support the notion that Followers and Retweets are good indicators of influence in the Twitter network. [Romero 2011]

Bakshy et al. offer a different perspective on identifying influential users. They classify influence into three different categories: first influence, last influence, and split influence. This method entailed analyzing the content of messages for specific URLS, and tracking how a message traveled through the network from its original author to the last person who retweeted it. To do this, they used the time each URL was posted. If person B is following person A, and person A posted the URL before B, and A is the only of B's friends to post the URL, they conclude person A influenced B to post the URL. However, if B has more than one friend who has previously posted the same URL, there are three choices for how to assign the corresponding influence: Assign full credit to the friend who posted it first, a.k.a. First Influence. Assign full credit to the friend who posted it last, a.k.a. Last Influence. Split credit equally among all prior-posting friends, a.k.a Split Influence. Each assignment makes a different assumption about the influence process. First Influence rewards primacy, assuming that individuals are influenced when they first see a new piece of information, even if they fail to immediately act on it, during which time they may see it again. Last Influence assumes the opposite, and instead attributes influence to the most recent (last) exposure. Split Influence assumes either that the likelihood of noticing a new piece of information, or equivalently the inclination to act on it, accumulates steadily as the information is posted by more. [Bakshy 2011].

Kwak et al. identify influential users based on the number of Followers and by Page Rank. The rankings by retweets were done on a sheer count of retweets. The key idea behind PageRank is to allow propagation of influence along the network of web pages, instead of just counting the number of other web pages pointing at the web page, as done by Google. They rank users in each of the two categories, and proceed with using Kendall's Tau to compare the ranks. They find the two rankings to be closely related. [Kwak 2010]

Others have also focused their efforts on analyzing the content of the Tweets in order to identify influential users. [Kwak 2010][Bakshy 2011][Liu 2013][Romero 2011][Yue 2013][Cha 2010] Our intuition is that 'topics' carry influence of their own, regardless of who propagates them. Identifying influence in 'topics' is a different type of study, worthy of a study of its own. The contents of Tweets will not be analyzed nor considered for this study. It is important to understand that an accepted paradigm for measuring online social influence does not exist. Research into online social influence is still under development. This research focuses on influence being asserted from one individual to others.

There are two additional methods related to identifying influential users that will be described in greater detail in Chapter IV. The first method consists of computing a Retweet to Tweet ratio for each user, for a specified time period. This idea is presented by Reilly et al. The second method consists of computing influence based on either a sheer number of followers, a sheer number of mentions, or a sheer number of retweets. This idea is presented by Cha et al. Both of these work are going to be implemented as part of this study. A third method, which is a method proposed in this study, is derived from Reilly et al. and Cha et al.'s work. This method is also described in greater detail in Chapter IV.

B. Related Work on Social Network Simulation

The manner in which social scientists analyze social interactions and human behavior varies largely. Some take a detailed approach and build models that account for many variables and parameters. Some proceed with simpler models. A simple model reduces the complexity of a given system and may be used to get a better understanding of how social mechanisms work. Though each have advantages and disadvantages, they serve a respective purpose. Regardless of the technique, computer simulation complements and aid in studies of real life scenarios. They help reveal patterns of human interaction. This allows for prediction on how certain changes in a system would affect its dynamic and outcome, ie. social influence. [Helbing 2012]

Gatti et al. created an agent-based Social Media Network Simulation which they refer to as SMSim. It is modeled as a discrete event simulation where events occur at an instant time, time steps. The basic agent actions are either to read a message or to post a message. The agent behavior is based on the Markov Chain transition probabilities, which are estimated from a data sample they obtained. The graph is represented as G = (A, R), where A is the set of agents and R is the set of followers relationships. [Gatti 2013]

Firat et al. simulated a social network to study the relationships between social actors, from isolated individuals to connected networks. They cluster data, a mathematical technique used by sociologists to unravel subgroups in a social network, using genetic algorithms. [Firat 2007] Liu et al. simulated a directed social network in order to analyze the spread of rumors in the social network. Their simulation is a bit more complex, as they simulate three types of users: spreaders, stiflers, and ignorants. [Liu 2011]

Online social networks are a widely used method for spreading information and news. Examples of online social networks that are currently popular include Facebook and Twitter. Facebook is an example of an undirected social network where the relationship between users must be reciprocal. In contrast, Twitter is a directed social network because the relationship between users need not be reciprocal. A study from 2010 found that only 21.1% of relationships in Twitter were reciprocal [Reilly 2014].

CHAPTER IV

APPROACHES FOR COMPUTING INFLUENCE

According to the Merriam-Webster dictionary, influence is defined as a person or thing that affects someone or something in an important way. An influential person is a person with an audience, an audience that is listening. Romero et al. believe that in order for individuals to become influential, they must not only obtain attention and thus be popular, but also overcome user passivity. A passive user is one who does not forward any content to the network [Romero 2011]. In Twitter, one forwards content by Retweeting.

This study is going to focus on implementing and analyzing two approaches for measuring influence that were previously proposed. The two methods are the works of Reilly et al. and Cha et al. Reilly's method yields one influence measure: normalized influence. Cha's method yields three influence measures: by followers, by retweets, and by mentions. The method we propose also yields one measure. The five results, which consist of lists of users ranked from 'most influential' to 'least influential', are going to be compared and contrasted to test if any of the measures are closely related.

A. Normalized Influence

Reilly et al. focused on identifying influential users within a set time period. They consider a user in a social network to be influential if the information he/she shares becomes widely disseminated throughout the network. They note that a user may forward information based on their interest in the information itself, not due to the person who sent the information. Additionally, forwarding information may or may not indicate the presence of factors that influence purchasing behavior or political opinions. Despite these limitations, they proceed with examining the use of information passing as a measure of influence in social networks because that is the behavior that is easily observable in the network [Reilly 2014].

They consider an influential user as one who has a high retweet per tweet ratio during a given time period. Influence is proposed as follows, where the influence of user i during a certain time period is the ratio of retweets to tweets:

$$I\mathrm{i} = \frac{Ni}{Mi}$$

where:

- I_i = Influence of user *i* in a certain time period
- N_i = The number of times user *i*'s tweets were retweeted in a certain time period
- M_i = The number of tweets posted by user *i* in a certain time period

Before users are ranked, influence is normalized by dividing the influence of each user by the sum of the influence of all users. Given the terminology above, along with the following additional terminology, the normalized influence of user i during a certain time period defined as follows:

$$\hat{I}i = \frac{Ii}{\sum_{k=1}^{U}(I_k)}$$

where

- $\hat{I}_i$= Normalized user influence
- U = Total number of users

During the specified time period, all users who have posted an original message are found. Next they identify the number of times that each of the user's messages have been retweeted during the given time period. The methods discussed above are used to compute the influence of each user and then rank the users by influence [Reilly 2014].

Reilly et al. point out how it is they identified retweets in their study. Twitter users utilize two different methods for retweeting: the native retweet and the organic retweet. In late 2009, Twitter added native functionality for retweets to its API. This functionality adds a retweet button to each tweet, when viewed in Twitter's graphical user interface. When a user wishes to forward a tweet, she clicks the retweet button on that tweet to forward the message to her followers. Counting the number of native retweets is a simple matter of using Twitter's API. The API provides a method that returns a list of the users who retweeted a particular tweet. Organic retweets on the other hand, arose from within the Twitter community. There are a number of organic retweeting practices, one of them being the common practice of adding "RT @username" to the beginning of a retweet. They simplified the identification of organic retweets by only searching for the "RT @username" syntax. By simplifying this, retweets that use alternative syntax are missed [Reilly 2014].

B. Indegree, Retweets, and Mentions

Cha et al. begin their paper by reiterating that there are different views as far as who influential users are and that there are different views on what makes a user influential. Though there have been important theoretical studies on the diffusion of influence, all have yielded radically different results. Traditional communication theory states that a minority of users, called influentials, excel in persuading others. A more modern view, in contrast, suggests that the key factors determining influence are the interpersonal relationship among ordinary users and the readiness of a society to adopt an innovation. One important thing they note is that these theories, are still just theories. Mainly because there has been a lack of empirical data to validate them. The recent boom of social networking sites and the data within such sites could allow researchers to empirically validate these theories [Cha 2010].

Using a large amount of data gathered from Twitter, they compare three different measures of influence: Indegree, retweets, and mentions. Focusing on different topics, they examine how the three types of influential users performed in spreading popular news topics. They investigate the dynamics of an individual's influence by topic and over time. They also characterize the precise behaviors that make ordinary individuals gain high influence over a short period of time. As stated in Chapter II, the team contacted Twitter administrators and they were allowed to collect data via 58 servers. Their IP addresses were whitelisted. They were able to collect data for over 54 million users, including over 1,750,000,000 tweets [Cha 2010]. For purposes of this study, we will only be analyzing and comparing the three measures of influence: Indegree, retweets, and mentions.

In their findings, they state that their analysis of the three influence measures yields a better understanding of the different roles users play in social media. Indegree represents popularity of a user; retweets represent the content value of one's tweets; and mentions represent the name value of a user. Hence, the top users based on the three measures have little overlap [Cha 2010].

Focusing first on an individual's potential to lead others to engage in a certain act, they highlight three "interpersonal" activities on Twitter. Tweets: Users interact by following updates of people who post interesting Tweets. Retweets: Users can pass along interesting pieces of information to their Followers. This act is popularly known as Retweeting. Retweets can typically be identified by the use of RT @username or via @username in Tweets. Mentions: Users can respond to other people's tweets. Mentions are identified by searching for @username in the tweet content, excluding retweets. A tweet that starts with @username is not broadcast to all Followers, but only to the replied user. A tweet containing @username in the middle of its text gets broadcast to all followers. These three activities represent the different types of influence of a person: followers, retweets, and mentions [Cha 2010].

The first element is the number of followers a user has, which they refer to as Indegree Influence. The second element is the number of retweets the user has generated, which they refer to as Retweet Influence. This measures the user's ability to generate content with pass along value. The third element is based on Mention, which measures the ability of a user to engage others in conversation. They refer to this as Mention Influence. Each user was ranked in each of the three categories.

- **Indegree Influence** – user with the most followers is ranked highest, and people below that sequentially
- **Retweet Influence** – user with highest retweet count is ranked highest, people below that sequentially
- **Mention Influence** – user with highest mention count is ranked highest, people below that sequentially

After all users were ranked in each category, the rankings were contrasted to one another as follows: Indegree to Retweet, Retweet to Mention, and Indegree to Mention. They used the Spearman's Rank Correlation Coefficient measure to quantify how a user's rank varies across different measures and examine what kinds of users are ranked high for a given measure [Cha

2010]. This measure is defined later in this chapter. They state that their inclusive and complete dataset guarantees reliability of the correlation estimates.

Rather than comparing the values directly, they used the relative order of users' ranks as a measure of difference. In order to do this, they sorted users by each measure, so that the rank of 1 indicates the most influential user and increasing rank indicates a less influential user. Users with the same influence value receive the average of the rank amongst them [Cha 2010].

In their approach, they found that normalizing retweets and mentions by total tweets would yield a different measure of influence, which might have led to very different results. They also found that normalization failed to rank users with the highest sheer number of retweets as influential. They adhered to using the sheer number of retweets and mentions without normalizing the values. Other measures such as the number of tweets and Outdegree (the number of people a user follows) were not found to be useful, because they identified robots and spammers as the most influential, respectively. These elements were not used [Cha 2010].

Cha et al. conclude that Indegree alone represents popularity but is not indicative of influence nor strong enough to engage an audience. Though there was a high correlation amongst the three pairs, but they discovered that the ties occurred only with the least influential users. Their analysis concludes that there is a stronger correlation between retweets and mentions [Cha 2010].

C. Proposed Method: Normalized Influence + Indegree

A user has the freedom to post as many messages as they'd like in a social network. We assume that in order to be influential, a user must meet two conditions: post messages and have an audience. Let's consider a user with a large audience. If he or she does not post messages, the user will never be propagated, thus, is non-influential. The same idea can be applied to a user who posts frequently but does not have audience. If he or she posts messages, there is no audience, no one to forward their content, and no one to influence.

When a message is passed along, or retweeted, this message is distributed outside its intended social circle. There is potential for the message to be further spread ahead. Our intuition is that Retweets are instrumental in identifying influential users. Retweets are not only indicative of influence, but they also give a user exposure to different audiences with different end results: A non-follower of this user ends up retweeting it, and/or the author of the Tweet gains new followers.

Many social scientists believe that influence is established over time [Pratkanis 2007]. In a social network, a user of a social network could gain followers over time. Users could establish themselves with an audience, a.k.a. their followers. The method being proposed is partly based on that idea.

Reilly et al. presented the idea of calculating a retweets to tweets ratio, during a given time period. An influential user is defined as a user who has a high retweet to tweet ratio [Reilly 2014]. Cha et al. presented the idea of using audience as a measure. In their work, they refer to this as Indegree influence. Users were sorted based on this measure, and rank 1 was assigned to the person with the most Followers, and an increasing rank to those with less. In case of a tie, users received the average rank amongst them. [Cha 2010]

Instead of ranking users based solely on the number of followers or their retweet to tweet ratio, we propose combining Indegree into the influence measure presented by Reilly et al. as follows:

$$Ï_i = \frac{N_i}{M_i} * Indegree$$

where

- Ïi is the influence of user *i*
- N_i is the number of times user *i* was retweeted
- M_i is the number of tweets by user *i*
- Indegree is the number of followers of user *i*

The method proposed considers three elements: Tweets, Retweets, and Followers. This yields a final score for each user $Ï_i$. An influential user would be one with a high Retweet to Tweet ratio and a high Follower base. In cases where Reilly et al.'s method yields a tie between users, Indegree could potentially be a tie breaker.

CHAPTER V

SOCIAL NETWORK SIMULATION

In spite of the flexibility of trees and the many different tree applications, trees, by their nature, have one limitation, namely, they can only represent relations of a hierarchical type, such as relations between parent and child. Other relations are only represented indirectly, such as the relation of being a sibling. A generalization of a tree, a graph, is a data structure in which this limitation is lifted. Intuitively, a graph is a collection of vertices (or nodes) and the connections between them. Generally, no restriction is imposed on the number of vertices in the graph or on the number of connections one vertex can have to other vertices [Drozdek 2010].

Also like trees, graphs are useful in a wide spectrum of problems such as computing distances, finding circularities in relationships, and determining connectivity. Graphs are versatile data structures that can represent a large number of different situations and events from diverse domains. Graphs for example are used to identify the shortest distance, or the shortest path, between two locations on a map Graph theory has grown into a sophisticated area of mathematics and computer science in the last 200 years since it was first studied. Many results are of theoretical interest [Mehta 2004].

A graph $G = (V,E)$ consists of a finite set of vertices $V = \{\upsilon1, \upsilon2, . . . , \upsilon n\}$ and a finite set E of edges $E = \{e1, e2, . . . , em\}$ To each edge e there corresponds a pair of vertices (u, υ) which e is said to be incident on.

A directed graph, or a digraph, G = (V, E) consists of a nonempty set V of vertices and a set E of edges (also called arcs), where each edge is a pair of vertices from V [Drozdek 2010]. In a directed graph, each edge had a direction. For each edge (vi,vj), vi is the source node and vj is the terminal node. If reciprocity exists, there would be an edge (vj,vi). Each node v has an Indegree, which accounts for all the nodes (or vertices) the point to v. Each node also has Outdegree, which represents the vertices (or nodes) that v points to. Cha et al. used the terms Indegree and Outdegree to refer to Followers and Following [Cha 2010].

For a given graph a number of different representations are possible. The ease of implementation, as well as the efficiency of a graph algorithm depends on the proper choice of the graph representation. The two most commonly used data structures for representing a graph (directed or undirected) are adjacency lists and adjacency matrix [Mehta 2004].

A. Definition of the Social Network Graph

For this study, we are going to be modeling a directed social network. The network will be represented as a directed graph G = (V,E). The network will consist of multiple vertices V, where each vertex $v_i \in V$ represents a user of the network. Each edge e $(v_i , v_j) \in E$, will represent the relationships between the users, such that v_i, $v_j \in V$, represents user v_i following user v_j. The graph is going to allow us to analyze user relationships, how messages propagate through the network, and user how influence propagates through the network.

B. Properties of Social Network Users

Each vertex, or node, in the graph is going to be assigned several properties. These properties will drive the behavior of the nodes during the simulation runs. Our simulation is based on a probabilistic model and will be configured to run for time steps *c*, a parameter that can be adjusted accordingly. The properties assigned to each vertex will consist of probabilities. Each probability determines if each vertex will mimic a specific behaviors during each time step *c* in the simulation. The properties of each vertex $v_{i,}$ in the network are the following:

- A Followers Generation Probability $P(_{Fvi})$: The probability that a user v_j will follow user v_i
- A Message Generation Probability $P(_{Mvi})$: The probability that a user v_i will post a message during a particular time step *c*
- A Message Forwarding Probability $P(_{Nvi})$: The probability that a user v_j will forward a message created by user v_i during a particular time step *c*

Upon running the simulation, the graph and all nodes will be created and assigned probabilities. The Message Generation Probability and the Message Forwarding Probability are both gathered and computed from the Twitter data sample. This is defined in later in the chapter. The Followers Generation probability is based on a random uniform distribution model. The relationship of all users will be established before the simulation begins running for its set time steps. The edges between the nodes (user relationships) will be recorded.

For every *c* in the simulation, a user may or may not generate a message. This is determined by their $P(_{Mvi})$. For every *c* in the simulation, when a user creates a message, their message may or may not be forwarded by followers. This is determined by their $P(_{Nvi})$. The end results of the simulation will be:

- A user may / may not have created a message at a time step *c*
- A user may / may not have forwarded a message in time step *c*
- If a user created a message at time step *c-1*, that message may / may not be forwarded by the user's followers

C. Simulation of the Social Network Graph

In order to simulate the network and create the directed graph, values have to be assigned to each individual node. As mentioned prior, the values assigned to the nodes are based on a sample of the Twitter social network. Users of the Twitter network engage in behaviors such as tweeting, retweeting, and following users. These behaviors are sampled to generate the properties that are assigned to the users of the simulated network. From the data sample, we can determine characteristics such as message creation rates and message forwarding rates. In our simulation, we will be modeling three different behaviors: the act of following a user, the act of creating a messages and the act of forwarding a message.

i. Simulating Social Network Users

A set number of vertices, or users, will be introduced into the simulated network. Every user will be represented as a node in the graph. The node will contain properties that controls its behavior in the simulation. The simulation will be configured to run for a total of time steps *c*. For every time step *c*, we will record the simulated behaviors of each user:

- User v_i is | is not followed by user v_j
- User v_i does not post | posts a message
- User v_j retweets | does not retweet user v_i

ii. Simulating Followers

Users of the simulated network will be assigned followers based on a random uniform distribution method. Each user will be assigned a probability of being followed, $P(v_{Fi})$. For each user in the simulated network, the probability that a user is followed is determined as follows:

$$v_j \textbf{\textit{ follows }} v_i \textbf{\textit{ IFF }} q \leq P(Fv_i)$$

where

- q is a random value, $0 \leq q < 1$

If the value of q is less than or equal to the probability assigned to user v_i, user v_i gains a follower. As a user gains a follower, an edge is inserted into the digraph.

iii. Simulating Message Creation

From the Twitter data sample, for a time period c, we collect the message count M_i of each user U_i in the set, and the date D_i they joined the network. This data is used to model message creation in the simulation. We identify the highest messaging rate *ɯ* belonging a single user, and the lowest messaging rate *ł* belonging to a single user.

To model message creation, we first determine a set of z user ranges u_j. The value of z is a parameter and can be increased or decreased accordingly. The ranges are created to categorize users based on their messaging rates during time period t. The distance between the ranges will be of equal size r, and the size of r is determined as follows:

$$\mathrm{r} = \frac{[(ɯ) - (ł)]}{z}\text{, where ł } > 0$$

where

- $ɯ$ is the highest messaging rate in the sample data
- $ł$ is the lowest messaging rate in the sample data
- z is the parameter that indicates the number of ranges u_j to be created
- r is the distance of the ranges u_j

We next determine the upper bound value H and lower bound value L of each range u_j as follows:

$$\textbf{when } j = 1 : \quad \begin{array}{c} H(u_j) = r \\ L(u_j) = 1 \text{ or } ł \end{array}$$

$$\textbf{when } 1 < j \leq z : \quad \begin{array}{c} H(u_j) = j * r \\ L(u_j) = (j - 1) * r \end{array}$$

where

- $H(u_j)$ is the upper bound value of range u_j
- $L(u_j)$ is the lower bound value of range u_j
- j represents the count of values in the range set; $1 \leq j \leq z$

One of the u_j range will have its $H(u_j)$ value equal to $ɯ$ and one of the ranges will have its $L(u_j)$ value set to $ł$ or 1, whichever is greater. For each user U_i, we first have to determine how long the user has been a member of the network. The tenure of their account is measured in days. This is calculated as follows:

$$T_i = (y - D_i)$$

where

- T_i is the tenure of U_i, in days
- y is the most recent date of activity in the sampled Twitter data
- D_i is the date user U_i joined the network

For each user U_i, we next determine their messaging rate over the tenure of their account. The messaging rate R_i is calculated as follows:

$$R_i = \frac{M_i}{T_i}$$

where

- R_i = is the rate at which user U_i created messages over time period t
- M_i = the number of messages created by U_i during time period t
- T_i = the tenure of user U_i

Each user U_i in the Twitter data sample is assigned to one of the ranges u_j, based on their messaging rate R_i as follows:

$$U_\text{i} \textbf{\textit{ is assigned to }} u_\text{j} \textbf{\textit{ IFF }} H(u_\text{j}) > R_i \geq L(u_\text{j})$$

where

- U_i is the user in the sample Twitter data
- u_j is the range U_i is assigned to
- R_i is the messaging rate of user U_i
- $H(u_\text{j})$ and $L(u_\text{j})$ are the max and min values of the range u_j

Once all users have been assigned to one of u_j ranges, we count how many users were assigned to each range u_j to derive our messaging probabilities. We calculate the percent of profiles assigned to each range u_j by taking the total number of assigned users W_j to each range u_j, and divide that by the number of users U in the sample Twitter data set as follows:

$$S_j = \frac{W_j}{U}$$

where:

- S_j is a percentage. It is one of the probabilistic values that could be assigned to the nodes of the simulated network.
- W_j is the numbers of users assigned to range u_j
- U is the number of users in the Twitter sample data set

The probability that a node, or user of the simulated network, will create a message during time step c will be determined by S_j. An S_j value will be assigned to each node at random.

iv. Simulating Forwarding of Messages

From the Twitter data sample, for a time period t, we collect the retweet count N_i of each user U_i in the set, and the count of followers F_i of each user. This data is used to model message forwarding in the network. We identify the highest retweet rate ɐ of a single user, and the lowest retweet rate ə of a single user.

To model message forwarding, we create a set of w user ranges h_j. The value of w is a parameter and can be increased or decreased accordingly. The ranges are created to categorize users based on their retweet rates during time period *t*. The distance between the ranges will be of equal size *r*, and the size of *r* is determined as follows:

$$r = \frac{[(\text{ɐ}) - (\text{ə})]}{w}, \text{ where } \text{ə} > 0$$

where

- *r* is the size of the ranges u_j
- *ɐ* is the highest retweet rate in the sample data
- *ə* is the lowest retweet rate in the sample data
- *w* is the parameter that indicates the number of ranges h_j to be created

We next determine the upper bound value *H* and lower bound value *L* of each range h_j as follows:

$$\textbf{when } j = 1 : \quad H(u_j) = r$$
$$L(u_j) = 1 \text{ or } \text{ə}$$

$$\textbf{when } 1 < j \leq w : \quad H(h_j) = j * r$$
$$L(h_j) = (j - 1) * r$$

where

- $H(h_j)$ is the upper bound value of profile u_j
- $L(h_j)$ is the lower bound value of profile u_j
- *j* represents the count of values in the range set; $1 \leq j \leq w$

One of the h_j ranges will have its $H(u_j)$ value equal to ɐ and one of the ranges will have its $L(h_j)$ value set to ɘ or 1, whichever is greater. For each user U_i, we next determine their retweet rate. The retweet rate Q_i is calculated as follows:

$$Q_i = \frac{N_i}{F_i}$$

where

- Q_i = is the rate at which user U_i was retweeted over time period t
- N_i = the count of times U_i was retweeted during time period t
- F_i = the count of followers of user U_i

Each user U_i in the Twitter data sample is assigned to one of the profiles h_j, based on their retweet rate Q_i as follows:

$$U_\mathrm{i} \textbf{\textit{ is assigned to }} h_j \textbf{\textit{ IFF }} H(h_j) > Q_i \geq L(h_j)$$

where

- U_i is the user in the sample Twitter data
- h_j is the user profile U_i is assigned to
- Q_i is the retweet rate of user U_i
- $H(h_j)$ and $L(h_j)$ are the max and min values of the range corresponding to profile $h_{j.}$

Once all users have been assigned to one of h_j ranges, we count how many users were assigned to each profile h_j to derive Message Forwading probabilities. We calculate the percent of profiles assigned to each range h_j by taking the total number of assigned users W_j to each range h_j, and divide that by the number of users U in the sample Twitter data set as follows:

$$K_{\mathrm{j}} = \frac{W_j}{U}$$

where

- K_j is a percentage. It is one of the probabilistic values that could be assigned to the nodes of the simulated network.
- W_j is the numbers of users assigned to profile h_j
- U is the number of users in the Twitter sample data set

In the simulated network, a user is assigned a probability K_j at random. A user must create a message during the time step *c* in order to have their message forwarded. If a user does not create a message during a time step *c*, message forwarding does not occur. When a user does create a message, K_j is the probability that each one of the user's followers will or will not pass the message on.

CHAPTER VI

IMPLEMENTATION

As part of this study, we had mentioned prior that we would be comparing influence rankings each of the three methods yielded. There are five influence rankings to compare: normalized influence, normalized influence with Indegree, Indegree, Followers, Retweets, and Mentions. The first ranking comes from the method proposed by Reilly et al. The second ranking comes from the method proposed by this study. The last three rankings are from the method proposed by Cha et al. In total, there are eleven comparisons to be made.

Spearman's rank correlation coefficient calculation is a nonparametric test; the coefficient assesses how well an arbitrary monotonic function could describe the relationship between two variables, without making any other assumptions about the particular nature of the relationship between the variables. The closer p is to +1 or -1, the stronger the likely correlation. A perfect positive correlation is +1 and a perfect negative correlation is -1 [Cha 2010]. Spearman's rank correlation coefficient calculation is defined as follows:

$$\rho = 1 - \frac{6 \sum (xi - yi)^2}{U^3 - U}$$

where

- ρ = The strength of the association between the two rank sets
- x_i = User ranking of the first category being compared
- y_i = User ranking of the second category being compared
- U = The total number of users

One thing to note is that Cha et al. applied all three rankings over their entire data set. In contrast, the method proposed by Reilly et al. as well as the derived method proposed in this study, limits the study to a set time period. Identifying influential users within a set time period entails limiting the Twitter sample data to a set time period. The time period can vary from a day to a week to a month, at the discretion of the person analyzing for influence. In implementing this study, we apply a set time period to data being sampled and being used to implement all rankings. In addition to a time restriction, we are only considering those users who have posted at least one message, have at least one follower, and have had their message forwarded at least once.

A. Implementation of the Influence Application

The application was written in Java. Java is an open source powerful programming language that is available to any user on any operating system. In addition, Java offers extensive libraries that are beneficial in working with large data sets as did this study. The data is stored in various databases created using MySQL. MySQL is the second most used relational database management system and is also open source. The applications as well as the databases were created in the Linux environment. The data collected from the Twitter API is also stored into MySQL database.

The application is designed to pull data from the Twitter data sample, for a specific time period. The query is hard coded in the application. The query can be edited to modify the time period for which it pulls data. The idea is that the same data sample should undergo all three influence metrics to get a true comparison between the methods.

The application retrieves, for each user: twitter user ID, count of tweets during the specified time frame, count of retweets during the specified time frame, count of followers, and count of mentions. Not all methods use the same data. Reilly et al.'s influence metric uses only three of those variables: the user ID, count of tweets, and count of retweets. Cha et al.'s method uses four of these: user ID, count of retweets, count of followers, and count of mentions. The method derived and presented in this study uses four variables: user ID, count of tweets, count of retweets, and count of followers.

The application consists of twenty-three classes, broken down as follows: three data classes, five sorting classes, three instance classes, eleven compare rank classes, and the main class that takes care of executing all classes.

i. The Main Class

RankUsers.java. This class is responsible for running the entire process from beginning to end.

ii. The Data Classes

A specific data class was created for each of the three methods. The purpose of doing so was to allow not only the collection of data from the Twitter API, but also to compute several of the key calculations as the data was being collected. In the application, these data classes were manipulated with the use of ArrayLists.

ChaInfluenceObj.java. This class contains seven variables, and it supports the implementation of the method presented by Cha et al. The first four variables consist of the data collected from the Twitter Sample. The last three variables are the user rank based on mentions, the user rank based on retweets, and the user rank based on followers.

NormInfluenceObj.java. This class contains six variables, and it supports the implementation of the method presented by Reilly et al. The first three variables consist of the data collected from the Twitter Sample. The last three variables are the ratio of retweets to tweets, the normalized ratio, and the user's final rank.

RTFInfluenceObj.java This class contains seven variables, and it supports the implementation of the derived method presented by this study. The first four variables consist of the data collected from the Twitter Sample. The last three variables are the ratio of retweets to tweets, the ratio of tweets to retweets multiplied by the count of followers, and the user's final rank.

iii. The Sorting Classes

To sort the ArrayLists created for each of the data classes above, we had to implement a few sorting classes specific to the element the list is being sorted by.

SortByFollowers.java , SortByMentions.java, SortByRetweets.java. Cha et al.'s method entails ranking users based on a sheer count of either Followers, Mentions or Retweets. Once the data was collected, to rank a user in each category, we had to sort the ArrayList three times. These sort methods are specifically written for the ChaInfluenceObj class.

SortByNi.java. Reilly et al.'s method sorts users after they've received a Normalized Influence score. This sort method is specific to the NormInfluenceObj.

SortByRTF.java. The method presented in this study sorts users after they've received an influence score based on the retweet to tweet ratio multiplied by the count of followers. This sort method is specific to the RTFInfluenceObj.

iv. The Instance Classes

The Instance Classes are what drive the computation of each method. Each instance class is programmed to collect the data from the Twitter API. It is here that the query can be edited. Once the data is collected, all calculations, sorting, and rankings are carried out. Each instance class, once the rankings are complete, writes the results to a different database. The name of the database where the results are stored is *usersRanked*.

Instance_ChaInfluence.java. This class processes the method proposed by Cha et al. This class writes three different results to the database. Ranks by retweets are written to the table *rank_retweets*. Ranks by mentions are written to the table *rank_mentions*. Ranks by followers are written to the table *rank_followers*.

Instance_NormInfluence.java. This class processes the method proposed by Reilly et al. Ranks by normalized influence are written to the table *rank_ni*.

Instance_RTFInfluence.java. This class processes the method proposed in this study. Ranks by retweet to tweet ratio multiplied by followers are written to the table *rank_rtf*.

v. The Compare Rankings Classes

We use Spearman's Correlation Coefficient formula to compare the ranks produced by the different methods to see if they are closely related. Each class pulls the corresponding rank from the database *usersRanked*. In total, there are eleven comparison's made. Each class writes the comparison results between to rankings to a database called *ranksCompared*. Each class compares two rankings.

RankedObj.java. Spearman's Coefficient Correlation is applied in this class. Each of the compare classes calls for the RankedObj class.

Compare_Ranks_fTOm.java Comparison of followers to mentions as presented by Cha et al. in their method. The results are stored to the table *followers_mentions*.

Compare_Ranks_rTOm.java. Comparison of mentions to retweets as presented by Cha et al. in their method. The results are stored to the table *retweets_mentions*.

Compare_Ranks_rTOf.java. Comparison of followers to retweets as presented by Cha et al. in their method. The results are stored to the table *retweets_followers*.

The following three rank comparisons compare the normalized influence rank proposed by Reilly et al. against the individual ranks yielded by the method proposed by Cha and team.

Compare_Ranks_niTOretweets.java. Comparison of Normalized Influence rank to the rank based on a sheer count of retweets. The results are stored to the table *ni_retweets*.

Compare_Ranks_niTOmentions.java. Comparison of Normalized Influence rank to the rank based on a sheer count of mentions. The results are stored to the table *ni_mentions*.

Compare_Ranks_niTOfollowers.java. Comparison of Normalized Influence rank to the rank based on a sheer count of followers. The results are stored to the table *ni_followers*.

The following three classes compare the ranks yielded by the method proposed by this study, which is referred to as the RTF rank, against the individual ranks yielded by the method proposed by Cha and team.

Compare_Ranks_rtfTOretweets.java. Comparison of RTF Rank to Retweets. The results are stored to the table *rtf_retweets*.

Compare_Ranks_rtfTOmentions.java. Comparison of RTF Rank to Mentions. The results are stored to the table *rtf_mentions*.

Compare_Ranks_rtfTOfollowers.java. Comparison of RTF Rank to Followers. The results are stored to the table *rtf_followers*.

The most important comparison was saved for last. The method proposed in this study is a derivate work based on the influence metric proposed by Reilly et al. Our intuition is that the ranking would be similar in nature, with followers being a tie breaker for those who tied in ranking.

Compare_Ranks_niTOrtf.java. Comparison of Normalized Influence to RTF. The results are stored to the table *ni_rtf*.

vi. The Databases

To prevent run time errors when writing the data to the databases, two database schemas were executed to create the databases. The database schema defines the tables within the database and what the variables in each table consist of.

usersRanked.sql. This is the database that stores all final rankings by each of three methods implemented for this study. The schema of the database is defined in Appendix A.

ranksCompared.sql. This is the database that stores all rank comparisons. The schema of the database is defined in Appendix A.

B. Implementation of the Simulation Application

The simulation application was also written in Java. It is designed to pull data from the Twitter data sample, for a specific time period. The query is hard coded in the application. The query can be edited to modify the time period for which it pulls data. From the Twitter data sample, we compute ranges of probabilities for message creation and message forwarding. This is explained in detail in Chapter 5, Sections C, Subections iii and iv. Once the probabilities are determined, the Twitter data sample is no longer needed.

The application calls for two separate classes. One class is the Nodes class. This class contains elements pertaining to the properties of each node. The second class is the Tweets class.

This class contains elements pertaining to message activity in the simulation. The graph is built using an existing library built for Java called JGraphT. We decided to use this library as it facilitates the implementation of a directed graph, and due to time restrictions on this study, our focus was on creating the directed social network, not necessarily worry about its implementation.

i. The Main Class

TGSim.java. This class is responsible for running the entire process from beginning to end.

ii. The Data Class

Nodes.java. This class serves as a data structure of four elements to be implemented as an ArrayList. The first variable is an integer and it serves to store the unique ID of each node. The last three variables are decimal values, each one a probability: probability of being followed, probability of creating a message, and probability of having a message forwarded by a follower.

Tweets.java. This class serves as a data structure of three elements to be implemented as an ArrayList. The first variable is an integer and it serves to store the unique ID of each message posted in the simulation. The last two variables are also integers: the ID of the author of the message and the time step the message was posted.

CHAPTER VII

CONCLUSION AND FUTURE WORK

The influence method presented in this paper is a derivative method based on prior work. We consider the retweet to tweet ratio as well as the user’s audience (followers) in formulating an influence measure. The Twitter Network has evolved significantly, and it keeps evolving. Retweeting was not a popular convention at the time Boyd et al. did their analysis. Today, new conventions, or methods of interaction, are being introduced into the network. Along with retweeting, users now have the option to ‘Favorite’ a tweet. ‘Favoriting’ a Tweet could perhaps be instrumental in identifying influential users. This paves way for future studies in influence. How do the new communication conventions help identify influence?

Limitations within the Twitter API make it challenging to obtain complete sets of data. To conduct a more detailed study, one would need to be collecting data for a rather long period of time. A much larger data set would allow for additional studies. Reilly et al. for example conducted their study over the time span of one day. With a much larger Sample data set, we could implement their method to rank users on a time span of one day, and do this for all sequential days available in the data sample. The measure of influence would be recorded for each user for each day. It would allow for studies on how influence is established over time.

With our simulation, we were able to create a directed graph to show the relationships between users in our social network. The Twitter data sample provides geographical information about its users. Future studies on the topic could take location into account when analyzing user relationships. The nodes could be placed on an actual map, instead of a generic graph, to give insight as to how online social networks have broken geographical barriers. The top influential users could be mapped along with their followers.

REFERENCES

"About Twitter". Retrieved from http://about.twitter.com/company. April 2015.

A. Drozdek. "Data Structures and Algorithms in Java", 2nd ed. Cengage Learning. 2010.

A. Firat, S. Chatterjee, and M. Yilmaz., "Genetic clustering of social networks using random Walks". *Comput. Stat. Data Anal.* 51, August 2007.

A.R. Pratkanis, "An invitation to social influence research," in The Science of Social Influence: Advances and Future Progress, A. R. Pratkanis, Ed. New York, New York, USA: Psychology Press, 2007, ch. 1, pp. 1–15.

B. Suh, L. Hong, P. Pirolli, and E. H. Chi. "Want to be Retweeted? Large Scale Analytics On Factors Impacting Retweet in Twitter Network". In Proceedings of the 2010 IEEE Second International Conference on Social Computing (SOCIALCOM '10). IEEE Computer Society, Washington, DC, USA, 177-184. 2010.

C. Aggarwal. 2011. Social Network Data Analytics 1st ed. Springer Publishing Company, Incorporated, 2011.

C. Reilly, D.Salinas, D. DeLeon "Ranking Users Based on Influence in a Directional Social Network" in CSCI '14 Proceedings of the 2014 International Conference on Computational Science and Computational Intelligence - Volume 02, Washington, DC, Mar. 2014.

D. Boyd, S. Golder, and G. Lotan, "Tweet, tweet, retweet: Conversational aspects of retweeting on Twitter," in HICSS-43 IEEE: Kauai, Hawaii, 2010.

D. Helbing, Social Self-Organization: Agent-Based Simulations and Experiments to Study Emergent Social Behavior. 1st Edition. Springer-Verlag Berlin Heidelberg. 2012.

D. Liu, Q. Wu, and W. Han, "Measuring Micro-blogging User Influence Based on User-Tweet Interaction Model" in Advances in Swarm Intelligence, Harbin, China, June 2013.

D. Liu , X. Chen: "Rumor Propagation in Online Social Networks Like Twitter -- A Simulation Study", Proceedings of the 2011 Third International Conference on Multimedia Information Networking and Security, p.278-282, November 04-06, 2011

D. Romero, W. Galuba, S. Asur and B. Huberman, "Influence and Passivity in Social Media" in WWW '11 Proceedings of the 20th International Conference Companion on World Wide Web, New York, NY, USA, 2011.

D. P. Mehta and S. Sahni. "Handbook of Data Structures and Applications", Chapman & Hall/Crc Computer and Information Science Series. Chapman & Hall/CRC. 2004.

E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts, "Everyone's an influencer: Quantifying influence on Twitter," in WSDM'11, Hong Kong, China, Feb. 2011.

G. Razis, I. Anagnostopoulos, "InfluenceTracker: Rating the impact of a Twitter account", AIAI 2014 Workshops, IFIP AICT 437, pp. 184–195, 2014.

H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in WWW2010, Raleigh, North Carolina, USA, Mar. 2010.

K. Weller, A. Bruns, J. Burgess, M. Mahrt, and C. Puschmann, "Twitter and Society". New York Peter Lang Publishing, 2014.

M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in Twitter: The million follower fallacy," in Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media, 2010, pp. 10–17.

M. Gatti, A. Appel, C. Nogueira, C. Pinhanez, P. Cavalin, S.Neto: "A simulation-based approach to analyze the information diffusion in Microblogging Online Social Network", Winter Simulation Conference 2013: pp. 1685-1696, 2013.

M. M. Uddin, M. Imran, and H. Sajjad, "Understanding Types of Users on Twitter". In Proc. of Social Com. 2014.

"Twitter API". Retreived from https://dev.twitter.com/rest/public/rate-limiting. April 2015.

W. Yue, H. Yong, H. Xio-Hai, D. Keng, "Impact of user influence on information multi-step communication in a micro-blog∗" in Published Online, Chengdu, China, Aug. 2013.

APPENDIX A

APPENDIX A

INFLUENCE - JAVA CLASSES AND DATABASES

rankUsers.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;

public class RankUsers
{

        public static void main(String args[]) throws SQLException, ClassNotFoundException
        {
                Instance_ChaInfluence runChaInf = new Instance_ChaInfluence();
                Instance_NormInfluence runNormInf = new Instance_NormInfluence();
                Instance_RTFInfluence runRTFInf = new Instance_RTFInfluence();

                Compare_Ranks_niTOrtf run_comp_set1 = new Compare_Ranks_niTOrtf();
                Compare_Ranks_niTOretweets run_comp_set2 = new Compare_Ranks_niTOretweets();
                Compare_Ranks_niTOfollowers run_comp_set3 = new Compare_Ranks_niTOfollowers();
                Compare_Ranks_niTOmentions run_comp_set4 = new Compare_Ranks_niTOmentions();
                Compare_Ranks_rtfTOretweets run_comp_set5 = new Compare_Ranks_rtfTOretweets();
                Compare_Ranks_rtfTOfollowers run_comp_set6 = new Compare_Ranks_rtfTOfollowers();
                Compare_Ranks_rtfTOmentions run_comp_set7 = new Compare_Ranks_rtfTOmentions();
                Compare_Ranks_rTOf run_comp_set8 = new Compare_Ranks_rTOf();
                Compare_Ranks_rTOm run_comp_set9 = new Compare_Ranks_rTOm();
                Compare_Ranks_fTOm run_comp_set10 = new Compare_Ranks_fTOm();
                RunSim generateData = new RunSim();

                System.out.println("Generating data...");
                generateData.start();
                System.out.println("Generating data complete.");

        }

}
```

ChaInfluenceObj.java

```
// ChaInfluenceObj.java
// This is the definition of the data object/structure. It contains
// Twitter information relevant to computing user influence
// according to how Cha et al. propose in their study.
//
// The structure contains seven elements:
// 1. username                * collected from SQL query
// 2. count of mentions       * collected from SQL query
// 3. count of retweets       * collected from SQL query
// 4. count of followers      * collected from SQL query
// 5. rank based on mentions  * calculation
// 6. rank based on retweets  * calculation
// 7. rank based on followers * calculation
//
//
// Written by Jose Villarreal Jr
// Spring 2015 - Univeristy of Texas Pan American
//

public class ChaInfluenceObj
{

        // The following seven variables correspond to the
        // elements that form part of the object.

                private String userName;

                private int countOfRetweets;
                private int countOfFollowers;
                private int countOfMentions;

                private int rankingMentions;
                private int rankingRetweets;
                private int rankingFollowers;

        // Default class constructor.

                public ChaInfluenceObj()
                {
                        userName = "";
                        countOfMentions = 0;
                        countOfRetweets = 0;
                        countOfFollowers = 0;

                        rankingMentions = 0;
                        rankingRetweets = 0;
                        rankingFollowers = 0;

                }


                public ChaInfluenceObj(String abc, int rCount, int fCount, int mCount)
                {
                        userName = abc;
                        countOfRetweets = rCount;
                        countOfFollowers = fCount;
                        countOfMentions = mCount;
                }



        // The following void functions are used
        // to append data to the object.
        //
        // setMentionRank - to insert the rank based on mentions to the object.
        // setRetweetRank - to insert the rank based on retweets to the object.
        // setFollowersRank - to insert the rank based on followers to the object.


                public void setMentionRank(int mRank)
                {
                        this.rankingMentions = mRank;
                }

                public void setRetweetRank(int rtRank)
                {
                        this.rankingRetweets = rtRank;
                }
```

```
		public void setFollowerRank(int fRank)
		{
			this.rankingFollowers = fRank;
		}


	// The following four functions are used
	// to collect data from the Data Obj to write to final output file.
	// getRankingMentions()  retrieves user ranking based on mentions.
	// getRankingRetweets()  retrieves user ranking based on retweets.
	// getRankingFollowers() retrieves user ranking based on followers.
	// getUserName() returns userName.
	//
	// getCountOfMentions()  is used to sort in class -> SortByMentions.java
	// getCountOfRetweets()  is used to sort in class -> SortByRetweets.java
	// getCountOfFollowers() is used to sort in class -> SortByFollowers.java


		public String getUserName()
		{
			return userName;
		}

		public int getRankingMentions()
		{
			return rankingMentions;
		}

		public int getRankingRetweets()
		{
			return rankingRetweets;
		}

		public int getRankingFollowers()
		{
			return rankingFollowers;
		}

		public int getCountOfMentions()
		{
			return countOfMentions;
		}

		public int getCountOfRetweets()
		{
			return countOfRetweets;
		}

		public int getCountOfFollowers()
		{
			return countOfFollowers;
		}


} // end of class definition
```

NormInfluenceObj.java

```
// NormInfluenceObj.java
// This is the definition of the data object/structure. It contains
// Twitter information relevant to computing user influence
// according to how Reilly et al. propose in their study.
//
// The structure contains six elements:
// 1. username                * collected from SQL query
// 2. count of tweets         * collected from SQL query
// 3. count of retweets               * collected from SQL query
// 4. ratio (retweets / tweets)       * calculation
// 5. normalized ratio        * calculation
// 6. rank score              * calculation
//
//
// Written by Jose Villarreal Jr
// Spring 2015 - Univeristy of Texas Pan American
//


public class NormInfluenceObj
{

        // The following six variables correspond to the
        // elements that form part of the object.

                private String userName;
                private int countOfTweets;
                private int countOfRetweets;
                private double ratioOfRetweetsTweets;
                private double normalizedRatio;
                private int ranking;


        // Default class constructor.

                public NormInfluenceObj()
                {
                        userName = "";
                        countOfTweets = 0;
                        countOfRetweets = 0;
                        ratioOfRetweetsTweets = 0.0;
                        normalizedRatio = 0.0;
                        ranking = 0;
                }


                public NormInfluenceObj(String abc, int tCount, int rCount, double rtc)
                {
                        userName = abc;
                        countOfTweets = tCount;
                        countOfRetweets = rCount;
                        ratioOfRetweetsTweets = rtc;
                }


        // The following void functions are used
        // to append data to the object.
        //
        // setNormalizedRatio - to insert the normalizedRatio into the object
        // setFinalRank - to insert ranking into the object

                public void setNormalizedRatio(double calculatedRatio)
                {
                        this.normalizedRatio = calculatedRatio;
                }

                public void setFinalRank(int ranked)
                {
                        this.ranking = ranked;
                }


        // The following two functions are used
        // to collect data from the obj to write to file.
        // getRanking() returns ranking
        // getUserName() returns userName
        // getNormalizedRatio() is used to sort the list - SortByNI.java

                public int getRanking()
                {
                        return ranking;
                }

                public String getUserName()
                {
                        return userName;
                }

                public double getNormalizedRatio()
                {
                        return normalizedRatio;
                }

                public double getRatioOfRetweetsTweets()
                {
                        return ratioOfRetweetsTweets;
                }

} // end of class definition
```

RTFInfluenceObj.java

```
// RTFInfluenceObj.java
// This is the definition of the data object/structure. It contains
// Twitter information relevant to computing user influence
// according to how JV proposed in his study.
//
// The structure contains seven elements:
// 1. username               * collected from SQL query
// 2. count of tweets        * collected from SQL query
// 3. count of retweets              * collected from SQL query
// 4. ratio (retweets / tweets)      * calculation
// 5. count of followers     * collected from SQL query
// 6. ratio * followers              * calculation
// 7. rank                   * calculation
//
// Written by Jose Villarreal Jr
// Spring 2015 - Univeristy of Texas Pan American
//


public class RTFInfluenceObj
{

        // The following seven variables correspond to the
        // elements that form part of the object.

                private String userName;
                private int countOfTweets;
                private int countOfRetweets;
                private double ratioOfRetweetsTweets;
                private int countOfFollowers;
                private double ratioRTF;
                private int ranking;

        // Default class constructor.

                public RTFInfluenceObj()
                {
                        userName = "";
                        countOfTweets = 0;
                        countOfRetweets = 0;
                        ratioOfRetweetsTweets = 0.0;
                        countOfFollowers = 0;
                        ratioRTF = 0.0;
                        ranking = 0;
                }
```

```
		public RTFInfluenceObj(String abc, int tCount, int rCount, double rtc, int fCount, 
double rtf)
		{
			userName = abc;
			countOfTweets = tCount;
			countOfRetweets = rCount;
			ratioOfRetweetsTweets = rtc;
			countOfFollowers = fCount;
			ratioRTF = rtf;
		}


	// The following void functions are used
	// to append data to the object.
	//
	// setNormalizedRatio - to insert the normalizedRatio into the object
	// setFinalRank - to insert ranking into the object


		public void setFinalRank(int ranked)
		{
			this.ranking = ranked;
		}



	// The following two functions are used
	// to collect data from the obj to write to file.
	// getRanking() returns ranking
	// getUserName() returns userName
	// getNormalizedRatio() is used to sort the list - SortByNI.java


		public int getRanking()
		{
			return ranking;
		}

		public String getUserName()
		{
			return userName;
		}

		public double getRTFRatio()
		{
			return ratioRTF;
		}


} // end public class RTFInfluenceObj
```

SortByFollowers.java

```
import java.util.*;
public class SortByFollowers implements Comparator<ChaInfluenceObj>
{

	private ArrayList<ChaInfluenceObj> sce;

	public SortByFollowers()
	{
		sce = new ArrayList<ChaInfluenceObj>();
	}




	@Override
```

```
        public int compare(ChaInfluenceObj set1, ChaInfluenceObj set2)
        {
                return Double.compare(set2.getCountOfFollowers(), set1.getCountOfFollowers());
        }


}
```

SortByMentions.java

```
import java.util.*;

public class SortByMentions implements Comparator<ChaInfluenceObj>
{

        private ArrayList<ChaInfluenceObj> sce;

        public SortByMentions()
        {
                sce = new ArrayList<ChaInfluenceObj>();
        }

        @Override
        public int compare(ChaInfluenceObj set1, ChaInfluenceObj set2)
        {

                return Double.compare(set2.getCountOfMentions(), set1.getCountOfMentions());
        }

}
```

SortByRetweets.java

```
import java.util.*;

public class SortByRetweets implements Comparator<ChaInfluenceObj>
{

        private ArrayList<ChaInfluenceObj> sce;

        public SortByRetweets()
        {
                sce = new ArrayList<ChaInfluenceObj>();
        }

        @Override
        public int compare(ChaInfluenceObj set1, ChaInfluenceObj set2)
        {

                return Double.compare(set2.getCountOfRetweets(), set1.getCountOfRetweets());
        }

}
```

SortByNI.java

```
import java.util.*;

public class SortByNI implements Comparator<NormInfluenceObj>
{

	private ArrayList<NormInfluenceObj> ni2;

	public SortByNI()
	{
		ni2 = new ArrayList<NormInfluenceObj>();
	}

	@Override
	public int compare(NormInfluenceObj set1, NormInfluenceObj set2)
	{

		return Double.compare(set2.getNormalizedRatio(), set1.getNormalizedRatio());
	}

}
```

SortByRTF.java

```
import java.util.*;

public class SortByRTF implements Comparator<RTFInfluenceObj>
{

	private ArrayList<RTFInfluenceObj> rtf1;

	public SortByRTF()
	{
		rtf1 = new ArrayList<RTFInfluenceObj>();
	}

	@Override
	public int compare(RTFInfluenceObj set1, RTFInfluenceObj set2)
	{

		return Double.compare(set2.getRTFRatio(), set1.getRTFRatio());
	}

}
```

Instance_ChaInfluence.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;


public class Instance_ChaInfluence
{

	// Global variables of the class

	private static final String url = "jdbc:mysql://localhost:3306/tweetSim";
	private static final String urlOutput = "jdbc:mysql://localhost:3306/usersRanked";
	private static final String user = "root";
	private static final String password = "root";
	private static final String myQuery = "SELECT userName, retweets, followers, mentions
	FROM users;";
```

```
private static Connection myConnection;
private static Statement myStatement;
private static ResultSet myResultSet;
private static PreparedStatement myPStatement_DROP;
private static PreparedStatement myPStatement_CREATE;
private static PreparedStatement myPStatement_WRITE;

private static ArrayList<ChaInfluenceObj> ci = new ArrayList<>();



// Main function
public static void start() throws SQLException, ClassNotFoundException
{

               connectToJDBC();
               collectFromJDBC();

               Collections.sort(ci, new SortByRetweets());
               rankUsersByRetweets();
               produceOutputFileRetweets();
               writeToJDBC_retweets();

               Collections.sort(ci, new SortByFollowers());
               rankUsersByFollowers();
               produceOutputFileFollowers();
               writeToJDBC_followers();

               Collections.sort(ci, new SortByMentions());
               rankUsersByMentions();
               produceOutputFileMentions();
               writeToJDBC_mentions();


} // end public static void main(String args[]) throws SQLException,
ClassNotFoundException


// connectToJDBC()
// Function to establish connection to the database.
// ********************************************************************
public static void connectToJDBC()
{
       try
       {
               myConnection = DriverManager.getConnection(url, user, password);
               myStatement = myConnection.createStatement();
               myResultSet = myStatement.executeQuery(myQuery);
       }

       catch (SQLException e)
       {
               e.printStackTrace();
       }


       catch (Exception e)
       {
               e.printStackTrace();
       }
} // end public static void connectToJDBC()
// ********************************************************************






// collectFromJDBC()
// Function to collect data from database if connection is successful.
// ********************************************************************
```

```
public static void collectFromJDBC()
{
        String  uName = "";
        int     uRetweets = 0;
        int     uFollowers = 0;
        int     uMentions = 0;


        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("userName");
                        uRetweets = myResultSet.getInt("retweets");
                        uFollowers = myResultSet.getInt("followers");
                        uMentions = myResultSet.getInt("mentions");

                        if (uRetweets != 0 && uFollowers != 0 && uMentions != 0){
                                ci.add(new
                        ChaInfluenceObj(uName,uRetweets,uFollowers,uMentions));}
                }
        myConnection.close();
        }



        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *********************************************************************



// *********************************************************************
public static void rankUsersByRetweets()
{

        int userRank = 0;
        int currentRetweets = 0;
        int previousRetweets = 0;

        for(ChaInfluenceObj p : ci)
        {
                currentRetweets = p.getCountOfRetweets();

                if (currentRetweets != previousRetweets)
                        {
                                userRank = userRank + 1;
                        }
                p.setRetweetRank(userRank);
                previousRetweets = p.getCountOfRetweets();

        }
} // end public static void rankUsersByRetweets()
// *********************************************************************



// *********************************************************************
public static void rankUsersByFollowers()
```

```
{

        int userRank = 0;
        int currentFollowers = 0;
        int previousFollowers = 0;

        for(ChaInfluenceObj p : ci)
        {
                currentFollowers = p.getCountOfFollowers();

                if (currentFollowers != previousFollowers)
                        {
                                userRank = userRank + 1;
                        }
                p.setFollowerRank(userRank);
                previousFollowers = p.getCountOfFollowers();

        }
} // end public static void rankUsersByFollowers()
// *********************************************************************



// *********************************************************************
public static void rankUsersByMentions()
{

        int userRank = 0;
        int currentMentions = 0;
        int previousMentions = 0;

        for(ChaInfluenceObj p : ci)
        {
                currentMentions = p.getCountOfMentions();

                if (currentMentions != previousMentions)
                        {
                                userRank = userRank + 1;
                        }
                p.setMentionRank(userRank);
                previousMentions = p.getCountOfMentions();

        }
} // end public static void rankUsersByFollowers()
// *********************************************************************
```

```
// The file contains two elements - user name and rank.
// *********************************************************************
public static void produceOutputFileRetweets()
```

```
{
        File file = new File("/home/jv/Twitter/users_ranked_by_retweets.txt");

        if (!file.exists()){
                try
                {
                        file.createNewFile();}
                        catch (IOException fg)
                        {
                                fg.printStackTrace();
                                System.exit(-1);
                        }
                }
        try
        {
                FileWriter fw = new FileWriter(file.getAbsoluteFile());
                BufferedWriter bw = new BufferedWriter(fw);

                for(ChaInfluenceObj p : ci)
                {
                        bw.write(p.getUserName() + " " + p.getRankingRetweets() + "\n");
                }

                bw.close();
        }

        catch (Exception e)
        {
                e.printStackTrace();
        }
} // end public static void produceOutputFileRetweets()
// *********************************************************************


// The file contains two elements - user name and rank.
// *********************************************************************
public static void produceOutputFileFollowers()
{
        File file = new File("/home/jv/Twitter/users_ranked_by_followers.txt");

        if (!file.exists()){
                try
                {
                        file.createNewFile();}
                        catch (IOException fg)
                        {
                                fg.printStackTrace();
                                System.exit(-1);
                        }
                }
        try
        {
                FileWriter fw = new FileWriter(file.getAbsoluteFile());
                BufferedWriter bw = new BufferedWriter(fw);

                for(ChaInfluenceObj p : ci)
                {
                        bw.write(p.getUserName() + " " + p.getRankingFollowers() + "\n");
                }
                bw.close();
        }

        catch (Exception e)
        {
                e.printStackTrace();
        }
} // end public static void produceOutputFileFollowers()
// *********************************************************************
// The file contains two elements - user name and rank.
// *********************************************************************
public static void produceOutputFileMentions()
```

```
{
        File file = new File("/home/jv/Twitter/users_ranked_by_mentions.txt");

        if (!file.exists()){
                try
                {
                        file.createNewFile();}
                        catch (IOException fg)
                        {
                                fg.printStackTrace();
                                System.exit(-1);
                        }
                }
        try
        {
                FileWriter fw = new FileWriter(file.getAbsoluteFile());
                BufferedWriter bw = new BufferedWriter(fw);

                for(ChaInfluenceObj p : ci)
                {
                        bw.write(p.getUserName() + " " + p.getRankingMentions() + "\n");
                }

                bw.close();
        }

        catch (Exception e)
        {
                e.printStackTrace();
        }
} // end public static void produceOutputFileMentions()
// *********************************************************************



// Write Rank results to Database usersRanked.
// *********************************************************************
public static void writeToJDBC_retweets()
{

        try
        {
                myConnection = DriverManager.getConnection(urlOutput, user, password);
                myStatement = myConnection.createStatement();
                String queryRetweets = "";

                myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
                rank_retweets;");
                myPStatement_DROP.execute();

                myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
                rank_retweets (userName VARCHAR(30), ranking INTEGER, PRIMARY
                KEY(userName));");
                myPStatement_CREATE.execute();


                for(ChaInfluenceObj p : ci)
                {
                        queryRetweets = "INSERT INTO rank_retweets (userName, ranking)
                        VALUES" + "( '" + p.getUserName() + "'," + p.getRankingRetweets() +
                        ");";
                        myPStatement_WRITE = myConnection.prepareStatement(queryRetweets);
                        myPStatement_WRITE.execute();
                }

        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// ******************************************************************


// Write Rank results to Database.
// There is a database called usersRanked. The database contains five tables.
// One of the tables is called rank_followers.
// It writes to DB : 1. userName 2. ranking
// ******************************************************************
public static void writeToJDBC_followers()
{

        try
        {
                myConnection = DriverManager.getConnection(urlOutput, user, password);
                myStatement = myConnection.createStatement();
                String queryRetweets = "";

                myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
                rank_followers;");
                myPStatement_DROP.execute();

                myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
                rank_followers (userName VARCHAR(30), ranking INTEGER, PRIMARY
                KEY(userName));");
                myPStatement_CREATE.execute();


                for(ChaInfluenceObj p : ci)
                {
                        queryRetweets = "INSERT INTO rank_followers (userName, ranking)
                        VALUES" + "( '" + p.getUserName() + "'," + p.getRankingFollowers()
                        + ");";
                        myPStatement_WRITE = myConnection.prepareStatement(queryRetweets);
                        myPStatement_WRITE.execute();
                }

        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// ******************************************************************







// Write Rank results to Database.
// There is a database called usersRanked. The database contains five tables.
// One of the tables is called rank_mentions.
```

```
        // It writes to DB : 1. userName 2. ranking
        // *********************************************************************
        public static void writeToJDBC_mentions()
        {

                try
                {
                        myConnection = DriverManager.getConnection(urlOutput, user, password);
                        myStatement = myConnection.createStatement();
                        String queryRetweets = "";

                        myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
                        rank_mentions;");
                        myPStatement_DROP.execute();

                        myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
                        rank_mentions (userName VARCHAR(30), ranking INTEGER, PRIMARY
                        KEY(userName));");
                        myPStatement_CREATE.execute();


                        for(ChaInfluenceObj p : ci)
                        {
                                queryRetweets = "INSERT INTO rank_mentions (userName, ranking)
                                VALUES" + "( '" + p.getUserName() + "'," + p.getRankingMentions() +
                                ");";
                                myPStatement_WRITE = myConnection.prepareStatement(queryRetweets);
                                myPStatement_WRITE.execute();
                        }

                }

                catch (SQLException e)
                {
                        e.printStackTrace();
                }


                catch (Exception e)
                {
                        e.printStackTrace();
                }

        } // end public static void collectFromJDBC()
        // *********************************************************************


} // end public class Instance_ChaInfluence
```

Instance_NormInfluence.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;


public class Instance_NormInfluence
{

        // Global variables of the class

        private static final String url = "jdbc:mysql://localhost:3306/tweetSim";
        private static final String urlOutput = "jdbc:mysql://localhost:3306/usersRanked";
        private static final String user = "root";
private static final String password = "root";
private static final String myQuery = "SELECT userName, tweets, retweets FROM users;";

private static Connection myConnection;
private static Statement myStatement;
private static ResultSet myResultSet;
private static PreparedStatement myPStatement_DROP;
private static PreparedStatement myPStatement_CREATE;
private static PreparedStatement myPStatement_WRITE;

private static ArrayList<NormInfluenceObj> ni = new ArrayList<>();
private static double sumAllRatios = 0.0000;


// Main function
public static void start() throws SQLException, ClassNotFoundException
{

               connectToJDBC();
               collectFromJDBC();
               calculateNormalizedInfluence();
               Collections.sort(ni, new SortByNI());
               rankUserSet();
               produceOutputFile();
               writeToJDBC_ni();


} // end public static void main(String args[]) throws SQLException,
ClassNotFoundException



// connectToJDBC()
// Function to establish connection to the database.
// *****************************************************************
public static void connectToJDBC()
{
       try
       {
               myConnection = DriverManager.getConnection(url, user, password);
               myStatement = myConnection.createStatement();
               myResultSet = myStatement.executeQuery(myQuery);
       }

       catch (SQLException e)
       {
               e.printStackTrace();
       }


       catch (Exception e)
       {
               e.printStackTrace();
       }
} // end public static void connectToJDBC()
// *****************************************************************









// collectFromJDBC()
// Function to collect data from database if connection is successful.
//
```

```
// The database returns three elements: (these elements are stored to variables)
// UserID of the Twitter user    -> stored to uName
// Count of Tweets by the user   -> stored to uTweets
// Count of Retweets of the user -> stored to uRetweets
//
// A fourth element - stored to variable uRatio is used to calculate
// the Retweet to Tweet ratio of each user.
// If Count of Tweets or Count of Retweets are equal to 0
// the uRatio is set to 0.
//
//
// As the Retweet/Tweet ratio of each user is calculated
// the function is also keeping a running sum of all ratios
// being stored to sumAllRatios.
// *****************************************************************
// *****************************************************************
public static void collectFromJDBC()
{
        String  uName = "";
        int     uTweets = 0;
        int     uRetweets = 0;
        double  uRatio = 0.0000;

        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("userName");
                        uTweets = myResultSet.getInt("tweets");
                        uRetweets = myResultSet.getInt("retweets");

                                if (uTweets == 0 || uRetweets == 0)
                                        {
                                                uRatio = 0.00;
                                        }
                                else
                                        {
                                                uRatio = (double)uRetweets / uTweets;
                                        }


                        sumAllRatios = sumAllRatios + uRatio;

                        // create instance of object
                        if (uRatio != 0.00) {
                                ni.add(new
                                NormInfluenceObj(uName,uTweets,uRetweets,uRatio));
                        }
                }
        myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *****************************************************************


// calculateNormalizedInfluence()
// In this loop, the individual Retweet to Tweet Ratio of each user
// is divided by the variable SumAllRatios.
// This is the normalizedInfluence.
// This is added to the data object.
// *******************************************************************

public static void calculateNormalizedInfluence()
{
        for(NormInfluenceObj p : ni)
        {
                p.setNormalizedRatio(p.getRatioOfRetweetsTweets()/ sumAllRatios);
        }
} // end public static void calculateNormalizedInfluence()
// *******************************************************************



// rankUserSet()
// This function ranks the sorted ArrayList of the NormInfluenceObj.
// The list is sorted based on Normalized Influence prior to being ranked.
//
// userRank is initialized to 0
// currentRatio & previousRatio are also initialized to 0
//
// In the first iteration of the for loop:
//      currentRatio = set to current user's Normalized Ratio
//      previousRatio = 0 and DOES NOT MATCH currentRatio
//      the first user on the list is ranked #1
//      previousRatio = set to current user's Normalized Ratio
//
// In cases where currentRatio = previousRatio = users tie for the same rank
//
// *******************************************************************
public static void rankUserSet()
{

        int userRank = 0;
        double currentRatio = 0.00;
        double previousRatio = 0.00;

        for(NormInfluenceObj p : ni)
        {

                currentRatio = p.getNormalizedRatio();

                if (currentRatio != previousRatio)
                        {
                                userRank = userRank + 1;
                        }
                p.setFinalRank(userRank);
                previousRatio = p.getNormalizedRatio();

        }
} // end public static void rankUserSet()
// *******************************************************************
```

```
// produceOutputFile()
// This function takes care of creating a text file that contains
// the final rankings.
// The file contains two elements - user name and rank.
// *********************************************************************
public static void produceOutputFile()
{
        File file = new File("/home/jv/Twitter/users_ranked_by_ni.txt");

        if (!file.exists()){
                try
                {
                        file.createNewFile();}
                        catch (IOException fg)
                        {
                                fg.printStackTrace();
                                System.exit(-1);
                        }
                }
        try
        {
                FileWriter fw = new FileWriter(file.getAbsoluteFile());
                BufferedWriter bw = new BufferedWriter(fw);

                for(NormInfluenceObj p : ni)
                {
                        bw.write(p.getUserName() + " " + p.getRanking() + "\n");
                }

                bw.close();
        }

        catch (Exception e)
        {
                e.printStackTrace();
        }
} // end public static void produceOutputFile()
// *********************************************************************


// Write Rank results to Database.
// There is a database called usersRanked. The database contains five tables.
// One of the tables is called rank_mentions.
// It writes to DB : 1. userName 2. ranking
// *********************************************************************
public static void writeToJDBC_ni()
{

        try
        {
                myConnection = DriverManager.getConnection(urlOutput, user, password);
                myStatement = myConnection.createStatement();
                String queryRetweets = "";

                myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
                rank_ni;");
                myPStatement_DROP.execute();

                myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE rank_ni
                (userName VARCHAR(30), ranking INTEGER, PRIMARY KEY(userName));");
                myPStatement_CREATE.execute();


                for(NormInfluenceObj p : ni)
                {
                        queryRetweets = "INSERT INTO rank_ni (userName, ranking) VALUES" +
                        "( '" + p.getUserName() + "'," + p.getRanking() + ");";
                        myPStatement_WRITE = myConnection.prepareStatement(queryRetweets);

                        myPStatement_WRITE.execute();
                }

        }
```

```
		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// *********************************************************************


} // end of class definition
```

Instance_RTFInfluence.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;


public class Instance_RTFInfluence
{

	// Global variables of the class

	private static final String url = "jdbc:mysql://localhost:3306/tweetSim";
	private static final String urlOutput = "jdbc:mysql://localhost:3306/usersRanked";
	private static final String user = "root";
	private static final String password = "root";
	private static final String myQuery = "SELECT userName, tweets, retweets, followers FROM
	users;";

	private static Connection myConnection;
	private static Statement myStatement;
	private static ResultSet myResultSet;
	private static PreparedStatement myPStatement_DROP;
	private static PreparedStatement myPStatement_CREATE;
	private static PreparedStatement myPStatement_WRITE;

	private static ArrayList<RTFInfluenceObj> rtf = new ArrayList<>();



	// Main function
	public static void start() throws SQLException, ClassNotFoundException
	{

			connectToJDBC();
			collectFromJDBC();
			Collections.sort(rtf, new SortByRTF());
			rankUserSet();
			produceOutputFile();
			writeToJDBC_rtf();


	} // end public static void main(String args[]) throws SQLException,
	ClassNotFoundException


	// connectToJDBC()
	// Function to establish connection to the database.
	// *********************************************************************
```

```
public static void connectToJDBC()
{
        try
        {
                myConnection = DriverManager.getConnection(url, user, password);
                myStatement = myConnection.createStatement();
                myResultSet = myStatement.executeQuery(myQuery);
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }
} // end public static void connectToJDBC()
// ******************************************************************



// collectFromJDBC()
// Function to collect data from database if connection is successful.
//
// The database returns three elements: (these elements are stored to variables)
// UserID of the Twitter user    -> stored to uName
// Count of Tweets by the user   -> stored to uTweets
// Count of Retweets of the user -> stored to uRetweets
//
// A fourth element - stored to variable uRatio is used to calculate
// the Retweet to Tweet ratio of each user.
// If Count of Tweets or Count of Retweets are equal to 0
// the uRatio is set to 0.
//
//
// As the Retweet/Tweet ratio of each user is calculated
// the function is also keeping a running sum of all ratios
// being stored to sumAllRatios.
// ******************************************************************
// ******************************************************************
public static void collectFromJDBC()
{
        String  uName = "";
        int     uTweets = 0;
        int     uRetweets = 0;
        double  uRatio = 0.0;
        int     uFollowers = 0;
        double   uRTF = 0.0;

        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("userName");
                        uTweets = myResultSet.getInt("tweets");
                        uRetweets = myResultSet.getInt("retweets");
                        uFollowers = myResultSet.getInt("followers");

                                if (uTweets == 0 || uRetweets == 0)
                                        {
                                                uRatio = 0.00;
                                        }
                                else
                                        {
                                                uRatio = (double)uRetweets / uTweets;
                                        }
                        uRTF = uRatio * uFollowers;


                        // create instance of object
                        if (uRatio != 0.00 && uFollowers != 0) {
                                rtf.add(new RTFInfluenceObj(uName,uTweets,uRetweets,uRatio,
                                uFollowers, uRTF));
                        }

                }
        myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *********************************************************************



// rankUserSet()
// This function ranks the sorted ArrayList of the RTFInfluenceObj.
// The list is sorted based on [ (Retweet/Tweet Ratio) * Followers ] score.
//
// userRank is initialized to 0
// currentRatio & previousRatio are also initialized to 0
//
// In the first iteration of the for loop:
//      currentRatio = set to current user's Normalized Ratio
//      previousRatio = 0 and DOES NOT MATCH currentRatio
//      the first user on the list is ranked #1
//      previousRatio = set to current user's Normalized Ratio
//
// In cases where currentRatio = previousRatio = users tie for the same rank
//
// *********************************************************************
public static void rankUserSet()
{

        int userRank = 0;
        double currentRatio = 0.00;
        double previousRatio = 0.00;

        for(RTFInfluenceObj p : rtf)
        {

                currentRatio = p.getRTFRatio();

                if (currentRatio != previousRatio)
                        {
                                userRank = userRank + 1;
                        }
                p.setFinalRank(userRank);
                previousRatio = p.getRTFRatio();

        }
} // end public static void rankUserSet()
// *********************************************************************


// produceOutputFile()
// This function takes care of creating a text file that contains
// the final rankings.
// The file contains two elements - user name and rank.
// ****************************************************************
public static void produceOutputFile()
{
	File file = new File("/home/jv/Twitter/users_ranked_by_rtf.txt");

	if (!file.exists()){
		try
		{
			file.createNewFile();}
			catch (IOException fg)
			{
				fg.printStackTrace();
				System.exit(-1);
			}
		}
	try
	{
		FileWriter fw = new FileWriter(file.getAbsoluteFile());
		BufferedWriter bw = new BufferedWriter(fw);

		for(RTFInfluenceObj p : rtf)
		{
			bw.write(p.getUserName() + " " + p.getRanking() + "\n");
		}

		bw.close();
	}

	catch (Exception e)
	{
		e.printStackTrace();
	}
} // end public static void produceOutputFile()
// ****************************************************************

// Write Rank results to Database.
// There is a database called usersRanked. The database contains five tables.
// One of the tables is called rank_mentions.
// It writes to DB : 1. userName 2. ranking
// ****************************************************************
public static void writeToJDBC_rtf()
{

	try
	{
		myConnection = DriverManager.getConnection(urlOutput, user, password);
		myStatement = myConnection.createStatement();
		String queryRetweets = "";

		myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
		rank_rtf;");
		myPStatement_DROP.execute();

		myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE rank_rtf
		(userName VARCHAR(30), ranking INTEGER, PRIMARY KEY(userName));");
		myPStatement_CREATE.execute();


		for(RTFInfluenceObj p : rtf)
		{
			queryRetweets = "INSERT INTO rank_rtf (userName, ranking) VALUES" +
		"( '" + p.getUserName() + "'," + p.getRanking() + ");";
			myPStatement_WRITE = myConnection.prepareStatement(queryRetweets);
			myPStatement_WRITE.execute();
		}

	}

	catch (SQLException e)
```

```
                {
                        e.printStackTrace();
                }


                catch (Exception e)
                {
                        e.printStackTrace();
                }

        } // end public static void collectFromJDBC()
        // ********************************************************************


} // end of class definition
```

RankedObj.java

```
public class RankedObj
{
        private String userName;
        private int rank1;
        private int rank2;
        private double score;


        public RankedObj()
        {
                userName = "";
                score = 0.00;
        }

        public RankedObj(String abc, int aRank, int bRank)
        {
                userName = abc;
                rank1 = aRank;
                rank2 = bRank;
        }

        public String getUserName()
        {
                return userName;
        }

        public int getRank1()
        {
                return rank1;
        }

        public int getRank2()
        {
                return rank2;
        }

        public double getScore()
        {
                return score;
        }

        public void setScore(double aScore)
        {
                this.score = aScore;
        }

}
```

Compare_Ranks_fTOm.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_fTOm
{

	private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
	private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
	private static final String user = "root";
	private static final String password = "root";
	private static final String myQuery = "SELECT rank_followers.userName,
	rank_followers.ranking, rank_mentions.ranking FROM rank_followers, rank_mentions WHERE
	rank_followers.userName = rank_mentions.userName;";


	private static Connection myConnection;
	private static Statement myStatement;
	private static ResultSet myResultSet;
	private static PreparedStatement myPStatement_DROP;
	private static PreparedStatement myPStatement_CREATE;
	private static PreparedStatement myPStatement_WRITE;

	private static ArrayList<RankedObj> rObj = new ArrayList<>();


	public static void start() throws SQLException, ClassNotFoundException
	{
			connectToJDBC();
			collectFromJDBC();
			compareRankings();
			writeToJDBC();
	}



	// connectToJDBC()
	// Function to establish connection to the database.
	// *********************************************************************
	public static void connectToJDBC()
	{
		try
		{
			myConnection = DriverManager.getConnection(url, user, password);
			myStatement = myConnection.createStatement();
			myResultSet = myStatement.executeQuery(myQuery);
		}

		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}
	} // end public static void connectToJDBC()
	// *********************************************************************



	// *********************************************************************
	public static void collectFromJDBC()
	{
		String  uName = "";
```

```
        int     uRank1 = 0;
        int     uRank2 = 0;

        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("rank_followers.userName");
                        uRank1 = myResultSet.getInt("rank_followers.ranking");
                        uRank2 = myResultSet.getInt("rank_mentions.ranking");

                        rObj.add(new RankedObj(uName,uRank1,uRank2));

                }

                myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *******************************************************************


// *******************************************************************
public static void compareRankings()
{

        double numer = 0.00;
        double denom = 0.00;
        int totalUsers = rObj.size();

        int uRank1 = 0;
        int uRank2 = 0;

        for(RankedObj p : rObj)
        {

                uRank1 = p.getRank1();
                uRank2 = p.getRank2();

                numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
                denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


                p.setScore(1 - (numer / denom));
        }


} // end public static void calculateNormalizedInfluence()
// *******************************************************************







// *******************************************************************
public static void writeToJDBC()
{
		try
		{
			myConnection = DriverManager.getConnection(urlOutput, user, password);
			myStatement = myConnection.createStatement();
			String myQuery = "";

			myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
			followers_mentions;");
			myPStatement_DROP.execute();

			myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
			followers_mentions (userName VARCHAR(30), score DOUBLE, PRIMARY
			KEY(userName));");
			myPStatement_CREATE.execute();


			for(RankedObj p : rObj)
			{
				myQuery = "INSERT INTO followers_mentions (userName, score) VALUES"
				+ "( '" + p.getUserName() + "'," + p.getScore() + ");";
				myPStatement_WRITE = myConnection.prepareStatement(myQuery);
				myPStatement_WRITE.execute();
			}

			myConnection.close();
		}

		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// ********************************************************************


} // end public class Compare_Ranks
```

Compare_Ranks_rTOm.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_rTOm
{

	private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
	private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
	private static final String user = "root";
	private static final String password = "root";
	private static final String myQuery = "SELECT rank_retweets.userName,
	rank_retweets.ranking, rank_mentions.ranking FROM rank_retweets, rank_mentions WHERE
	rank_retweets.userName = rank_mentions.userName;";


	private static Connection myConnection;
private static Statement myStatement;
private static ResultSet myResultSet;
private static PreparedStatement myPStatement_DROP;
private static PreparedStatement myPStatement_CREATE;
private static PreparedStatement myPStatement_WRITE;

private static ArrayList<RankedObj> rObj = new ArrayList<>();

public static void start() throws SQLException, ClassNotFoundException
{
               connectToJDBC();
               collectFromJDBC();
               compareRankings();
               writeToJDBC();
}


// connectToJDBC()
// Function to establish connection to the database.
// *******************************************************************
public static void connectToJDBC()
{
       try
       {
               myConnection = DriverManager.getConnection(url, user, password);
               myStatement = myConnection.createStatement();
               myResultSet = myStatement.executeQuery(myQuery);
       }

       catch (SQLException e)
       {
               e.printStackTrace();
       }


       catch (Exception e)
       {
               e.printStackTrace();
       }
} // end public static void connectToJDBC()
// *******************************************************************


// *******************************************************************
public static void collectFromJDBC()
{
       String  uName = "";
       int     uRank1 = 0;
       int     uRank2 = 0;

       try
       {

               while (myResultSet.next())
               {
                       uName = myResultSet.getString("rank_retweets.userName");
                       uRank1 = myResultSet.getInt("rank_retweets.ranking");
                       uRank2 = myResultSet.getInt("rank_mentions.ranking");

                       rObj.add(new RankedObj(uName,uRank1,uRank2));

               }

               myConnection.close();
       }



       catch (SQLException e)
       {
               e.printStackTrace();
```

```
	}


	catch (Exception e)
	{
		e.printStackTrace();
	}

} // end public static void collectFromJDBC()
// *****************************************************************


// *****************************************************************
public static void compareRankings()
{

	double numer = 0.00;
	double denom = 0.00;
	int totalUsers = rObj.size();

	int uRank1 = 0;
	int uRank2 = 0;

	for(RankedObj p : rObj)
	{

		uRank1 = p.getRank1();
		uRank2 = p.getRank2();

		numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
		denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


		p.setScore(1 - (numer / denom));
	}


} // end public static void calculateNormalizedInfluence()
// *****************************************************************


// *****************************************************************
public static void writeToJDBC()
{
	try
	{
		myConnection = DriverManager.getConnection(urlOutput, user, password);
		myStatement = myConnection.createStatement();
		String myQuery = "";

		myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
		retweets_mentions;");

		myPStatement_DROP.execute();

		myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
		retweets_mentions (userName VARCHAR(30), score DOUBLE, PRIMARY
		KEY(userName));");

		myPStatement_CREATE.execute();

		for(RankedObj p : rObj)
		{
			myQuery = "INSERT INTO retweets_mentions (userName, score) VALUES"
			+ "( '" + p.getUserName() + "'," + p.getScore() + ");";
			myPStatement_WRITE = myConnection.prepareStatement(myQuery);
			myPStatement_WRITE.execute();
		}

		myConnection.close();
	}
```

```
		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// ********************************************************************

} // end public class Compare_Ranks
```

Compare_Ranks_rTOf.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_rTOf
{

	private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
	private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
	private static final String user = "root";
	private static final String password = "root";
	private static final String myQuery = "SELECT rank_retweets.userName,
	rank_retweets.ranking, rank_followers.ranking FROM rank_retweets, rank_followers WHERE
	rank_retweets.userName = rank_followers.userName;";


	private static Connection myConnection;
	private static Statement myStatement;
	private static ResultSet myResultSet;
	private static PreparedStatement myPStatement_DROP;
	private static PreparedStatement myPStatement_CREATE;
	private static PreparedStatement myPStatement_WRITE;

	private static ArrayList<RankedObj> rObj = new ArrayList<>();

	public static void start() throws SQLException, ClassNotFoundException
	{
			connectToJDBC();
			collectFromJDBC();
			compareRankings();
			writeToJDBC();
	}


	// connectToJDBC()
	// Function to establish connection to the database.
	// ********************************************************************
	public static void connectToJDBC()
	{
		try
		{
			myConnection = DriverManager.getConnection(url, user, password);
			myStatement = myConnection.createStatement();
			myResultSet = myStatement.executeQuery(myQuery);
		}
```

```
        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }
} // end public static void connectToJDBC()
// *******************************************************************



// *******************************************************************
public static void collectFromJDBC()
{
        String  uName = "";
        int     uRank1 = 0;
        int     uRank2 = 0;

        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("rank_retweets.userName");
                        uRank1 = myResultSet.getInt("rank_retweets.ranking");
                        uRank2 = myResultSet.getInt("rank_followers.ranking");

                        rObj.add(new RankedObj(uName,uRank1,uRank2));

                }

                myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *******************************************************************




// *******************************************************************
public static void compareRankings()
{

        double numer = 0.00;
        double denom = 0.00;
        int totalUsers = rObj.size();

        int uRank1 = 0;
        int uRank2 = 0;

        for(RankedObj p : rObj)
        {

                uRank1 = p.getRank1();
```

```
                    uRank2 = p.getRank2();

                    numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
                    denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


                    p.setScore(1 - (numer / denom));
            }


    } // end public static void calculateNormalizedInfluence()
    // *********************************************************************


    // *********************************************************************
    public static void writeToJDBC()
    {

            try
            {
                    myConnection = DriverManager.getConnection(urlOutput, user, password);
                    myStatement = myConnection.createStatement();
                    String myQuery = "";

                    myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
                    retweets_followers;");

                    myPStatement_DROP.execute();

                    myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
                    retweets_followers (userName VARCHAR(30), score DOUBLE, PRIMARY
                    KEY(userName));");

                    myPStatement_CREATE.execute();


                    for(RankedObj p : rObj)
                    {
                            myQuery = "INSERT INTO retweets_followers (userName, score) VALUES"
                            + "( '" + p.getUserName() + "'," + p.getScore() + ");";
                            myPStatement_WRITE = myConnection.prepareStatement(myQuery);
                            myPStatement_WRITE.execute();
                    }

                    myConnection.close();
            }

            catch (SQLException e)
            {
                    e.printStackTrace();
            }


            catch (Exception e)
            {
                    e.printStackTrace();
            }

    } // end public static void collectFromJDBC()
    // *********************************************************************

} // end public class Compare_Ranks
```

Compare_Ranks_niTOretweets.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_niTOretweets
{

        private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
        private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
        private static final String user = "root";
        private static final String password = "root";
        private static final String myQuery = "SELECT rank_ni.userName, rank_ni.ranking,
        rank_retweets.ranking FROM rank_ni, rank_retweets WHERE rank_ni.userName =
        rank_retweets.userName;";


        private static Connection myConnection;
        private static Statement myStatement;
        private static ResultSet myResultSet;
        private static PreparedStatement myPStatement_DROP;
        private static PreparedStatement myPStatement_CREATE;
        private static PreparedStatement myPStatement_WRITE;

        private static ArrayList<RankedObj> rObj = new ArrayList<>();

        public static void start() throws SQLException, ClassNotFoundException
        {
                        connectToJDBC();
                        collectFromJDBC();
                        compareRankings();
                        writeToJDBC();
        }


        // connectToJDBC()
        // Function to establish connection to the database.
        // ****************************************************************
        public static void connectToJDBC()
        {
                try
                {
                        myConnection = DriverManager.getConnection(url, user, password);
                        myStatement = myConnection.createStatement();
                        myResultSet = myStatement.executeQuery(myQuery);
                }

                catch (SQLException e)
                {
                        e.printStackTrace();
                }


                catch (Exception e)
                {
                        e.printStackTrace();
                }
        } // end public static void connectToJDBC()
        // ****************************************************************






        // ****************************************************************
        public static void collectFromJDBC()
        {
```

```
		String	uName = "";
		int	uRank1 = 0;
		int	uRank2 = 0;

		try
		{

			while (myResultSet.next())
			{
				uName = myResultSet.getString("rank_ni.userName");
				uRank1 = myResultSet.getInt("rank_ni.ranking");
				uRank2 = myResultSet.getInt("rank_retweets.ranking");

				rObj.add(new RankedObj(uName,uRank1,uRank2));

			}

			myConnection.close();
		}

		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// ******************************************************************




	// ******************************************************************
	public static void compareRankings()
	{

		double numer = 0.00;
		double denom = 0.00;
		int totalUsers = rObj.size();

		int uRank1 = 0;
		int uRank2 = 0;

		for(RankedObj p : rObj)
		{

			uRank1 = p.getRank1();
			uRank2 = p.getRank2();

			numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
			denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


			p.setScore(1 - (numer / denom));
		}


	} // end public static void calculateNormalizedInfluence()
	// ******************************************************************




	// ******************************************************************
	public static void writeToJDBC()
	{
		try
		{
			myConnection = DriverManager.getConnection(urlOutput, user, password);
			myStatement = myConnection.createStatement();
			String myQuery = "";

			myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
			ni_retweets;");
			myPStatement_DROP.execute();

			myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
			ni_retweets (userName VARCHAR(30), score DOUBLE, PRIMARY
			KEY(userName));");

			myPStatement_CREATE.execute();


			for(RankedObj p : rObj)
			{
				myQuery = "INSERT INTO ni_retweets (userName, score) VALUES" + "(
				'" + p.getUserName() + "'," + p.getScore() + ");";
				myPStatement_WRITE = myConnection.prepareStatement(myQuery);
				myPStatement_WRITE.execute();
			}

			myConnection.close();
		}

		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// ********************************************************************

} // end public class Compare_Ranks
```

Compare_Ranks_niTOmentions.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_niTOmentions
{

	private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
	private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
	private static final String user = "root";
	private static final String password = "root";
	private static final String myQuery = "SELECT rank_ni.userName, rank_ni.ranking,
	rank_mentions.ranking FROM rank_ni, rank_mentions WHERE rank_ni.userName =
	rank_mentions.userName;";


	private static Connection myConnection;
private static Statement myStatement;
private static ResultSet myResultSet;
private static PreparedStatement myPStatement_DROP;
private static PreparedStatement myPStatement_CREATE;
private static PreparedStatement myPStatement_WRITE;

private static ArrayList<RankedObj> rObj = new ArrayList<>();

public static void start() throws SQLException, ClassNotFoundException
{
               connectToJDBC();
               collectFromJDBC();
               compareRankings();
               writeToJDBC();
}



// connectToJDBC()
// Function to establish connection to the database.
// *******************************************************************
public static void connectToJDBC()
{
       try
       {
               myConnection = DriverManager.getConnection(url, user, password);
               myStatement = myConnection.createStatement();
               myResultSet = myStatement.executeQuery(myQuery);
       }

       catch (SQLException e)
       {
               e.printStackTrace();
       }


       catch (Exception e)
       {
               e.printStackTrace();
       }
} // end public static void connectToJDBC()
// *******************************************************************



// *******************************************************************
public static void collectFromJDBC()
{
       String  uName = "";
       int     uRank1 = 0;
       int     uRank2 = 0;

       try
       {

               while (myResultSet.next())
               {
                       uName = myResultSet.getString("rank_ni.userName");
                       uRank1 = myResultSet.getInt("rank_ni.ranking");
                       uRank2 = myResultSet.getInt("rank_mentions.ranking");

                       rObj.add(new RankedObj(uName,uRank1,uRank2));

               }

               myConnection.close();
       }

       catch (SQLException e)
       {
               e.printStackTrace();
```

```
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *****************************************************************


// *****************************************************************
public static void compareRankings()
{

        double numer = 0.00;
        double denom = 0.00;
        int totalUsers = rObj.size();

        int uRank1 = 0;
        int uRank2 = 0;

        for(RankedObj p : rObj)
        {

                uRank1 = p.getRank1();
                uRank2 = p.getRank2();

                numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
                denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


                p.setScore(1 - (numer / denom));
        }


} // end public static void calculateNormalizedInfluence()
// *****************************************************************


// *****************************************************************
public static void writeToJDBC()
{

        try
        {
                myConnection = DriverManager.getConnection(urlOutput, user, password);
                myStatement = myConnection.createStatement();
                String myQuery = "";

                myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
                ni_mentions;");

                myPStatement_DROP.execute();

                myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
                ni_mentions (userName VARCHAR(30), score DOUBLE, PRIMARY
                KEY(userName));");

                myPStatement_CREATE.execute();


                for(RankedObj p : rObj)
                {
                        myQuery = "INSERT INTO ni_mentions (userName, score) VALUES" + "(
                        '" + p.getUserName() + "'," + p.getScore() + ");";
                        myPStatement_WRITE = myConnection.prepareStatement(myQuery);
                        myPStatement_WRITE.execute();
                }
                myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

    } // end public static void collectFromJDBC()
    // ********************************************************************

} // end public class Compare_Ranks
```

Compare_Ranks_niTOfollowers.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_niTOfollowers
{

    private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
    private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
    private static final String user = "root";
    private static final String password = "root";
    private static final String myQuery = "SELECT rank_ni.userName, rank_ni.ranking,
    rank_followers.ranking FROM rank_ni, rank_followers WHERE rank_ni.userName =
    rank_followers.userName;";


    private static Connection myConnection;
    private static Statement myStatement;
    private static ResultSet myResultSet;
    private static PreparedStatement myPStatement_DROP;
    private static PreparedStatement myPStatement_CREATE;
    private static PreparedStatement myPStatement_WRITE;

    private static ArrayList<RankedObj> rObj = new ArrayList<>();

    public static void start() throws SQLException, ClassNotFoundException
    {
                connectToJDBC();
                collectFromJDBC();
                compareRankings();
                writeToJDBC();
    }


    // connectToJDBC()
    // Function to establish connection to the database.
    // ********************************************************************
    public static void connectToJDBC()
    {
        try
        {
                myConnection = DriverManager.getConnection(url, user, password);
                myStatement = myConnection.createStatement();
                myResultSet = myStatement.executeQuery(myQuery);
        }
        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }
} // end public static void connectToJDBC()
// *******************************************************************


// *******************************************************************
public static void collectFromJDBC()
{
        String  uName = "";
        int     uRank1 = 0;
        int     uRank2 = 0;

        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("rank_ni.userName");
                        uRank1 = myResultSet.getInt("rank_ni.ranking");
                        uRank2 = myResultSet.getInt("rank_followers.ranking");

                        rObj.add(new RankedObj(uName,uRank1,uRank2));

                }

                myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *******************************************************************


// *******************************************************************
public static void compareRankings()
{

        double numer = 0.00;
        double denom = 0.00;
        int totalUsers = rObj.size();

        int uRank1 = 0;
        int uRank2 = 0;

        for(RankedObj p : rObj)
        {

                uRank1 = p.getRank1();
                uRank2 = p.getRank2();

                numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
```

```
                        denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


                        p.setScore(1 - (numer / denom));
                }


        } // end public static void calculateNormalizedInfluence()
        // *******************************************************************



        // *******************************************************************
        public static void writeToJDBC()
        {

                try
                {
                        myConnection = DriverManager.getConnection(urlOutput, user, password);
                        myStatement = myConnection.createStatement();
                        String myQuery = "";

                        myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
                        ni_followers;");

                        myPStatement_DROP.execute();

                        myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
                        ni_followers (userName VARCHAR(30), score DOUBLE, PRIMARY
                        KEY(userName));");

                        myPStatement_CREATE.execute();


                        for(RankedObj p : rObj)
                        {
                                myQuery = "INSERT INTO ni_followers (userName, score) VALUES" + "(
                                '" + p.getUserName() + "'," + p.getScore() + ");";
                                myPStatement_WRITE = myConnection.prepareStatement(myQuery);
                                myPStatement_WRITE.execute();
                        }

                        myConnection.close();
                }

                catch (SQLException e)
                {
                        e.printStackTrace();
                }


                catch (Exception e)
                {
                        e.printStackTrace();
                }

        } // end public static void collectFromJDBC()
        // *******************************************************************

} // end public class Compare_Ranks
```

Compare_Ranks_rtfTOretweets.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_rtfTOretweets
{

        private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
        private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
        private static final String user = "root";
        private static final String password = "root";
        private static final String myQuery = "SELECT rank_rtf.userName, rank_rtf.ranking,
        rank_retweets.ranking FROM rank_rtf, rank_retweets WHERE rank_rtf.userName =
        rank_retweets.userName;";


        private static Connection myConnection;
        private static Statement myStatement;
        private static ResultSet myResultSet;
        private static PreparedStatement myPStatement_DROP;
        private static PreparedStatement myPStatement_CREATE;
        private static PreparedStatement myPStatement_WRITE;

        private static ArrayList<RankedObj> rObj = new ArrayList<>();

        public static void start() throws SQLException, ClassNotFoundException
        {
                        connectToJDBC();
                        collectFromJDBC();
                        compareRankings();
                        writeToJDBC();
        }


        // connectToJDBC()
        // Function to establish connection to the database.
        // ******************************************************************
        public static void connectToJDBC()
        {
                try
                {
                        myConnection = DriverManager.getConnection(url, user, password);
                        myStatement = myConnection.createStatement();
                        myResultSet = myStatement.executeQuery(myQuery);
                }

                catch (SQLException e)
                {
                        e.printStackTrace();
                }


                catch (Exception e)
                {
                        e.printStackTrace();
                }
        } // end public static void connectToJDBC()
        // ******************************************************************






        // ******************************************************************
        public static void collectFromJDBC()
        {
```

```
        String  uName = "";
        int     uRank1 = 0;
        int     uRank2 = 0;

        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("rank_rtf.userName");
                        uRank1 = myResultSet.getInt("rank_rtf.ranking");
                        uRank2 = myResultSet.getInt("rank_retweets.ranking");

                        rObj.add(new RankedObj(uName,uRank1,uRank2));

                }

                myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// ******************************************************************



// ******************************************************************
public static void compareRankings()
{

        double numer = 0.00;
        double denom = 0.00;
        int totalUsers = rObj.size();

        int uRank1 = 0;
        int uRank2 = 0;

        for(RankedObj p : rObj)
        {

                uRank1 = p.getRank1();
                uRank2 = p.getRank2();

                numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
                denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


                p.setScore(1 - (numer / denom));
        }


} // end public static void calculateNormalizedInfluence()
// ******************************************************************





// ******************************************************************
public static void writeToJDBC()
{
		try
		{
			myConnection = DriverManager.getConnection(urlOutput, user, password);
			myStatement = myConnection.createStatement();
			String myQuery = "";

			myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
			rtf_retweets;");

			myPStatement_DROP.execute();

			myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
			rtf_retweets (userName VARCHAR(30), score DOUBLE, PRIMARY
			KEY(userName));");

			myPStatement_CREATE.execute();


			for(RankedObj p : rObj)
			{
				myQuery = "INSERT INTO rtf_retweets (userName, score) VALUES" + "(
				'" + p.getUserName() + "'," + p.getScore() + ");";
				myPStatement_WRITE = myConnection.prepareStatement(myQuery);
				myPStatement_WRITE.execute();
			}

			myConnection.close();
		}

		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// *********************************************************************

} // end public class Compare_Ranks
```

Compare_Ranks_rtfTOmentions.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_rtfTOmentions
{

	private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
	private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
	private static final String user = "root";
	private static final String password = "root";
	private static final String myQuery = "SELECT rank_rtf.userName, rank_rtf.ranking,
	rank_mentions.ranking FROM rank_rtf, rank_mentions WHERE rank_rtf.userName =
	rank_mentions.userName;";
```

```
private static Connection myConnection;
private static Statement myStatement;
private static ResultSet myResultSet;
private static PreparedStatement myPStatement_DROP;
private static PreparedStatement myPStatement_CREATE;
private static PreparedStatement myPStatement_WRITE;

private static ArrayList<RankedObj> rObj = new ArrayList<>();

public static void start() throws SQLException, ClassNotFoundException
{
               connectToJDBC();
               collectFromJDBC();
               compareRankings();
               writeToJDBC();
}


// connectToJDBC()
// Function to establish connection to the database.
// *******************************************************************
public static void connectToJDBC()
{
       try
       {
               myConnection = DriverManager.getConnection(url, user, password);
               myStatement = myConnection.createStatement();
               myResultSet = myStatement.executeQuery(myQuery);
       }

       catch (SQLException e)
       {
               e.printStackTrace();
       }


       catch (Exception e)
       {
               e.printStackTrace();
       }
} // end public static void connectToJDBC()
// *******************************************************************


// *******************************************************************
public static void collectFromJDBC()
{
       String  uName = "";
       int     uRank1 = 0;
       int     uRank2 = 0;

       try
       {

               while (myResultSet.next())
               {
                       uName = myResultSet.getString("rank_rtf.userName");
                       uRank1 = myResultSet.getInt("rank_rtf.ranking");
                       uRank2 = myResultSet.getInt("rank_mentions.ranking");

                       rObj.add(new RankedObj(uName,uRank1,uRank2));

               }

               myConnection.close();
       }

       catch (SQLException e)
       {
               e.printStackTrace();
       }
```

```
        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *****************************************************************


// *****************************************************************
public static void compareRankings()
{

        double numer = 0.00;
        double denom = 0.00;
        int totalUsers = rObj.size();

        int uRank1 = 0;
        int uRank2 = 0;

        for(RankedObj p : rObj)
        {

                uRank1 = p.getRank1();
                uRank2 = p.getRank2();

                numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
                denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


                p.setScore(1 - (numer / denom));
        }


} // end public static void calculateNormalizedInfluence()
// *****************************************************************


// *****************************************************************
public static void writeToJDBC()
{

        try
        {
                myConnection = DriverManager.getConnection(urlOutput, user, password);
                myStatement = myConnection.createStatement();
                String myQuery = "";

                myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
                rtf_mentions;");

                myPStatement_DROP.execute();

                myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
                rtf_mentions (userName VARCHAR(30), score DOUBLE, PRIMARY
                KEY(userName));");

                myPStatement_CREATE.execute();


                for(RankedObj p : rObj)
                {
                        myQuery = "INSERT INTO rtf_mentions (userName, score) VALUES" + "(
                        '" + p.getUserName() + "'," + p.getScore() + ");";
                        myPStatement_WRITE = myConnection.prepareStatement(myQuery);
                        myPStatement_WRITE.execute();
                }

                myConnection.close();
```

```
		}

		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// *********************************************************************

} // end public class Compare_Ranks
```

Compare_Ranks_rtfTOfollowers.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_rtfTOfollowers
{

	private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
	private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
	private static final String user = "root";
	private static final String password = "root";
	private static final String myQuery = "SELECT rank_rtf.userName, rank_rtf.ranking,
	rank_followers.ranking FROM rank_rtf, rank_followers WHERE rank_rtf.userName =
	rank_followers.userName;";


	private static Connection myConnection;
	private static Statement myStatement;
	private static ResultSet myResultSet;
	private static PreparedStatement myPStatement_DROP;
	private static PreparedStatement myPStatement_CREATE;
	private static PreparedStatement myPStatement_WRITE;

	private static ArrayList<RankedObj> rObj = new ArrayList<>();

	public static void start() throws SQLException, ClassNotFoundException
	{
			connectToJDBC();
			collectFromJDBC();
			compareRankings();
			writeToJDBC();
	}







	// connectToJDBC()
	// Function to establish connection to the database.
	// *********************************************************************
	public static void connectToJDBC()
	{
        try
        {
                myConnection = DriverManager.getConnection(url, user, password);
                myStatement = myConnection.createStatement();
                myResultSet = myStatement.executeQuery(myQuery);
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }
} // end public static void connectToJDBC()
// ******************************************************************


// ******************************************************************
public static void collectFromJDBC()
{
        String  uName = "";
        int     uRank1 = 0;
        int     uRank2 = 0;

        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("rank_rtf.userName");
                        uRank1 = myResultSet.getInt("rank_rtf.ranking");
                        uRank2 = myResultSet.getInt("rank_followers.ranking");

                        rObj.add(new RankedObj(uName,uRank1,uRank2));

                }

                myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// ******************************************************************



// ******************************************************************
public static void compareRankings()
{

        double numer = 0.00;
        double denom = 0.00;
        int totalUsers = rObj.size();

        int uRank1 = 0;
        int uRank2 = 0;
```

```
		for(RankedObj p : rObj)
		{

			uRank1 = p.getRank1();
			uRank2 = p.getRank2();

			numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
			denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


			p.setScore(1 - (numer / denom));
		}


	} // end public static void calculateNormalizedInfluence()
	// *********************************************************************



	// *********************************************************************
	public static void writeToJDBC()
	{

		try
		{
			myConnection = DriverManager.getConnection(urlOutput, user, password);
			myStatement = myConnection.createStatement();
			String myQuery = "";

			myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
			rtf_followers;");

			myPStatement_DROP.execute();

			myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE
			rtf_followers (userName VARCHAR(30), score DOUBLE, PRIMARY
			KEY(userName));");

			myPStatement_CREATE.execute();


			for(RankedObj p : rObj)
			{
				myQuery = "INSERT INTO rtf_followers (userName, score) VALUES" + "(
				'" + p.getUserName() + "'," + p.getScore() + ");";
				myPStatement_WRITE = myConnection.prepareStatement(myQuery);
				myPStatement_WRITE.execute();
			}

			myConnection.close();
		}

		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// *********************************************************************

} // end public class Compare_Ranks
```

Compare_Ranks_niTOrtf.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;

public class Compare_Ranks_niTOrtf
{

        private static final String url = "jdbc:mysql://localhost:3306/usersRanked";
        private static final String urlOutput = "jdbc:mysql://localhost:3306/ranksCompared";
        private static final String user = "root";
        private static final String password = "root";
        private static final String myQuery = "SELECT rank_ni.userName, rank_ni.ranking,
        rank_rtf.ranking FROM rank_ni, rank_rtf WHERE rank_ni.userName = rank_rtf.userName;";


        private static Connection myConnection;
        private static Statement myStatement;
        private static ResultSet myResultSet;
        private static PreparedStatement myPStatement_DROP;
        private static PreparedStatement myPStatement_CREATE;
        private static PreparedStatement myPStatement_WRITE;

        private static ArrayList<RankedObj> rObj = new ArrayList<>();

        public static void start() throws SQLException, ClassNotFoundException
        {
                        connectToJDBC();
                        collectFromJDBC();
                        compareRankings();
                        writeToJDBC();
        }



        // connectToJDBC()
        // Function to establish connection to the database.
        // ******************************************************************
        public static void connectToJDBC()
        {
                try
                {
                        myConnection = DriverManager.getConnection(url, user, password);
                        myStatement = myConnection.createStatement();
                        myResultSet = myStatement.executeQuery(myQuery);
                }

                catch (SQLException e)
                {
                        e.printStackTrace();
                }


                catch (Exception e)
                {
                        e.printStackTrace();
                }
        } // end public static void connectToJDBC()
        // ******************************************************************


        // ******************************************************************
        public static void collectFromJDBC()
        {
                String  uName = "";
        int     uRank1 = 0;
        int     uRank2 = 0;

        try
        {

                while (myResultSet.next())
                {
                        uName = myResultSet.getString("rank_ni.userName");
                        uRank1 = myResultSet.getInt("rank_ni.ranking");
                        uRank2 = myResultSet.getInt("rank_rtf.ranking");

                        rObj.add(new RankedObj(uName,uRank1,uRank2));

                }

                myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

} // end public static void collectFromJDBC()
// *********************************************************************


// *********************************************************************
public static void compareRankings()
{

        double numer = 0.00;
        double denom = 0.00;
        int totalUsers = rObj.size();

        int uRank1 = 0;
        int uRank2 = 0;

        for(RankedObj p : rObj)
        {

                uRank1 = p.getRank1();
                uRank2 = p.getRank2();

                numer = 6 * ((uRank1 - uRank2) * (uRank1 - uRank2));
                denom = (totalUsers * totalUsers * totalUsers) - totalUsers;


                p.setScore(1 - (numer / denom));
        }


} // end public static void calculateNormalizedInfluence()
// *********************************************************************






// *********************************************************************
public static void writeToJDBC()
{
		try
		{
			myConnection = DriverManager.getConnection(urlOutput, user, password);
			myStatement = myConnection.createStatement();
			String myQuery = "";

			myPStatement_DROP = myConnection.prepareStatement("DROP TABLE IF EXISTS
			ni_rtf;");

			myPStatement_DROP.execute();

			myPStatement_CREATE = myConnection.prepareStatement("CREATE TABLE ni_rtf
			(userName VARCHAR(30), score DOUBLE, PRIMARY KEY(userName));");

			myPStatement_CREATE.execute();


			for(RankedObj p : rObj)
			{
				myQuery = "INSERT INTO ni_rtf (userName, score) VALUES" + "( '" +
				p.getUserName() + "'," + p.getScore() + ");";
				myPStatement_WRITE = myConnection.prepareStatement(myQuery);
				myPStatement_WRITE.execute();
			}

			myConnection.close();
		}

		catch (SQLException e)
		{
			e.printStackTrace();
		}


		catch (Exception e)
		{
			e.printStackTrace();
		}

	} // end public static void collectFromJDBC()
	// *********************************************************************
} // end public class Compare_Ranks
```

usersRanked.sql

```
DROP SCHEMA IF EXISTS usersRanked;

CREATE SCHEMA usersRanked;

USE usersRanked;


DROP TABLE IF EXISTS rank_ni;

CREATE TABLE rank_ni (
       userName       VARCHAR(30),
       ranking        INTEGER,
       PRIMARY KEY(userName)
);



DROP TABLE IF EXISTS rank_rtf;

CREATE TABLE rank_rtf (
       userName       VARCHAR(30),
       ranking        INTEGER,
       PRIMARY KEY(userName)
);



DROP TABLE IF EXISTS rank_retweets;

CREATE TABLE rank_retweets (
       userName       VARCHAR(30),
       ranking        INTEGER,
       PRIMARY KEY(userName)
);



DROP TABLE IF EXISTS rank_followers;

CREATE TABLE rank_followers (
       userName       VARCHAR(30),
       ranking        INTEGER,
       PRIMARY KEY(userName)
);



DROP TABLE IF EXISTS rank_mentions;

CREATE TABLE rank_mentions (
       userName       VARCHAR(30),
       ranking        INTEGER,
       PRIMARY KEY(userName)
);
```

ranksCompared.sql

```
DROP SCHEMA IF EXISTS ranksCompared;
```

```
CREATE SCHEMA ranksCompared;

USE ranksCompared;

DROP TABLE IF EXISTS ni_rtf;
CREATE TABLE ni_rtf (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);

DROP TABLE IF EXISTS ni_retweets;
CREATE TABLE ni_retweets (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);

DROP TABLE IF EXISTS ni_followers;
CREATE TABLE ni_followers (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);

DROP TABLE IF EXISTS ni_mentions;
CREATE TABLE ni_mentions (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);

DROP TABLE IF EXISTS rtf_retweets;
CREATE TABLE rtf_retweets (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);

DROP TABLE IF EXISTS rtf_followers;
CREATE TABLE rtf_followers (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);

DROP TABLE IF EXISTS rtf_mentions;
CREATE TABLE rtf_mentions (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);


DROP TABLE IF EXISTS retweets_followers;
CREATE TABLE retweets_followers (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);

DROP TABLE IF EXISTS retweets_mentions;
CREATE TABLE retweets_mentions (
       userName       VARCHAR(30),
       score          DOUBLE,
       PRIMARY KEY(userName)
);

DROP TABLE IF EXISTS followers_mentions;
CREATE TABLE followers_mentions (
       userName        VARCHAR(30),
       score           DOUBLE,
       PRIMARY KEY(userName)
);
```

APPENDIX B

APPENDIX B

SIMULATiON - JAVA CLASSES

TGsim.java

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.*;
import java.io.*;
import org.jgrapht.alg.*;
import org.jgrapht.*;
import org.jgrapht.graph.*;
import java.text.*;

public class TGsim
{



        /*  Variables used to pull data sample from Twitter API /JDBC Connection  */

                private static final String url = "jdbc:mysql://localhost:3306/twitter";
                private static final String user = "root";
                private static final String password = "root";

                private static final String myQuery = "SELECT userIndex, tweets, retweets,
                followers," + "tenure FROM user WHERE tweets > 1;";

                private static Connection myConnection;
                private static Statement myStatement;
                private static ResultSet myResultSet;




        /* Variables used to calculate Tweet rates from sample data*/

                private static double highest_TweetRate = 0.0;
                private static double lowest_TweetRate = 0.0;


                private static int tweet_profiles = 10;

                private static List<Double> allTweetRates = new ArrayList<>();

                // tweet_probabilities[i][4] contains the probablity assigned to a user to
                // determine whether or not a user creates a message

                private static double[][] tweet_probabilities = new double[tweet_profiles][4];
```

```
/* Variables used to calculate Retweet rates from sample data*/

        private static double highest_RetweetRate = 0.0;
        private static double lowest_RetweetRate = 0.0;

        private static int retweet_profiles = 10;

        private static List<Double> allRetweetRates = new ArrayList<>();

        // retweet_probabilities[i][4] contains the probablity assigned to a user to
        // determine whether or not a user retweets a message
                private static double[][] retweet_probabilities = new
                double[retweet_profiles][4];





/* Variables used to create the nodes of the Simulated Network*/


        private static int nodes = 50;
        private static int timeSteps = 50;

        // This array store the properties of each node.
                private static ArrayList<Nodes> SIMusers = new ArrayList<>();


        // This array stores every message created during the simulation.
                private static ArrayList<Tweets> SIMtweets = new ArrayList<>();


        // This array stores every message forwarded during the simulation.
                private static ArrayList<Retweets> SIMretweets = new ArrayList<>();

        // This is the graph
        private static DirectedGraph<Nodes, DefaultEdge> directedGraph =
                new DefaultDirectedGraph<Nodes, DefaultEdge>
                (DefaultEdge.class);




public static void main(String args[])
{

        connectToJDBC();
        collectFromJDBC();
        calculate_tweet_probabilities();
        calculate_retweet_probabilities();
        assign_probabilities();
        assign_followers();
        simulate_messages2();
        create_the_graph();
        get_all_edges();

}




public static void connectToJDBC()
{
        try
        {
                myConnection = DriverManager.getConnection(url, user, password);
                myStatement = myConnection.createStatement();
                myResultSet = myStatement.executeQuery(myQuery);
        }
        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }
}




/*************************************************************************************/
public static void collectFromJDBC()
{

        int userTweets;
        int userTenure;
        int userRetweets;
        int userFollowers;

        double TweetRate;
        double RetweetRate;


        try
        {

                while (myResultSet.next())
                {
                        userTweets = myResultSet.getInt("tweets");
                        userTenure = myResultSet.getInt("tenure");
                        userRetweets = myResultSet.getInt("retweets");
                        userFollowers = myResultSet.getInt("followers");

                        TweetRate = (double)userTweets/userTenure;
                        RetweetRate = (double) userRetweets/userFollowers;

                        allTweetRates.add(TweetRate);
                        allRetweetRates.add(RetweetRate);

                        if(highest_TweetRate < TweetRate)
                        {
                                highest_TweetRate = TweetRate;
                        }

                        if(lowest_TweetRate > TweetRate)
                        {
                                lowest_TweetRate = TweetRate;
                        }

                        if(highest_RetweetRate < RetweetRate)
                        {
                                highest_RetweetRate = RetweetRate;
                        }

                        if(lowest_RetweetRate > RetweetRate)
                        {
                                lowest_RetweetRate = RetweetRate;
                        }


                }

        myConnection.close();
        }

        catch (SQLException e)
        {
                e.printStackTrace();
        }


        catch (Exception e)
        {
                e.printStackTrace();
        }

}




/*****************************************************************************************/
public static void calculate_tweet_probabilities()
{
        int i, k, l;
        double rate = 0.0;
        double temp = 0.0;
        boolean breakLoop = false;

        rate = (highest_TweetRate - lowest_TweetRate)/tweet_profiles;

        for( k = 0; k < tweet_profiles; k = k + 1)
        {
                tweet_probabilities[k][0] = (double) (k + 1);
                tweet_probabilities[k][1] = rate * (double)(k + 1);

        }

        for ( l = 0; l < allTweetRates.size(); l = l + 1)
        {
                temp = allTweetRates.get(l);

                i = 0;

                do{
                        if(temp > tweet_probabilities[i][1])
                        {
                                i = i + 1;
                                breakLoop = false;
                        }

                        if(temp <= tweet_probabilities[i][1])
                        {
                                tweet_probabilities[i][2] = (double) 1 +
                                tweet_probabilities[i][2];
                                breakLoop = true;
                        }


                }while(breakLoop == false);
        }

        for (i = 0; i < tweet_profiles; i = i + 1)
        {

                double tempDecimal = tweet_probabilities[i][2] / allTweetRates.size();
                tweet_probabilities[i][3] = tempDecimal;
        }



}
```

```
        /**************************************************************************************/
        public static void calculate_retweet_probabilities()
        {
                int i, k, l;
                double rate = 0.0;
                double temp = 0.0;
                boolean breakLoop = false;

                rate = (highest_RetweetRate - lowest_RetweetRate)/retweet_profiles;

                for( k = 0; k < retweet_profiles; k = k + 1)
                {
                        retweet_probabilities[k][0] = (double) (k + 1);
                        retweet_probabilities[k][1] = rate * (double)(k + 1);

                }

                for ( l = 0; l < allRetweetRates.size(); l = l + 1)
                {
                        temp = allRetweetRates.get(l);

                        i = 0;

                        do{
                                if(temp > retweet_probabilities[i][1])
                                {
                                        i = i + 1;
                                        breakLoop = false;
                                }

                                if(temp <= retweet_probabilities[i][1])
                                {
                                        retweet_probabilities[i][2] = (double) 1 +
                                retweet_probabilities[i][2];
                                        breakLoop = true;
                                }


                        }while(breakLoop == false);
                }

                for (i = 0; i < retweet_profiles; i = i + 1)
                {

                        double tempDecimal = retweet_probabilities[i][2] / allRetweetRates.size();
                        retweet_probabilities[i][3] = tempDecimal;
                }



        }






        /*********************************************************************************/
        public static void assign_probabilities()
        {
                Random tweetP = new Random();
                Random retweetP = new Random();

                for(int i = 0; i < nodes; i = i + 1)
                {

                        SIMusers.add(new Nodes(i, Math.random(),
tweet_probabilities[tweetP.nextInt(10)][3],tweet_probabilities[tweetP.nextInt(10)][3]));

                }
}



/*****************************************************************************/
public static void assign_followers()
{

        for(Nodes k : SIMusers)
        {
                directedGraph.addVertex(k);
        }


        for(Nodes p : SIMusers)
        {
                int temp_user = p.get_nodeID();
                double temp_prob = p.get_probability_follow();

                for(Nodes j : SIMusers)
                {
                        double ill_follow_you = Math.random();

                        if(j.get_nodeID() != temp_user)
                        {
                                if( temp_prob >= ill_follow_you)
                                {
                                        directedGraph.addEdge(p,j);

                                }

                        }

                }
        }
}



/*****************************************************************************/
public static void simulate_messages_forwarding()
{

        for (Nodes p : SIMusers)
        {
                int temp_user = p.get_nodeID();
                double temp_prob = p.get_probability_tweet();

                double ill_retweet = Math.random();

                for(Nodes j : SIMusers)
                {

                        if(SIMusers.containsEdge(p,j) == TRUE)
                        {
                                temp_tweeter = j.get_nodeID();

                                for (int i = 1; i <= timeSteps; i = i + 1)
                                {
                                }
                        }
                }
        }

}
```

```
	/*******************************************************************************/
	public static void simulate_messages2()
	{
		int tweetID = 1;

		for (int i = 1; i <= timeSteps; i = i + 1)
		{
			for(Nodes p : SIMusers)
			{
				int temp_user = p.get_nodeID();
				double temp_prob = p.get_probability_tweet();

				double ill_message = Math.random();

				if(temp_prob >= ill_message)
				{
					SIMtweets.add(new Tweets(temp_user, tweetID, i));
					tweetID = tweetID + 1;
				}
			}
		}




	}



	/********************************************************************************/
	public static void create_the_graph()
	{
		//System.out.println(directedGraph.toString());
	}




	public static void get_all_edges()
	{
		for(Nodes p : SIMusers)
		{
			System.out.println(directedGraph.edgesOf(p));
		}
	}


}
```

Nodes.java

```
public class Nodes
{

	private int nodeID;
	private double probability_follow;
	private double probability_tweet;
	private double probability_retweet;


	/* class constructor */

		public Nodes()
		{
```

```
		nodeID = 0;
		probability_follow = 0.0;
		probability_tweet = 0.0;
		probability_retweet = 0.0;
	}


	public Nodes(int a, double b, double c, double d)
	{
		nodeID = a;
		probability_follow = b;
		probability_tweet = c;
		probability_retweet = d;
	}



/* public set class values */

	public void set_node_id(int this_id)
	{
		this.nodeID = this_id;
	}

	public void set_probability_follow(double this_prob)
	{
		this.probability_follow = this_prob;
	}

	public void set_probability_tweet(double this_prob)
	{
		this.probability_tweet = this_prob;
	}

	public void set_probability_retweet(double this_prob)
	{
		this.probability_retweet = this_prob;
	}




/* public get class values */

	public int get_nodeID()
	{
		return nodeID;
	}

	public double get_probability_follow()
	{
		return probability_follow;
	}

	public double get_probability_tweet()
	{
		return probability_tweet;
	}

	public double get_probability_retweet()
	{
		return probability_retweet;
	}


@Override
public String toString()
{

	return  "User " + this.get_nodeID();
}
```

```
}
```

Tweets.java

```
public class Tweets
{

        private int userID;
        private int tweetID;
        private int timeID;



        /* class constructor */

                public Tweets()
                {
                        userID = 0;
                        tweetID = 0;
                        timeID = 0;
                }

                public Tweets(int a, int b, int c)
                {
                        userID = a;
                        tweetID = b;
                        timeID = c;
                }


        /* public set class values */

                public void set_user_id(int this_id)
                {
                        this.userID = this_id;
                }

                public void set_tweet_id(int this_id)
                {
                        this.tweetID = this_id;
                }

                public void set_timeS_id(int this_id)
                {
                        this.timeID = this_id;
                }



        /* public get class values */

                public int get_userID()
                {
                        return userID;
                }

                public int get_tweetID()
                {
                        return tweetID;
                }

                public int get_timeID()
                {
                        return timeID;
                }


}
```

BIOGRAPHICAL SKETCH

Jose ‘Jv’ Villarreal earned his Bachelor of Arts in Spanish from the University of Texas Pan American in 2013. As an undergraduate, he completed a minor in Computer Science as well as a minor in English. He completed his Master of Science in Computer Science May of 2015.

Prior to earning his Master of Science, Jose had worked for over ten years in data analytics and business intelligence roles for T-Mobile, Comcast, AIG, and JP Morgan Chase. As a graduate student, he was given the opportunity to work as an Assistant Instructor for two semesters, teaching two sections of CSCI 1201 – Intro to Computers Systems. His goal is to begin a career as a software engineer/software developer. Jose’s thesis, *Use of a Simulated Directional Social Network to Compare Measures of User Influence*, was supervised by Dr. Christine Reilly.

For any questions regarding my research, I can be reached via mail: 2807 Vista Del Viento St., Mission, Tx 78572. As well, I can be reached via email at iamjv@outlook.com.