RE-RANKING REAL-TIME WEB TWEETS TO FIND RELIABLE AND INFLUENTIAL TWITTERERS

AHMED HUSAIN AL-SINAN

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIRMENTS
FOR THE DEGREE OF
MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGY
YORK UNIVERSITY
TORONTO, ONTARIO

FEBRUARY 2014

Abstract

Twitter is a powerful social media tool to share information on different topics around the world. Following different users/accounts is the most effective way to get information propagated in Twitter. Due to Twitter's limited searching and lack of navigation support, searching Twitter is not easy and requires effort to find reliable information. This thesis proposed a new methodology to rank tweets based on their authority with the goal of aiding users identifying influential Twitterers. This methodology, HIRKM rank, is influenced by PageRank, Alexa Rank, original tweet or a retweet and the use of hash tags to determine the authorisation of each tweet. This method is applied to rank TREC 2011 microblogging dataset which contains over 16 million tweets based on 50 predefined topics. The results are a list of tweets presented in a descending order based on their authorities which are relevant to the users search queries and will be evaluated using TREC's official golden standard for the microblogging dataset.

Acknowledgments

I would like to thank Dr. Jimmy (Xiangji) Huang for being my supervisor during my master's program here at York University. Also, I would like to thank him for his guidance and moral support for me to write this thesis until it is complete. I would further like to thank all my friends and colleagues in the Information Retrieval and Knowledge Management Research Lab especially: Dawid Kasperowicz, Dr. Jeff (Zheng) Ye, Dr. Mariam Daoud and James (Jun) Miao. They encouraged me during my research and helped me to understand what information retrieval is all about.

I would additionally like to thank the School of Information Technology at York University mainly the Graduate Program Director Professor Zijiang Yang, Administrative Assistance Ellis Lau and the former Department Secretary Sandy Yang. They made me feel welcome from the first day I joined the department.

Last but not least, I thank my family and my friends who supported me to continue with my graduate studies in Canada in every single way. Without you I would not be here and I really feel blessed for you all to be part of my life.

Table of Contents

Abbreviations

IR – Stands for ***I****nformation* ***R****etrieval*.

BM25 – Stands for ***B****est* ***M****atch* ***25*** which is a popular weighting function in the ***IR*** field.

MAP – Stands for ***M****ean* ***A****verage* ***P****recision*.

P@N – ***P*** refers to the number of relevant documents retrieved and ***N*** refers to the number of retrieved documents overall.

URL – Stands for a ***U****niform* ***R****esource* ***L****ocator* which also known as web address.

WWW – Stands for ***W****orld* ***W****ide* ***W****eb*.

HTML – Stands for ***H****yper* ***T****ext* ***M****arkup* ***L****anguage*; the authoring language used to create documents on the ***WWW***.

XML – Stands for ***Ex****tensible* ***M****arkup* ***L****anguage*.

TREC – Stands for ***T****ext* ***R****etrieval* ***C****onference* which is a conference that supports research in the ***IR*** field.

SQL – Stands for ***S****tructured* ***Q****uery* ***L****anguage*.

Chapter 1

Introduction

1.1 Background

Microblogging is a form of blogging but it differs from traditional blogging because it allows users to share short messages, along with images, videos and URL[1] links. When this service first introduced, people started using it for different purposes such as: sharing global news information, tracking real time events, expressing and promoting political views, advertising and many more. There are many different microblogging services were available on the web at first like Google Buzz, Tumblr and Jaiku. But the most renowned microblogging service is Twitter which made a global impact and changed the microblogging world in general.

Twitter microblogging service was launched in 2006 and it allowed users to write and read short messages/posts known as “tweets” from different Twitter users, whom are called “Tweeters” or “Twitterers”. Twitter allows its registered users to send and read messages that are up to 140 characters or less. With the limit of only 140 characters, Twitter automatically shortens any media

[1] URL: A uniform resource locator.

attachments within the tweet such as long URL, image and video links by using its own URL shortening service (t.co). This service was created in order to save space for the rest of the post since the tweet is only restricted to 140 characters. Twitter users are able to interact with each other by reading the tweets and take actions of those tweets such as sharing them. Meaning, a user can follow different users and repost their tweets in an effort to further spread the information contained within the Tweet. This feature is called “Retweeting” and it is the most popular feature that Twitter has. The more retweets a Tweet gets, the more popular it becomes [3]. Following different Twitter accounts will allow followers to receive real time tweet updates of the accounts they are following. Usually, these tweets are about topics of interest from both sides the author and the followers. Each tweet written by the author goes to the followers’ timeline giving specific details of each tweet such as date/time of creation of the tweet and the topic or the purpose of the tweet. In general, that is what made Twitter so popular because all its posts based on real-time events [3].

With over 300 million users worldwide and millions of tweets per day, Twitter became a target of different research fields of how to use this big flow of data. For this thesis, the aim is to benefit from Twitter’s microblogging data in the information retrieval (IR) field.

1.2 Motivations

With the increase growth of the information that flows on the Web nowadays, it is regular to surf the Web as part of our daily activities. The Internet is used on daily basis such as for work, pleasure, education, entertainment, connecting with others and also to search for information that we need to acquire. To further explain the search activity, searching has been always important for computers in general whether for documents on computers hard drives or for a simple Internet search used by one of web search engines. On the subject of the Internet, there is a huge amount of rich information that is scattered everywhere through the Web. However, having this flow of huge information

does not make it easy to be used in the right way to benefit of that information which might costs the loss of some important data. In order to solve this problem, the information needs to be treated in terms of validity by knowing the source of that information to check how reliable and accurate the source is. Web search engines can give a solution to this problem.

1.2.1 Web Search Engines

There are several web search engines available online and each one has its own way or method to rank and to display the information into several ranks. For example, search engines like Google rank Webpages based on a link analysis algorithm, called PageRank, to find the best results available on the Web and display them according to the "link popularity" based on that algorithm. With that, Webpages are ranked higher depending on how many links refer to them from different websites. With this method, it is safe to say that the sources of information on those Webpages are assumed to be more reliable than any other Webpages on the Web. Also not to mention that Google search is the most used web search engine on the World Wide Web. This is where information retrieval (IR) plays a major role in this activity by searching and retrieving the right information from the Web. Recently, social media networks became a topic of interests to the information retrieval field and some researches were done regarding that topic. Interestingly, most of social media networks now are offering the search engine property in order to find specific and valuable information in that network. At the present time, people from different parts of the world started sharing information through different social media networks. However, one obstacle regarding social media is how hard to get or to retrieve information from those networks. The main reason for that is the structure of the network was not built for that kind of purpose. Also, it is a struggle to know which information in this huge network is real or not because anyone can write anything on the Web these days. Since the internet is open and not restricted to anyone, then sharing fake information through the web is something understandable. Another thing to

consider is not to cause privacy issues because this information can be considered as private information between some social media users. As long as the information is not private, it will be ideal to find a way for others to use it so it can be beneficial for everyone.

1.2.2 Information Retrieval and Social Media Networks

By looking back to the obstacles of the social media, it came to my intention that knowing how valid the information that is shared in that network is quite challenging. Searching for the source of that information can solve that problem, but another question came in mind which is: Is that enough? The answer to that question might seem easy like for example, if the information comes from a reliable source in the social media network, then the information most likely to be real. But again, how do we know if that source is reliable? Social media networks are one of the biggest networks on the web so fake and false information is likely to exist there like anywhere on the web. Normally, social media users follow other users who share the same topics of interests. So, the first step is to know if the users that we follow share influential information or not? This is one of the main goals for this thesis which is to find influential social media users.

Popular social media networks like Twitter and Facebook took a major part in global events like the Arab Sprig revolutions that started in the Middle East in 2011. Sharing important live information to the public around the world made the actual news media look hopeless comparing to the social media that shared the information form the heart of the events. This type of information is called “breaking news stories” and it allows many online users to communicate with each other by receiving real-time updates on popular news, personally and globally, either from eyewitness users or other accounts that are interested in the same topic. Whether these breaking news were in simple text or media format, the majority of information that people received about the Arab Spring events

were considered to be valid. So, this proves that we can get real and valuable information out of the social networks.

1.2.3 Adapting Web Search Techniques to Social Media Networks

Since social media networks is mainly dealing with breaking news, then real-time Web information plays a major role in this field. It also means retrieving old information from social media is considered to be impractical. Meaning, social networks only deal with the latest breaking news so old news is not much to concern. With the lack of navigation support and limited searching in the social media networks, displaying authoritative posts became really hard no matter how old or new they are. So it came to mind that it will be best to rank these posts based on how important they are like any Web search engine does when searching for a query. The results will be displayed in a descending order of authority based on the search queries.

With the help of specific post's features, it will be able to identify which post is authoritative and relevance to the search queries. This is the reasons why Twitter is chosen for this thesis because its tweets have specific features to help determine how authoritative the tweets are. Also, the ranking system that Twitter is using is purely just based on the date and time of the creation of the tweet. Another ranking option that was added to twitter recently is ranking the tweets by checking the number of followers of that user. If the user tweeted about the search topic and has more followers than other user who tweeted about the search topic as well, then that user will highly be at the top of the ranking scale. Twitter assumes that the more followers the user has then the higher ranking that user should get. Both of these ranking methods are not practical and lack important properties of any efficient ranking system. In other words, they are too simple and they will not be effective to give an accurate ranking of the tweets. With the new proposed ranking method, ranking and identifying authoritative

tweets will be more effective. Also, this new method will help to recognize influential Twitterers which is another goal of this thesis.

1.3 Contributions

In this thesis, a methodology is proposed to identify and to rank Tiwtter's tweets based on their authority to improve retrieval performance. The main goal is to aid users locating authoritative tweets which will help recognising influential Twitter users. This methodology is influenced by other existing methodologies such as PageRank and Alexa Rank to help improve the ranking of the tweets. Additionally, this methodology will prove that it will be possible to determine the authority of the Twitterer and how relevant the tweet is to the query topic with the use of specific features that are within the tweets.

The following set of tweet features are presented to explain why they would help improve tweet ranking performance.

1. URL links: Each tweet can contain URL links to websites, videos and images. With the help of PageRank and Alexa Rank methodologies, it will be possible to determine how important these links in the World Wide Web.

2. Username ID: The "@" symbol precedes all Twitters usernames. It is optional for these tweets to be directed to specific Twitterers by linking them with the "@"symbol following a single or multiple usernames. If not then the tweets can generally be directed to everyone in Twitter, normally the author's followers, just simply by not mentioning any username within them.

3. Hash Tag symbol "#": The tweets can contain Hash Tags which are used to mark keywords or topics in a tweet to help users categorize the tweets while searching. The Hash Tag feature was created by the Twitterers and it became really popular especially to find out popular topics that are trending in Twitter globally and locally.

4. Retweets: The most important feature that helps spread the information within the tweet. If the tweet gets large amount of retweets from different users, then it is safe to assume that information that tweet has is important. However, most of the credit will go to the original author of that tweet since that user was the main source of that information.

5. Number of followers: Users whom are subscribers to other Twitter users to follow their tweets. It is important to know how many followers any user has to get an idea of how popular that user is. However, just looking by that number alone to decide if the tweet is important is not practical.

6. Verified twitter users: A verified badge helps users in Twitter to identify certain users, mostly popular celebrities and well-known companies, for high-quality sources of information regarding that Twitter account. This feature is not available to everyone so only the Twitter accounts that can get verified if they only meet certain qualifications assigned by Twitter's company.

The proposed methodology will use some of these features on an existing retrieval technique, which is provided by an information retrieval system, in order to achieve a better retrieval score of the original retrieval technique. This methodology will work on BM25 scoring function to produce new results that are both authoritative and will be ranked according on how relevant they are to the search query. BM25 is selected to be the baseline for this methodology and will build on its weight function in order to enhance it for a chance to get better retrieval results. By selecting the appropriate tweet features and using the appropriate combinations, this goal will be achievable.

1.4 Thesis Outline

The thesis is organized into six main chapters. It also includes a bibliography along with four appendix sections. The first chapter is introduction were an overview will be given about Twitter and explaining microblogging in general. The second chapter is literature review were other researches will be

addressed which can be related to microblogging and information retrieval. The third chapter is about the challenges that this research faced and how the proposed solutions of this thesis found a way fixed them. The fourth chapter is all about the materials that were used for this thesis including the dataset and the programs as part of the experiment. The fifth chapter will explain how to acquire the results of the experiment and study them in order to continue with the evaluation process. The sixth chapter comes last which provides an understanding of what the thesis accomplished from the final results. Also, the sixth chapter will suggest other strategies to improve the results of this thesis in the future.

Chapter 2

Literature Review

Information retrieval (IR) is the technique and process of searching, recovering and interpreting information from large amounts of stored data. It is a huge area in the information technology field where a lot of its techniques has been discovered, developed and tested for many decades. A lot of work has been done in the IR field which ended up with many researches in this area and this work has continuously improving until this day. However, the social media network was not part of interest for many IR researchers at first and microblogging is one of those areas to be related to the social network. It is understandable that the structure of these social networks was not built for that kind of procedure so it is a struggle to retrieve information out of them. With the increase popularity of Twitter and other social networks on the web, the social media network became a target of interest to many different researchers especially in the IR area. Their main goal is to benefit from of this huge network and the data that is stored in it.

The following sections will introduce some of the researches that are related to the IR field and other technology fields as well that can be related to this thesis.

2.1 Adaptive Other Search Methods on Twitter

One of the approaches to organize the search results in Twitter is to adapt other search methods to it such as faceted search method. Adapting faceted search on Twitter will help personalize tweets to different categories which will make the search topics well-organized and easy to display. One of these researches was done by Ilknur Celik, Fabian Abel and Patrick Siehndel [1]. Their goal was to solve the content exploration in Twitter because it requires time and effort to search and reach the right information from the tweets. So they focused on investigating ways to enhance searching and browsing by adaptive and personalized faceted searching. Normally, searching in Twitter is done by searching for a single keyword and then display recent posts related to that keyword from different Twitterers by calculating how many followers and followees they have. The results are time-sensitive as it is one of the features Twitter is using to get the search results displayed in that order. By adapting faceted search on Twitter, it will allow users to search for topics by specifying queries regarding different dimensions and properties of the topics. Even though the information on Twitter is short and unstructured, faceted search will put a limit on the size and the number of keywords that can be used as search parameters without to risk filtering out relevant results. However, the researchers admit that the ranking of the tweets that matched the faceted query is a problem and it can be solved by exploiting the popularity of the tweets like by measuring the retweets or by the popularity of the user that published the tweet. In other words, ranking the tweets was not their main concern in this research. Their main concern is to make search on Twitter more effective by presenting different methods such as personalized and context-adaptive methods.

2.2 Recognizing the Influential Twitterers

Another approach was done to identify the influential users/accounts to get the most reliable information possible in Twitter. One of these researches was done by Jianshu Weng, Ee-Peng Lim, Jin Jiang and Qi He [2]. This research

was the first to report the phenomenon of homophily[2] in the Twitter. Also, this research will help the users to know of whom users/accounts to follow for getting a timely update on their topic of interests that these users share. This can also help to identifies which Twitter account is active or not so users can avoid following online scammers. To do so, the researchers proposed to measure the influence of the twitterers called “TwitterRank” which is an extension of “PageRank” algorithm that is used by Google’s search engine. This paper also agrees that the number of Twitter followers alone cannot determine how popular and influential the twitterers are no matter what kind of topic we are searching for. So basically, they agree that Twitter’s ranking system for its users is not convenient at all. The approach of this research was done very well and it did give good results. However, their main goal was only to find the influential twitterers not to improve tweets ranking based on topics users are searching for in Twitter. They did mention that this will somehow improve the search ranking but they did not show any work to prove that at all. Finally, they mentioned that removing numbers, URLs, words with less than 3 characters and the words in the form of “@username” from the tweet will not help in the topic modeling without giving any valid reason for that assumption. This thesis is against the idea of removing such features from any microblogging dataset and in later chapters will explain why this thesis does not support that in order to improve Tweets ranking and in order to recognize influential Twitterers.

2.3 Information Spreading with Retweeting

There are research tactics to predict how the information spreads in Twitter and there are some written researches about that topic. Most of these researches agree that retweeting is the most important feature to predict information spreading in Twitter. One of these researches is worth to mention was written by Bongwon Suh, Lichan Hong, Peter Pirolli and Ed H. Chi [3]. The authors of this research proved that retweeting is the key mechanism for

[2] Twitterers follow other users because of similar interests which will lead them to follow back as well.

information diffusion in Twitter. Also, these authors showed curiosity to know how and why certain information spreads more widely in Twitter than any other posts. In other words, what makes the tweet retweetable? By examining number of Twitter features that can affect the retweetability of the tweets, they found out that among all the tweet features that URLs and hash tags are the most important to make the tweet retweetable. They also proved that the number of followers and followees are important as well but not as important as the URLs and hash tags which contradict what other researches proved. This paper has the best explanation of the term “retweet” in my opinion. It describes it as: “Retweeting can be understood as a form of information diffusion since the original tweet is propagated to a new set of audience, namely the followers of the retweeter.” Another research was done to follow this research by Sasa Petrovic, Miles Osborne and Victor Lavrenko [4]. However, they used human experiments for the approach of their research. Both of these researches collected their datasets from crawling Twitter’s API which it took a while to collect because Twitter only provides a small sample of the entire stream. This research collected a total of roughly 21 million tweets dataset while the previous research collected 74 million tweets. Again, it all depends on how long to crawl Twitter’s API for the tweets. What is interesting about this research is the authors divided Twitter’s features into two sets: social features and tweet features. The social features are related to the author of the tweet such as: Number of his/her followers, Number of accounts he/she is following, is the user verified and is the user’s language is English. The social features can be checked from Twitter’s API which is the biggest advantage to get the dataset from there. The Tweet features on the other hand are related to the tweet or the post itself such as: Number of hash tags, URLs, mentions, length of the tweet and if the tweet is a reply (which means a direct message from user to another user). Their research showed that the Tweet features are more beneficial than the Social features to predict if the post to be retweetable or not. And since they used human experiments, they also mentioned that it is possible to predict how well the post will spread just by looking at text of the tweet itself and it will be better to know who is the author

too. Lastly, they found out by removing infrequent words from the tweet will hurt the performance of their research. This thesis supports this theory on how important is to use every character in the tweet so they ended up using all the words in the post. Mainly, the goal of this thesis is to show how important the Tweet features are in any research regarding Twitter. The importance of the Tweet should be determined by what it contains and that is what is trying to be approved in this thesis. One more thing to mention is there are different types/styles of retweeting depending on the purpose of that action. Meaning, retweeting can be used for other purposes other than just spreading information like to reply to a post or to join a conversation that is trending on Twitter. Some researchers wrote a paper regarding that feature in order to examine the practice of retweeting from a conversational aspect [5]. They discussed the different types/styles of retweets by giving examples of each type/style. Also, they discussed what and why people retweet but not in depth since it is not their main goal in that research.

By taking this background knowledge into consideration, the next section will explain the problem in details, what exactly is needed to fixed this problem and how to fix it.

Chapter 3

Information Retrieval Challenges and Proposed Solutions

3.1 Retrieval Challenges for Microblogging

Information retrieval techniques in general have been examined using many statistical methods before being used in many researches. With the rich amount of information that is available in many digital forms, several of these IR techniques have shown good results which prove how accurate they are. However, when it comes to search for specific information in these digital forms then the limited search feature is noticed which is one of the problems in most social media networks. With the Internet becoming available to everyone, searching for information became easy since the web contains countless numbers of rich data about almost everything. With the use of the web search engines, it even became easier to find the popular information based on different algorithms that are used in information retrieval techniques. With social media is considered to be part of the Internet domain, the same retrieval techniques of

web search engines cannot be used on social media networks because the form of data differs from social media service to another. Even though a unified search model for social media was proposed as a solution for the retrieval task, that will not necessary solve all the problems due to the difference characteristics among the different types of social media [8]. Also, since there are many different IR researches on different types of social media services such as: blogging, microblogging, forums and wikis, researches shown it is highly proven that different retrieval techniques are needed to improve retrieval tasks to each type of social media [9]. In fact, most of these researches only focus on one type of social media which proves how different they are from each other.

As a member of different social media services, I noticed that almost every information I need to acquire can be found in these services especially Twitter. Comparing to any traditional media channels such as CNN, BBC and FOX news, Twitter can provide its users with fast and wide information spread and for the information they are interested in only. In other words, the users will not receive any random or unwanted information. In fact, a study showed that 85% of Twitter topics are headline news or persistent news in nature [10]. I also noticed it is hard to look back for old tweets even with the search engine that Twitter is providing, which is one of its jobs, to serve that purpose. Plus the limitation of that search engine showed that Twitter needs a better system to display and to arrange popular tweets related to the search topic like any web search engine. These simple ideas are what started my approach to write this thesis and gave me the urge to investigate more about it.

3.2 Proposed Approach to Address the Challenges

Understanding how Twitter search engine works was important in order to approach this study. Twitter is using the number of followers as one of the keys to determine the popularity of the Twitterers [11] [12]. However, it is not the main key to determine that as other researches explained [2]. Unfortunately, this is the system that Twitter is using for its search engine in order to rank the tweets as

well. The more followers Twitterers have, the higher rank their tweets will get in the search results. Here is a list of the problems that we can get from this system:

1. Tweet content: Checking the number of followers alone cannot indicate the authority of the tweet. The tweet can be related to the topic but that does not necessary mean it is important.

2. Author's knowledge of the Tweet topic: Twitter allows its users to express and write anything without any restrictions as long as it does not exceed 140 characters. If a specific tweet will be ranked then its author should have knowledge about the topic of the tweet or at least provide a reliable source for the information.

3. Active/inactive followers: Not all accounts on Twitter are real and not all of them are active as well. Fake and inactive accounts can cause for spam tweets and these accounts are used to increase the number followers and will help improve in the tweet ranking. So, it is important to communicate with real and active followers on Twitter because the more active they are the more they likely are to share the content of your tweets.

With these points in mind, the main goal is to find the authoritative tweets which later will lead to find influential Twitterers. By finding the authoritative tweets, it will be ideal display them in descending order based on their authority according to the search topic. Then, we can find the influential Twitterers whom are the authors of these authoritative tweets. In fact, these two are different tasks but they can be combined in order to improve the search system for Twitter. This is what makes this method different and unique comparing to others.

3.3 Uniqueness

There are different researches that proposed to improve the post ranking system in Twitter and each research has its own different approach to achieve

that. However, some of those researches did not use any of the tweet features to improve the ranking like [1] and others claimed that removing such features will improve the ranking like [2]. Based on how Twitter search engine works, tweet features play a major role to rank real-time web tweets in general based on the search query. In fact, Twitter can use these features for further purposes like for identifying influential twitter accounts as well. This section will explain why tweet features are important in Twitter and why the approach of this thesis is considered unique by using those features for the tweet ranking and for finding influential Twitterers.

Tweet features are not only important for ranking tweets but they are important to Twitter in general. In fact, they are part of the reason why Twitter is so successful and made it what it is today. According to [2], identifying influential twitter accounts can help companies with their marketing plan since Twitter is also considered a marketing platform for many companies. By targeting and focusing on the influential marketing accounts, this will increase the efficiency of the marketing campaign for these companies. Also, those features can also be used to help event detection and to identify the importance of these events [13]. This paper explains in details some of the tweet features that are mentioned in this thesis and grouped them in a section named (Twitter-specific features). Even though the paper did not explain how exactly these features will be used in the research, the results showed progress for event detection which clarifies how important these features are. Overall, tweet features are important for Twitter and they did help with different microblogging researchers in the past. So the goal for this thesis is to find a way to use these features to rank tweets which can lead to identify influential Twitterers.

Twitter Inc. is trying to come up with several ways to improve their microblogging ranking system by applying different methods. One of these methods that Twitter Inc. is currently using is by checking some of the tweet features like: Number of the account followers, number of users following the account (followees) and verified accounts. These three tweet features alone can

rank some important tweets but they are not enough to recognize influential Twitterers [2]. By knowing that, it becomes clear that ranking system needs improvement. Unlike the approach of this thesis, identifying influential Twitterers is one of the main goals by re-ranking the tweets based on their authority. By checking if the tweet has more tweet features that can be related to the query, the probability of ranking important tweets will increase. This is the main difference of this approach and what makes this method unique comparing to others.

The method that is proposed in this thesis focuses on the tweet itself by checking how many of these features are within the tweet. With that, Identifying influential Twitterers will be possible based on how knowledgeable these users are to the search topic. Ranking these tweets will be similar to any ranking system that being used to any familiar Web Search engine. The tweet results will be displayed in a descending order based on their importance to the search query by applying the thesis proposed methodology to improve the ranking which will be explain in details in the next section.

3.4 Proposed Methods to Improve Microblogging Retrieval

The strategy for this section is to define the starting point and the goal or outcome of the proposed method of this thesis. As it was mentioned before, the main goal of this study is to find authoritative tweets in which will help to recognize the influential Twitterers. To understand the structure of both methods, they need to be explained in details especially the starting point. The reason for that is the starting point contains all the heavy work for this method. For that, the starting point had to be dived into two separate tasks. These two different tasks are designed to get two different ranking scores and they are: BM25, which is the initial retrieval ranking score, and HIRKM tweet ranking score which is the main method of this thesis. The final step is to compare the results of these two scores against the golden standard of TREC 2011 microblogging dataset with the official results evaluation that was assigned by TREC.

In order to get the initial retrieval score, an indexed dataset was needed and that will be explained in chapter 4. The indexed dataset was used to get the retrieval scoring function such as BM25 or any other retrieval techniques. The score will be displayed in two sections: Tweet ID and tweet ranking score. The tweet ID is a UID[3] which normally is numeric or alphanumeric string that is associated with a single entity within a given system. UIDs make it possible to address that entity, so that it can be accessed and interacted with [29]. For the tweet ID, it is all numeric and it is given in the dataset to identify each tweet in the dataset. The tweet ranking score is the retrieval scoring result of each tweet after the task of the retrieval technique. It is a float of numbers indicating how likely the corresponding tweet is relevant the query.

The next section will explain BM25 retrieval method and will give a brief introduction about it and how it was developed to become one of the most recognized techniques in the information retrieval field. Furthermore, the information of BM25 in the next section was inspired by other information retrieval researchers that used BM25 as a baseline in their work [14] [34] [35] [36] [37] [38] [39] [40] [41] [42].

3.4.1 BM25

As mentioned in previous chapters, BM25 is arguably considered to be one of the most important advancements in the field of information retrieval. The function has proven itself by being used with a wide variety of content when ranking data was necessary, and it has always performed well in each of the fields of research. With its high tuneability and being well defined, BM25 is the go-to ranking function for most researchers. BM25 takes an ad-hoc approach to ranking documents, where the formulas are used because they seem to be plausible.

[3] UID: Unique Identifier

In this section, we will have a clear understanding of how BM25 started from a basic weighting model, and evolved into sophisticated ranking model by adding various improvements such as the 2-Poisson model, term frequency improvements, document length improvements, and query term frequency improvements.

	Relevant	**Non-Relevant**
Term "Hurricane" occurs	$P(\underline{x}\|R)$	$P(\underline{x}\|\overline{R})$
Term "Hurricane" does not occur	$P(\underline{0}\|R)$	$P(\underline{0}\|\overline{R})$

Table 1: Contingency Table to Calculate a Document's Relevance

3.4.1.1 Basic Weighting Model

From a statistics point of view, the basic weighting equation that BM25 is derived from is expressed in the following equation:

$$w(\underline{x}) = \log \frac{P(\underline{x}|R)P(\underline{0}|\overline{R})}{P(\underline{x}|\overline{R})P(\underline{0}|R)}$$

Equation 1: Initial Weighting Model

Where $\underline{\boldsymbol{x}}$ is a weight of a given document, *P* refers to the probability, $\underline{\mathbf{0}}$ is a reference vector representing a zero-weighted document, and where *R* and $\overline{\boldsymbol{R}}$ are representative of relevance and non-relevance document respectively. For example, each component of $\underline{\boldsymbol{x}}$ may represent the presence or absence of a query term in the document or its document frequency, and $\underline{\mathbf{0}}$ could be the "natural" zero vector representing all query terms absent. An exemplification of this is seen in Table 1 for a single term query, "hurricane". Single term queries can be understood as the simplest queries possible as they only have one term. One calculates a document's relevance using Equation 1.

If we assume the terms are independent from each other, even for the queries that contain multiple terms, Equation 1 can be used to calculate the relevance of a document to a specific query by decomposing *w* into individual term weights. The equation can then be transformed to the following:

$$w = \log \frac{p(1-q)}{q(1-p)}$$

Equation 2: Transformed Equation Based on Individual Terms

Where $p = p(term\ present|R)$ and $q = p\left(term\ present|\overline{R}\right)$. With an appropriate estimation method, the equation can be transformed to become:

$$w^{(1)} = \log \frac{(r+0.5)/(R-r+0.5)}{(n-r+0.5)/(N-n-R+r+0.5)}$$

Equation 3: BM1 as Used by S.E. Robertson in TREC-1

Where *N* is the number of indexed documents, *n* is the number of documents in *N* containing the sought out term, *R* is the number of known relevant documents, and *r* is the number of documents in *R* containing the sought for term. The value *0.5* is used to smooth out the results. If we do not smooth the results, the $w^{(1)}$ will come from Equation 2 by replacing *p* with *r/R*, and *q* with $\frac{n-r}{N-R}$ respectively. $w^{(1)}$ from Equation 3 is also known as BM1 and it is used by S.E. Robertson in TREC-1.

3.4.1.2 2-Position Model

$w^{(1)}$ has the ability to model the presence and absence of terms; however, it cannot model the within-document term frequency. If one deals with within-document term frequency rather than the presence and absence of terms, then the equation is as follows:

$$w = \log \frac{p_{tf} q_0}{q_{tf} p_0}$$

Equation 4: Within-Document Term Frequency Weighting Equation

Where $p_{tf} = p(term\ present\ with\ frequency\ tf|R)$, q is the corresponding probability for $\overline{R}$, and p_0 and q_0 are those for term absence.

Some work has been done in creating a technique to model within-document term frequencies by means of the mixture of traditional Poisson distributions. Hater originally began work on 2-Position distribution [26]. Before discussing the 2-Poisson model, it is worth explaining the ideas that are necessary for the model to function.

One assumes that the occurrences of a term in a document have a random nature that reflects a real, but hidden distinction between documents that are about the concept represented by the term and those that are not. The documents that are about a given concept are described as being *elite* for that particular term. One may draw an inference about a given concept being elite from the term frequency, but this inference will actually be probabilistic. Additionally, relevance is related to a term being elite rather than to term frequency, which is assumed to be dependent only on a term being elite. The term-independence assumption is replaced by the assumption that the elite properties of different terms are independent of each other. It is useful to introduce this hidden elite variable in order to gain an understanding of the relationship between multiple term occurrences and their corresponding relevance.

The 2-Position model is a specific distributional assumption based on the elite variable hypothesis discussed above. The assumption is that the distribution of within-document frequencies is Poisson for the elite documents, and also for the non-elite documents but with different means. The 2-Position model assumes that a document length is constant.

For the 2-Position model, there are usually some estimation problems because the general estimation method for the Position parameters is not well defined, and because the model is too complex by requiring a large number of different parameters for establishing an estimation. Successive work on mixed-position models have been suggested. They provide alternative estimation methods that may be preferable what was exists [27]. Combining the 2-Position model with Equation 1, one can obtain the following weight equation for a term *t*:

$$w = \log \frac{\left(p' \lambda^{tf} e^{-\lambda} + (1-p')\mu^{tf} e^{-\mu}\right)(q' e^{-\lambda} + (1-q')e^{-\mu})}{(q' \lambda^{tf} e^{-\lambda} + (1-q')\mu^{tf} e^{-\mu})(p' e^{-\lambda} + (1-p')e^{-\mu})}$$

Equation 5: Combination of the 2-Position Model with the Initial Weighting Model

Where λ and μ are the Position means for *tf* in the elite and non-elite sets for t respectively, $p' = p(document\ elite\ for\ t|R)$, and q' is the corresponding probability for $\overline{R}$.

The estimation problem is apparent in Equation 5 in that there are four parameters for each term, which none is likely to have direct evidence because of the elite variable being a hidden variable. It is because of this problem that makes Equation 5 inflexible, which leads one having to go through the rough model approach.

3.4.1.3 Term Frequency Improvement

In order to allow within-document term frequency *tf* to influence the weight, different functions are utilized. Once of the functions is effective and based on the rational that even if we do not use the full Equation 5, one may use it to suggest the shape of an appropriate equation. Looking at Equation 5, one can see the following characteristics:

- it is zero for *tf = 0;*

- it increases monotonically with *tf*
- but to an asymptotic maximum;
- that approximates to the Robertson weight that would be given to a direct indicator of being elite

After rearranging Equation 5, we get the following formula:

$$w = \log \frac{\left(p' + (1-p')(\mu/\lambda)^{tf} e^{\lambda-\mu}\right)\left(q' e^{\mu-\lambda} + (1-q')\right)}{\left(q' + (1-q')(\mu/\lambda)^{tf} e^{\lambda-\mu}\right)\left(p' e^{\mu-\lambda} + (1-p')\right)}$$

Equation 6: Rearranged Equation of the Combination of the 2-Position Equation with the Initial Weighting Model

From Equation 6, μ is smaller than λ. As *tf* →∞, $(\mu/\lambda)^{tf}$ goes to zero, and $e^{\mu-\lambda}$ is small and as such, can be estimated as:

$$w = \log \frac{p'(1-q')}{q'(1-p')}$$

Equation 7: The Estimation of the Rearranged Equation of the Combination of the 2-Position Equation with the Initial Weighting Model

It is necessary to construct an equation that is *tf*-related and satisfies the characteristics outlined for Equation 5. Such an equation can be constructed by the following principles: The function $tf/(constant + tf)$ increases from 0 to an asymptotic maximum in approximately the right fashion. The constant determines the rate that the increase drops off. With a large constant, the function is linear for small *tf*, whereas with a small constant the effect of increasing *tf* rapidly decreases. This function has an asymptotic maximum so it needs to be multiplied by an appropriate weight similar to that of Equation 7. Since one cannot estimate Equation 7 directly, the alternative is using the

ordinary Robertson weight $w^{(1)}$, based on the presence and absence of a term. With this, one obtains the following:

$$w = \frac{tf}{(k_1 + tf)} w^{(1)}$$

Equation 8: Weight of a Term Based on the Presence and Absence of a Term using the S.E. Robertson Equation from TREC-1

Where k_1 is an unknown constant. The model does not convey anything about the kind of value that is to be expected for k_1. S.E. Robertson determines the value for k_1 by experiments with TREC datasets. They found values around 1.0 – 2.0 are correct values for TREC data. Additionally, they pointed out that the shape of Equation 8 is different from Equation 6 in one important way; Equation 6 is convex towards the upper left, whereas Equation 8 can be S-shaped with some combinations of parameters, which increases slowly in the beginning, then rapidly increased in the center, and finally slowly again.

3.4.1.4 Document Length Improvement

After the document term frequency is integrated into the weighting function, the document length becomes the next issue that needs addressing.

In real situations, a document can be short or long. Both these short and long documents may also have the same subject. At a minimum, there are three reasons why documents lengths vary in length. The first reason is that some documents may cover more material than other documents. For example, a long document may consist of a wide variety of unrelated short documents concatenated together. This is known as the scope hypothesis. The second reason why document length varies is that a long document may be similar to a short document in terms of the message it conveys; however, because the two documents have different authors, the individuals writing style of the authors makes one longer than the other. For example, one document covers a similar

scope to a short document, but it uses more words to convey the same content. This is known as the verbosity hypothesis. The third reason is that real document collections have a combination of the aforementioned two reasons.

There is little progress in relation to the scope hypothesis, and the work on document length discussed here assumes the verbosity hypothesis. The verbosity hypothesis implies that a document's properties, such as relevance and being elite, can be regarded as being independent from the document length. Being elite is given for a term, and the number of occurrences of a given term depends on the document length. From this perspective, one can incorporate this hypothesis by normalizing *tf* for a document length *d*. Assuming the value of k_1 is appropriate to documents that have an average length Δ, the weight of a term is then expressed as:

$$w = \frac{tf}{\left(\frac{k_1 * d}{\Delta} + tf\right)} w^{(1)}$$

Equation 9: The Weight of a Term Based on Average Document Length

3.4.1.5 Query Term Frequency Improvement

There is natural symmetry between documents and queries, and this suggests that one could treat within-query term frequency *qtf* in a similar fashion to within-document term frequency. This suggests that by analogy with Equation 8, a weighting function for query terms can be as follows:

$$w = \frac{qtf}{k_3 + qtf} w^{(1)}$$

Equation 10: Weight of a Term Based on the Presence and Absence of a Term using the S.E. Robertson Equation from TREC-1 for Query Term Frequencies

Where k_3 is an unknown constant. Experiments suggest a large value of k_3 is effective, making the following equation to be equivalent to Equation 10:

$$w = qtf * w^{(1)}$$

Equation 11: Weight of a Term Based on the Presence and Absence of a Term using the S.E. Robertson Equation from TREC-1 for Query Term Frequencies with Large k_3 Value

Equation 11 can be thought as the normalization of query term frequencies. The basic assumption is that the frequency of query terms should have a direct effect on the weighting function. The more frequently a term appears in a query, the more important that term should be. It matches with the human intuition for natural language whereby they emphasize points by repeating key terms.

3.4.1.6 Final BM25 Formula

Once the above individual improvements are complete, they are integrated together to create the final weighting function. When there is no relevance information, $w^{(1)}$ approximates to the following equation:

$$w^{(1)} = \log \frac{N - n + 0.5}{n + 0.5}$$

Equation 12: BM1 as Used by S.E. Robertson in TREC-1 Revised for No Relevance Information

Based on this equation, two weighting equations become available for one to use:

$$w = \frac{tf}{k_1 + tf} * \log\frac{N - n + 0.5}{n + 0.5} * \frac{qtf}{k_3 + qtf}$$

Equation 13: BM15 Weighting Equation

$$w = \frac{tf}{\frac{k_1 * d}{\Delta} + tf} * \log\frac{N - n + 0.5}{n + 0.5} * \frac{qtf}{k_3 + qtf}$$

Equation 14: BM11 Weighting Equation

Equation 13 and Equation 14 can be combined into a single function known as BM25 that allows for numerous variations. The term frequency component is implemented as:

$$tf = \frac{tf^c}{K^c + tf^c}$$

Equation 15: Term Frequency Component in BM25

Where $K = k_1\left((1 - b) + b\frac{d}{\Delta}\right)$. Therefore, if *c = 1*, and *b = 1* gives the equation for BM11 and if *b = 0* gives the equation for BM15. Different values of *b* give a mix of the two equations. BM25 is referred to as *BM25(k_1, k_2, k_3, b)*. Therefore, the whole weight equation becomes the following:

$$w = \frac{tf^c}{K^c + tf} * \log\frac{N - n + 0.5}{n + 0.5} * \frac{qtf}{k_3 + qtf}\ and\ k_2 * nq\frac{\Delta - d}{\Delta + d}$$

Equation 16: BM25 Weighting Equation

There is an item $k_2 * nq\frac{\Delta - d}{\Delta + d}$ in Equation 16 that is called the correction factor. More details on the correction factor can be found in paper [28]. The k_2 value is usually set as *0* in experiments.

After explaining what BM25 retrieval technique is and how it was developed, it is time to understand what kind of connection it has to the proposed method of this thesis and how it will help it to get a better ranking score in this study. The proposed method is called HIRKM tweet ranking score and it will be explain in the section.

3.4.2 HIRKM tweet ranking method

HIRKM stands for: Huang's Information Retrieval and Knowledge Management which is a reference to the information retrieval lab that belongs to my supervisor Dr. Jimmy Huang at York University[4]. This method is the main method of this thesis and it consists of different formulas and percentage scores to give the appropriate value to each tweet feature if they are available in the tweets. However, there are certain steps needed to be done before getting into HIRKM method and its tweet feature values. This section will explain how HIRKM method was developed and how each of its tweet features got their values from.

The first step is to get Terrier's BM25 results file that has the relevant ranking results which belongs to each query topic. The reason for that is HIRKM was formed based on BM25 ranking technique with the addition of the tweet feature values that this study believes will approve the ranking of BM25. So basically, adjusting the BM25 score of each tweet to the tweet feature values will result a new ranking score and that score will be called HIRKIM tweet ranking score.

Second step is in order to use and to benefit from the tweet features; they all needed to be identified at first to check how many of these features are available in each tweet. Also another issue is how to organize the whole dataset to be able to identify all the available tweet features in each tweet. Building a database was the best option for that issue so a SQL[5] database was built stored

[4] http://www.yorku.ca/jhuang/irlab/index.php
[5] SQL: Structure Query Language

with the microblogging dataset. MySQL database was chosen for this task due to its popularity in the open source world and since large organizations rely on it such as Google and Facebook [30]. To build the MySQL database, TREC's statistic file was selected for that purpose. The statistic file contains the exact TREC microblog 2011 dataset but in a (.txt) format which made it more suitable for this task.

There are many tweet features that can be found in any common tweet such as: Username ID, verification badge, number of followers, number of followees, retweets, hash tags and URL links. Some of these features are considered mandatory in any tweet such as the username ID which is the author of the tweet. Some are considered optional to be added in the tweet such as hash tags which are used to check trending topics on Twitter. With the certain limitation that TREC allows using for the dataset, some of these main features are not provided in the dataset in order to prevent any privacy issues with Twitter .Inc. Some of these features are the verification badge, number of followers and number of followees of each Twitterer.

By identifying the remaining tweet features, they will be organized and stored in different database fields to be able to know which of these tweet features are available in the tweets that belong to dataset. Next is to propose several formulas and percentage scores to each of these features in order to assign a value to help get the tweet feature score. The Twitter username ID does not need a value at this part of the method but will still be stored in the database with other tweet features for later work regarding identifying the influential Twitterers. The tweet ID will be the primary key in the database will be stored in the database so it can be linked to the indexed data in Terrier. Chapter 4 will explain the database section in further details. After assigning a value to each feature, a new ranking score will be given to every relevant tweet related to the query topics which will create a new result ranking file based on the HIRKM method.

For this method, there are 3 main tweet features we need to identify before staring the re-ranking process and they are: URL links, Hash tags and retweets. Here are the proposed methods to the right values for each of these tweet features:

1) URL links: The value of a URL links are calculated using two existing services: (1) PageRank - represented by *PG(x)* - and, (2) Alexa Rank - represented by *AR(x)*, where *x* is the URL. The values of PageRank and Alexa Rank are normalized to be a value between 0 and 1. The normalization process for PageRank involves taking one-tenth of the PageRank. For example: if a URLs PageRank is 6; one-tenth of 6 is 0.6, which becomes the PageRank value of that URL. The normalization process of Alexa Rank involves creating 6 groupings: (1) URLs ranked 1 - 1,000, (2) URLs ranked 1,001 - 10,000, (3) URLs ranked 10,001 - 100,000, (4) URLs ranked 100,001 - 1,000,000, (5) URLs ranked 1,000,001 - 4,000,000 and, (6) URLS ranked over 4,000,000 or with no rank. Each of the groups is assigned a predetermined value: (1) 1.0, (2) 0.8, (3) 0.6, (4) 0.4, (5) 0.2 and, (6) 0.0. The proposed methodology considers PageRank and Alexa Rank to be equal in their authority as they calculate two distinct aspects of a given URL. As such, the average ranking of PageRank and Alexa Rank is taken to represent the final ranking of any URL contained within a Tweet using equation 17, where *x* is the URL.

$$URL(x) = (PG(x) + AR(x))/2$$

Equation 17: Calculation for the Rank of URLs

2) Hash tags: Hash tags are used to mark keywords or topics in a tweet to help users categorize the tweets while searching. It is optional for the Twitterers to add a Hash tag or several ones into their tweets as long as the tweet in general does not exceed 140 characters. Normally, hash tags are used for searching trending topics in Twitter which lately became a popular technique for tweet searching.

The TREC 2011 microblog dataset contains over 16 million tweets. And since the dataset is so big, there is no appropriate way to check if all the tweets have hash tags in them or not. Also, it will be hard to know exactly what they are and to what topic they belong to. For that reason, a .txt file was created and it was filled with different hash tags that were relevant and related to TREC's 2011 Twitter query topics. For example, one of TREC's Twitter topics is about the protests that happened in Egypt in the year 2011 as part of the Arab Spring. So, different hash tags were created related to that specific topic such as: #Egypt, #EgyptProtests, #Protests #ArabSpring, and so on. Just a reminder, that hash tags in general were created by many different Twitterers to make it easier for them to search different topics. So the purpose for this .txt file was only trying to detect as much possible hash tags in the tweets based on the topic query but not all of them. In order to know what kind of hash tags to create to each topic, a good knowledge of every topic was necessary so an extensive research was made in order to have an understanding of every topic possible. Searching online and reading articles related to all 50 topics that happened in the year 2011 was the main lead to know what kind of hash tags needed to be created for this method. Also, searching the popular and trending hash tags that were used for these topics in the year 2011 was needed and they were added to the .txt file as well. The reason for that is in order to make sure to be as much accurate as possible to when these tweets were created at that time of year. After acquiring them, they were added to the .txt file with the rest of hash tags. There are 50 Twitter query topics for TREC 2011 microblog dataset and 479 hashtags were created in the .txt file to detect them. One last thing to mention is hash tags that have misspelling errors were not considered in this study and were not added to the .txt file.

If a single relevant hash tag was detected in a tweet, then a value of 50% boost will be given to the new tweet score. If a second hash tag was detected, a 25% boost will be given to the new hash value tag which will result a 75% boost to both hash tags in total as the new hash tag feature score. Meaning, the

percentage will be decreased by half of the pervious hash tag score when a new relevant hash tag to be detected. The reason for that is, the more hash tags to be detected in the tweet the nosier it gets and that is why the other hash tags gets a less value than the first one. If no hash tags were detected in the tweet then the hash tag value is 0. Equation 18 explains the calculations of how hash tags get their values. The n refers to the number hash tags in the tweet, the ($\frac{1}{2}$) refers to the value of the first hash tag detected in the tweet and x is the hash tag(s).

$$HT(x) = \left(\frac{1}{2}\right)^{n}$$

Equation 18: Calculation for the Rank of Hash Tags

3) Retweets: The original plan for this study is the value of a retweet should be given to the original tweet and to its original author. However, since we are only dealing with dataset sample of the tweets then the chances of the existence of the original tweet there is really low. Later, an alternative plan was created since we can only deal with retweets in this dataset. In general, retweets should get a value but since it is not an original tweet then it should get a less value to the one that was planned to the original tweet. Of course, the first step is to detect if the tweet is a retweet or not.

Checking if the tweet is from an original author was not an issue since we were dealing with a sample dataset. According to [5], there are different styles of retweets that are used by different Twitterers. The most common one is by using the two letters R and T together to form (RT) and use that somewhere in the content of the tweet. So if the RT exists in the tweet then it clearly shows that it is not written by an original author so it is confirmed as a retweet. If no RT was detected, then the tweet is original and it was written by an original author. For this thesis, this style of retweets was chosen for the approach on identifying the original tweets from others which are the retweets.

The value of retweets, RT(x), will be either represented by 0.5 or 1 depending if the tweet is original or not. If it is a retweet, then the value of the BM25 score, URL value and the hash tag(s) value will be multiplied by 0.5. The reason of that is it is not an original tweet so it should get half the value of the original tweet. However, if it is not a retweet and comes from an original source, then the value of all the scores will be multiplied by 1 because it is an original tweet so it deserves the full value.

After acquiring the values of all three tweet features, it is time to adjust those values with the BM25 score to form the new tweet ranking formula of this study; HIRKM. Equation 19 will explain how the HIRKM method is calculated– represented by *HIRKM(x).*

$$HIRKM(x) = \big(BM25(x) + URL(x) + HT(x) \big) \times RT(x)$$

Equation 19: Calculation of HIRKM Tweet Ranking Method

3.4.3 Comparing scores to the golden standard

After obtaining both of BM25 score and HIRKM score, it is time to compare both results to the golden standard that belongs to TREC 2011 microblogging dataset. The golden standard is referred to the most accurate result possible without any restrictions [16]. Comparing the results of both methods to the golden standard will give the difference in performance values of each method and will store them in a file called the evaluation file. This procedure is done manually by writing a programing script that takes both methods results so it can be compared to the golden standard. The next chapter will explain the implementation process for HIRKM and what it was needed for it to be developed and to be evaluated with TREC's microblogging 2011 golden standard to be compared with BM25.

Chapter 4

Experimental Settings and Implementation

This section will explain implementation procedure for this study. In order to conduct an information retrieval experiment, there are some components needed in order to achieve the right results. These results must be measurable in order to compare them with other systems that are involved in the same research field. These components can be explained in these sections:

1. A raw microblogging dataset that can be related to the real world and can be part of it. This dataset must not be manipulated in any way before acquiring or it will not be acceptable for testing purposes.

2. A format of the dataset that can be used and recognized by the given information retrieval platform for this study.

3. An information retrieval platform for indexing and retrieval techniques to be used on the dataset.

4. A database system in order to store the dataset to organize it and to identify all the tweet features possible in all the tweets.

5. A method to evaluate the new results with other retrieval techniques for comparison in order to get the final results.

The following sections will explain the implementation of these components in details and how they work in order achieve reliable and accurate results.

4.1 Empirical Dataset

The first approach for this thesis was to get a microblogging dataset from Twitter's API. Unfortunately, that option was not available for several reasons. The main reason for not using Twitter's API is because it is time consuming which will take long time to crawl the data from there. According to [6] and [7], which are written by the same researchers, an access was needed to REST API to be able to download tweets from Twitter's API without rate limit restrictions. Researchers with that type of access can download the data in a feature-rich JSON[6] format. The advantage of crawling from Twitter's API is it gives a full tweet feature details of each tweet that cannot be possible to obtain from anywhere else. Here are some examples of these exclusive features: number of retweets of each tweet, the identity of the Twitterers who retweeted or replied to each tweet and finally the number of favorite tweets that were chosen by the followers or different Twitterers. Unfortunately, most of the microblog researchers do not have that type of access so alternative solutions were reserved to get a testable micoblogging dataset.

The microblogging dataset that was used for this thesis was acquired from Text Retrieval Conference (TREC) website of the year 2011 and is called Tweets2011 corpus. This corpus is consisted of 2 weeks of sampled tweets (24th January 2011 – 8th February 2011) courtesy of Twitter. It was designed to be reusable and represents both important and spam tweets. The size of the corpus

[6] JavaScript Object Notation

is approximately 16 million tweets and it includes different types of tweets such as retweets and replies. The corpus is split into different files called “blocks” and each block contains about 10,000 tweets of each day (i.e., block of tweets). Each of these tweets is in JSON format that is similar to the ones from Twitter’s API. However, they are not identical when it comes to the rich detailed JSON format that can be obtained from Twitter’s API so most tweet features are not available. Within the corpus, tweets are ordered by tweet ID which is the main feature of TREC’s JSON format [17].

After acquiring the dataset, what is needed is to make a use of an existing information retrieval system which can provide with suitable retrieval techniques to experiment with.

4.2 Terrier IR System

For this study, Terrier information retrieval platform was chosen for indexing and retrieval task. The main reason for choosing Terrier is because it is a well-known platform in the IR field. Also, Terrier showed good retrieval performances in other researches using classical information retrieval techniques such as BM25 scoring function. According to selected research papers, BM25 achieves good retrieval scores with Terrier IR platform [18]. Terrier is a highly flexible, efficient, and effective open source search engine, readily deployable on large-scale collections of documents. Also, Terrier implements state-of-the-art indexing and retrieval functionalities, and provides an ideal platform for the rapid development and evaluation of large-scale retrieval applications [19].

In order to use Terrier in this study, the dataset had to be converted to a format that can be recognized by Terrier IR platform. This is a necessary step in order to use the Terrier IR platform for the indexing part because the raw dataset is not readable yet for Terrier. So, the dataset was converted to TREC format mainly because it is applicable to Terrier and it is a recognized format that was used for many different researches in the in the field of information retrieval. Also,

the microblogging dataset is from TREC so it was best to convert it to the format of the Text Retrieval format. After conversion to TREC format, the dataset will be ready for the index procedure which will be explained in the next section of this chapter. For TREC format, there are many documents delimited by <DOC></DOC> tags [20]. Here is an example of how a tweet will look like after the conversion to TREC format:

<DOC>

<TweetID> 23041985 </TweetID>

<USERNAME> alsinan_ahmed </USERNAME>

<STATUS> 200 </STATUS>

<Tweet> Hello world ! It's Friday so follow @dkasperowicz #FF </Tweet>

<Time> 2011-01-10 00:45:54 </Time>

</DOC>

With TREC format, all tweets will be organized, readable and mainly the tweets within the dataset can be used for the indexing purpose with Terrier IR platform.

There is another step that takes part in the information retrieval researches which normally takes place before the indexing process and that is data pre-processing. Data pre-processing can be defined as a sequence of operations in order to clean, normalize and extract information from the data [21] [22]. Information retrieval researchers normally remove stop words, full stops, commas, apostrophise and other characters in that they think it can help to improve the ranking. However, this step will not take place in this study and will not be applied to the microblogging dataset. The reason for that is a single Twitter post can only fit only 140 characters so removing such characters from

the tweet itself can cause a lot of problems for the purpose of searching, retrieving and ranking the post. Also, Terrier does its own pre-processing part in order index the dataset so it was decided that one part of pre-processing was enough. In other words, every character counts in every tweet to reach the goal of this thesis. Also, there are researches that prove how important these features are and removing them will cause problems [4].

The process to build an index of the dataset and the performing retrieval techniques will be explained in the next section of this chapter.

4.3 Building Index and Performing Retrieval Techniques

After the dataset been converted to TREC format, it is time to build a Terrier index and then apply the retrieval weighting function on the indexed documents to get the evaluation results. The following sections will explain these two steps in details.

4.3.1 Terrier Indexes

Terrier, like most information retrieval systems, has the ability to build indexes and to preform many weight retrieval techniques as well. For the indexing part, Terrier creates the indexes documents based on the converted TREC dataset that was explained earlier in this chapter. By default, Terrier uses TRECCollection which parses corpora in TREC format delimited by the <DOC></DOC> tags like the converted dataset. For Tokeniser[7] implementation [23], Terrier uses the EnglishTokeniser[8] [24] by default. When indexing using another language, a different Tokeniser can be set up for that purpose [20]. Lastly, Terrier uses three methods for indexing documents and they are: 1) Classical-two pass indexing 2) Single-pass indexing 3) MapReduced indexing.

[7] A Tokeniser is a class which is responsible for tokenising a block of text, which is usually used by document implementations, into a stream of tokens.

[8] Tokenises text obtained from a text stream assuming English language. Acceptable characters are A-Z and 0-9.

For the first method, classical-two pass indexing; there are two techniques for classical indexing to be implemented. The first technique is BasicIndexer were the system performs stemming and stopword removal on all tokens. This a form of pre-processing that Terrier automatically preforms on the dataset before indexing. After that part ends, the system creates three data structures and they are:

1- DirectIndex: It is a compressed file that contains every term in each examined document and later is used for automatic query expansion.

2- DocumentIndex: A fixed-length entry file that contains information about the examined documents, the number of indexed tokens, the identifier of each document, and the equalizer of its corresponding entry in the direct index is stored.

3- Lexicon: Another fixed-length entry file that contains information about the vocabulary of the examined dataset.

Once all that data structures are completed, the InvertedIndex data structure will be created by Terrier where it contains the inverted values of the DirectIndex.

The second classical indexing technique is BlockIndexer which has the same functionality as the first technique. However, this technique can control a larger DirectIndex and InvertedIndex for storing the positions that each token occurs in each document. Also, it is better to use this technique over the first one when it comes to allowing queries to use term positions information, to add restrictions for searching multiple terms and to ignore any matches that do not follow the restrictions.

The second method is single-pass indexing where memory tracking is the main concern for this method. This method is implemented by other two techniques and they are BasicSinglePassIndexer and BlockSinglePassIndexer. The BasicSinglePassIndexer technique indexing operates in two phases. First, it

passes through the document collection to build an in-memory representation of the posting lists after several runs. The second phase is about merging these runs to create the final inverted file. The BlockSinglePassIndexer technique is similar to the first by performing a single inversion. The difference is it when it comes to index the document collection; it saves block information for the indexed terms. According to Terrier's website [20], single-pass indexing is by far quicker than the two-pass indexing method.

Third method that Terrier uses is the MapReduce Indexing and it is for large-scale collections. Basically, this method is the single-pass but in a larger scale due to the increase size of test collections. This method uses the single-pass indexer to index sections of each collection as map tasks. The results of this method come in three forms:

a) Terms and mini posting lists known as runs.

b) Document indices from each map task.

c) Information about the number of documents saved per run.

To index the dataset for this thesis, the single-pass indexing method was used for this procedure because it is the fastest method of the all the three and also it does not consume that much memory for this task. In order to Terrier to index, it needs to do its own pre-processing, stemming and stopword removal, on all tokens. It took approximately 30 minutes to finish indexing the documents. No problems were reported for Terrier to index the dataset and the index status is shown in Figure 4.1.

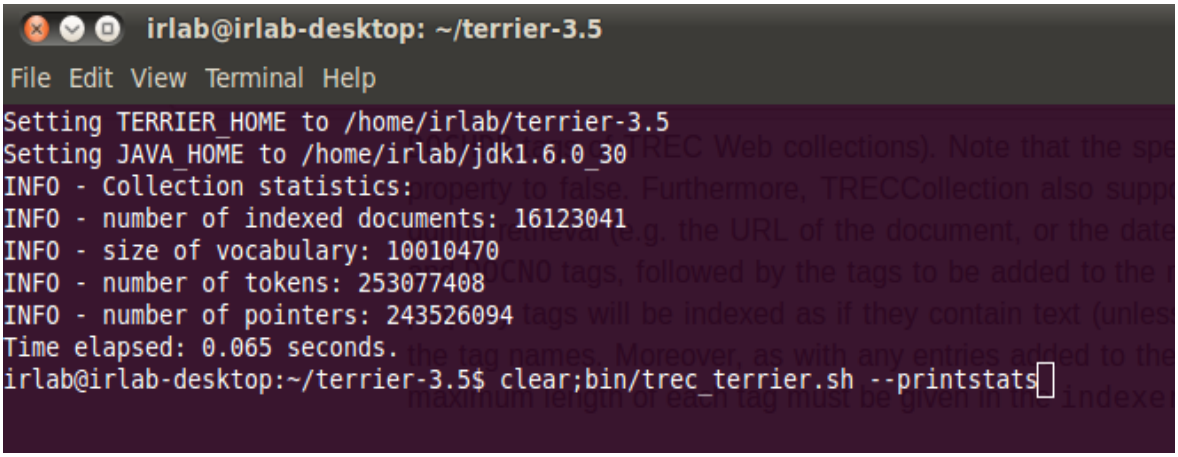


Figure 4.1: Terrier dataset Index status

4.3.2 BM25 Retrieval

As it was mentioned in this chapter, the dataset was converted to TREC format for the indexing purpose using Terrier IR platform. The indexed documents are needed to get the initial ranking score of the retrieval scoring technique such as BM25 or any other retrieval scoring function that are provided by Terrier. For this thesis, only BM25 retrieval results were needed so they can be compared with the results of the proposed method later on with TREC 2011 microblogging golden standard.

After configuring the settings for Terrier to use BM25 as the retrieval function for the indexed documents, the retrieval results will be stored in a new file called the (Results Ranking File). This file contains the tweets ranking results based on Terrier's algorithm for BM25. It took approximately 13 seconds for Terrier to retrieve the score with BM25. Next section, Preparing Queries, will give more details about the results ranking file.

The BM25 retrieval results needed to be kept for the re-ranking stage to get the HIRKM retrieval results at the very end. The next section will give more details about the ranking process and also about the results ranking file.

4.4 Preparing Queries

Queries preparation is an important procedure in order to get to the goal this thesis. TREC provided a list of 50 query topics to be used for experimenting with the microblog 2011 dataset. These queries were arranged in a marked up tag format which is similar to XML[9] format. Here is an example of how a query topic will look like:

<top>

<num> Number: 33 </num>

<title> Egypt </title>

<querytime> 2011-08-08 15:33:57 </querytime>

<querytweettime> 22041985 </querytweettime>

</top>

- The num tag is the query number.
- The title tag is the user's query representation.
- The querytime tag is the timestamp of the query in a readable form.
- The querytweettime tag is the timestamp of the query in terms of the chronologically nearest tweet id in the dataset.

When the system submits to rank the queries, the runs must be submitted to follow the standard TREC format. Here is an example of how the submitted runs will look like in the results file of the ranking method:

[9] Extensible Markup Language

```
01        Q0        3857291841983309        1        0.999        myRun
01        Q0        3857291841983302        2        0.878        myRun
01        Q0        3857291841983301        3        0.314        myRun
...
02        Q0        3857291841983301        1        0.989        myRun
...
```

The fields in order are the topic number, a literal “Q0”, a tweet ID, the retrieval rank of the tweet, the score and the identifier name of the run which is going to be BM25. There will be a second run for HIRKM later after identifying the tweet features in the tweets. This will be explained further in section 4.6.

The next section will give more details on how to build a MySQL database in order to store and organize the dataset to help identifying the tweet features for the HIRKM method.

4.5 Building MySQL Database

After obtaining the BM25 ranking results, the next step is to get HIRKIM ranking results so both methods can be compared to the official golden standard for TREC’s 2011 microblogging dataset. Building MySQL database is a main step in order to acquire the HIRKIM method after acquiring the BM25 ranking results. The main reason for storing the dataset into a MySQL database is to separate all the key aspects of the tweets in different fields. With that, all the tweets in general will be more organized to read and to use for the purpose of this study. By storing the dataset in the database, the tweet features will be easy to identify. However, there are some certain steps are needed to be done at first.

The first step is to build a MySQL database by the name (**ahmed_thesis**) that contains a table with the name (**twitterdataset**). The table was created with five fields or columns in order to separate the components of each tweet. The names of the five fields are:

1- **tweet_id:** The primary key of the table which contains a unique number to identify each tweet in the dataset.

2- **username:** The author and creator of the tweet.

3- **status:** Http[10] response code to specify the status of each tweet [25].

4- **created_at:** Time and date of the creation of each tweet.

5- **tweet:** A post on Twitter that was created to be shared with the other Twitterers.

Figure 4.2 shows how the **twitterdataset** table looks like in MySQL GUI[11] browser tool with its five fields.

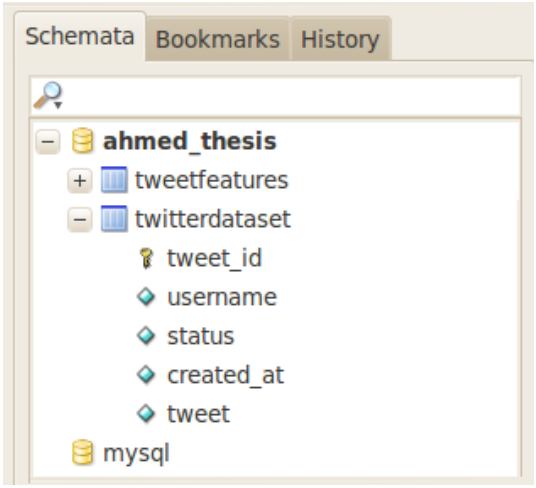


Figure 4.2: twitterdataset table in MySQL Database

For testing purposes, some SQL commands been executed to check how the tweets will be displayed in the MySQL query browser. Figure 4.3 shows an example of one of those commands and how the tweets are displayed after separating the key components of each tweet into five different fields.

[10] Hypertext transfer protocol
[11] GUI: Graphical User Interface

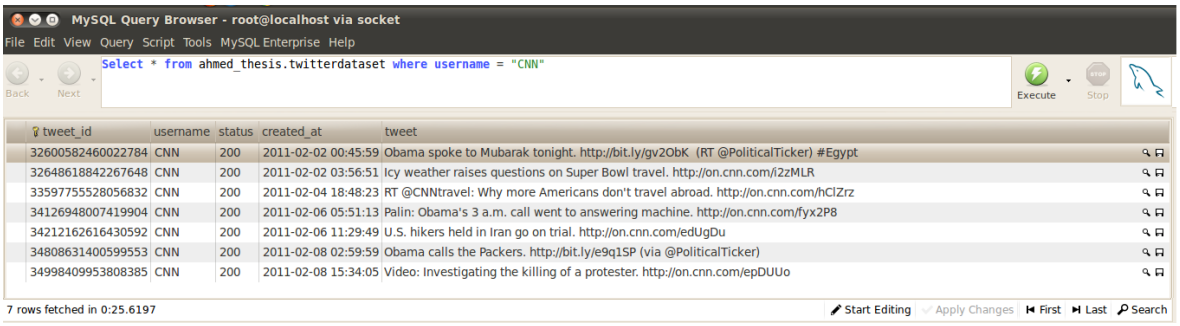


Figure 4.3: Example of how the tweets will be displayed after separating the components

The second step is to create a second table that contains and describes each tweet feature that can be available in the tweet and link them to the main table using a foreign key[12][15]. For that, table (**tweetfeatures**) was created and has the following columns:

1- **tweeFeatureID**: The primary key for this table that identifies a row in the database. Each row in this table needs to be identified to know exactly which row to look for.

2- **tweet_id**: The foreign key that connects between the **twitterdataset** table and **tweetfeature** table.

3- **featureType**: This column describes what kind of tweet features are in each row of the tweet. Like, does the tweet have any hash tag(s), URLs? Or is the tweet original or a retweet?

Different values were assigned to distinguish which of these tweet features are available in the tweet. Here are the tweet features with their values:

- Hash tags = 0
- URLs = 1

[12] Foreign Key: Is a database field or collection of fields in one table that uniquely identifies a row of another table.

- Retweets = 2

If a tweet has more than one tweet feature in it then then there will be different entries for that tweet in the database depending on how many tweet features are in the tweet. For example: If a tweet has a relevant hash tag and a URL then the tweet will be stored twice in the database with two feature types 0 and 1.

4- **featureText**: This column gives us the extracted feature we were looking for in a given tweet.

5- **featureValue**: This column gives a weight to each tweet feature that got extracted from the tweet in order to give the new tweet rank.

Figure 4.4 shows how the **tweetfeatures** table looks like in MySQL GUI browser tool with its five fields.

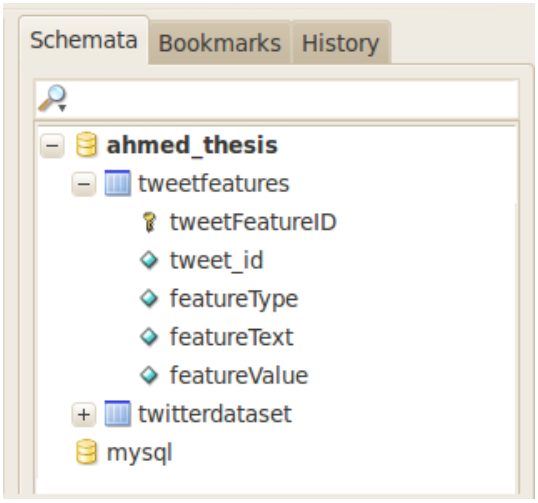
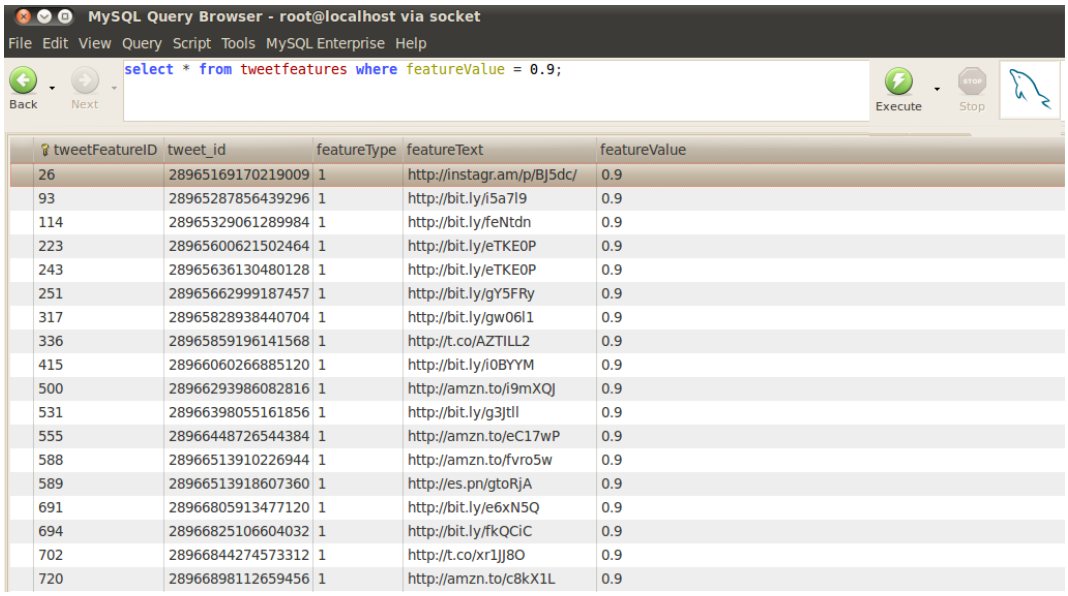


Figure 4.4: tweetfeatures table in MySQL Database

Similar to the previous table, another SQL command been executed to check how the **twitterfeatures** table fields will be displayed in the MySQL query browser. Figure 4.5 shows an example of one of those SQL commands.



tweetFeatureID	tweet_id	featureType	featureText	featureValue
26	28965169170219009	1	http://instagr.am/p/BJ5dc/	0.9
93	28965287856439296	1	http://bit.ly/i5a7l9	0.9
114	28965329061289984	1	http://bit.ly/feNtdn	0.9
223	28965600621502464	1	http://bit.ly/eTKE0P	0.9
243	28965636130480128	1	http://bit.ly/eTKE0P	0.9
251	28965662999187457	1	http://bit.ly/gY5FRy	0.9
317	28965828938440704	1	http://bit.ly/gw06l1	0.9
336	28965859196141568	1	http://t.co/AZTILL2	0.9
415	28966060266885120	1	http://bit.ly/i0BYYM	0.9
500	28966293986082816	1	http://amzn.to/i9mXQJ	0.9
531	28966398055161856	1	http://bit.ly/g3Jtll	0.9
555	28966448726544384	1	http://amzn.to/eC17wP	0.9
588	28966513910226944	1	http://amzn.to/fvro5w	0.9
589	28966513918607360	1	http://es.pn/gtoRjA	0.9
691	28966805913477120	1	http://bit.ly/e6xN5Q	0.9
694	28966825106604032	1	http://bit.ly/fkQCiC	0.9
702	28966844274573312	1	http://t.co/xr1JJ8O	0.9
720	28966898112659456	1	http://amzn.to/c8kX1L	0.9

Figure 4.5: How the tweetfeatures table fields are displayed

When both tables are created, it is time to prepare for the final step to get the HIRKM re-ranking retrieval scores that is based on the BM25 retrieval technique. HIRKM retrieval method and its scores will be explained in the next section.

4.6 HIRKM Retrieval

After finding all the requirements that is needed for the main method of this thesis, it is time to obtain the re-ranking results which is for the HIRKM retrieval method.

Similar to the previous BM25 Retrieval section of this chapter, the ranking process for HIRKM will be the same but after including the tweet features scores to its formula in order to get the new ranking score. This process is called the re-ranking process. However, Terrier IR platform will not be needed again for this part. As it was mentioned in previous sections, the HIRKM method is based on the BM25 retrieval technique with the addition of tweet features scores. So basically, the only item needed is the BM25 ranking results file which was

already acquired from Terrier from earlier. By adjusting the BM25 scores with the tweet feature scores, the re-ranking process will start and will result a new ranking file for the HIRKM method storing its re-ranking retrieval scores.

The following section will explain how the evaluation results work for both BM25 and HIRKM and what are the official evaluation methods for TREC 2011 microblog dataset.

4.7 Evaluation of Results

After all result files are generated, it is time for the evaluation process to compare these results to the golden standard. As it was mentioned before in this thesis, TREC already provided the official initial module and clear guidelines as to how the evaluation process works. This module is called (trec_eval).

Trec_eval module address several methods that are used to evaluate different information retrieval systems. This module can handle streams of quires and documents. This module works by interacting with information retrieval platforms in order to achieve series of retrieval tasks. In the end, it gives the final results after processing a set of files for the retrieval experiment. This is a necessary step for the retrieval methods to compare them to the best score which is the golden standard. The golden standard is an evaluation script that was provided by TREC to evaluate the performance of the retrieval methods to generate candidate concepts.

For this study, the evaluation module that was used to get the final results goes by the name trec_eval 9. The number 9 referrers to the version of this module and it was the latest version of it as well according to TREC's official website [31].

The following chapter will cover the evaluation results for both BM25 and the proposed method HIRKM. The results will be presented by the official

evaluation metrics assigned by TREC for the microblogging dataset which will also be explained in depth in the next chapter.

Chapter 5

Results and Evaluation

The ability to know how effective the results are can be challenging especially in the social media domain in the field of information retrieval. There are over 16 million tweets, 50 topic queries and 2 retrieval methods. In order to properly establish effective results, a baseline must be determined by using an existing methodology in the information retrieval field. Next section will give more details regarding how the baseline was made to generate results by using specific criteria which will also be explained further in this chapter.

5.1 Baseline

To ensure high performance and accuracy to be achieved for the microblogging dataset, a baseline must be determined for the ranking concept. For that, BM25 was used to generate that baseline because it is one of the most popular ranking methods in the information retrieval field. In fact, BM25 been used as one of the main baselines in different IR researches [32]. The BM25 parameters values were set by Terrier as follows: b to 0.75, k_1 to 1.2 and k_3 to 8. According to Terrier's website, these are the default standard values that

are set for BM25 [33]. Figure 5 explains how the established baseline that used with BM25 ranking method with the assigned parameters values.

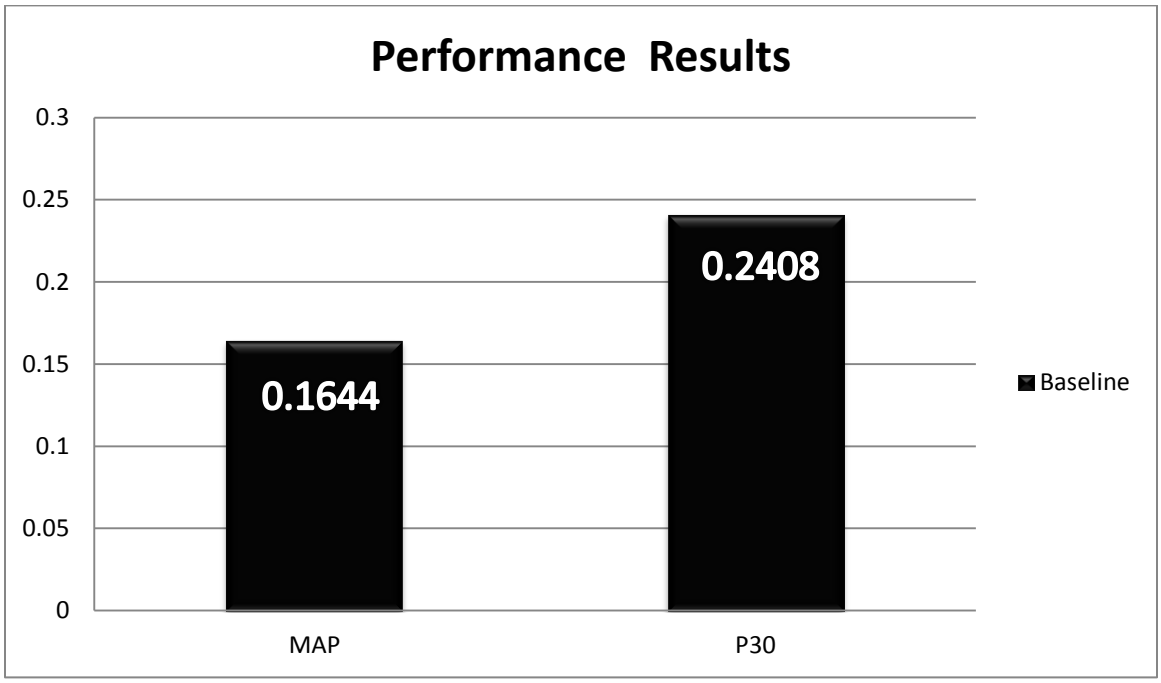


Figure 5.1: BM25 Performance Measure

The results are in a scale of 0 – 1, where the number 0 indicates no tweets from the results were considered relevant using the BM25 ranking method, and number 1 indicates that all the tweets from the results were considered relevant using the BM25 ranking method.

5.2 Performance Criteria

As it was mentioned in the last section of chapter 4, TREC provided an official initial module for the evaluation process and it is called trec_eval. This module generates different evaluation methods provided by TREC. Since the dataset is also provided by TREC, it is logical as well to use this module for the evaluation measures. For the microblogging dataset, TREC restricted two evaluation measures for that dataset and they are: Mean average precision (MAP) and precision @ 30 (P30).

For each query, the system should provide up to 1000 tweets. The P30 measure counts the number of relevant tweets in the top 30 tweets in the ranked list for a given topic. The MAP measure is an individual average precision score for a single query is calculated after each relevant document is retrieved. The mean average precision for the run is the mean of these average precision scores. All query topics are expressed in English and non-English topics will be considered irrelevant. An example of the evaluation measures for BM25 is shown in Figure 5.2.

```
runid                   all     BM25b0.75
num_q                   all     49
num_ret                 all     46992
num_rel                 all     2965
num_rel_ret             all     1873
map                     all     0.1644
gm_map                  all     0.0854
Rprec                   all     0.2254
bpref                   all     0.2688
recip_rank              all     0.3867
iprec_at_recall_0.00    all     0.4682
iprec_at_recall_0.10    all     0.3501
iprec_at_recall_0.20    all     0.2906
iprec_at_recall_0.30    all     0.2643
iprec_at_recall_0.40    all     0.2128
iprec_at_recall_0.50    all     0.1884
iprec_at_recall_0.60    all     0.1263
iprec_at_recall_0.70    all     0.0721
iprec_at_recall_0.80    all     0.0407
iprec_at_recall_0.90    all     0.0119
iprec_at_recall_1.00    all     0.0027
P_5                     all     0.2898
P_10                    all     0.2816
P_15                    all     0.2653
P_20                    all     0.2633
P_30                    all     0.2408
P_100                   all     0.1710
P_200                   all     0.1201
P_500                   all     0.0634
P_1000                  all     0.0382
irlab@irlab-desktop:~/Desktop/trec_eval.9.0$
```

Figure 5.2: trec_eval 9 evaluation measures for BM25 ranking method

As shown in Figure 5.2, Only 49 quires out 50 were ranked. The reason for that is none of the tweets in the dataset were found relevant to one of the 50 query topics so those tweets did not take part of the ranking. Other than that,

there were no other actions to report so the BM25 retrieval was successful and went according to plan.

5.3 Results

Terrier is the main retrieval tool in order to get the final results for this study. It is also the tool that created the baseline using the BM25 ranking method as it was explained in section 5.1. For this study, two runs were generated before getting the final evaluation results. The first run was using HIRKM without Terrier and BM25 and the second run was using HIRKM tweet features to fine tune the results of Terrier and BM25 as its baseline. Each of these will be explained in the next of this section.

5.3.1 HIRKM Run without Using Terrier and BM25

The first run and approach of this thesis was to immediately rank HIRKM and compare it to the golden standard without using Terrier and BM25 as its baseline. For that, the first step for this approach was to identify all the tweet features in the tweets in the TREC 2011 microblogging dataset and then rank them based on the formulas that were proposed in chapter 3. However, this approach was really time consuming and it took almost 4 months to identify and rank each tweet feature in the dataset especially the URLs part for Alexa and Page Rank. After gathering all the organized tweets, the second step is to see if these tweets were relevant to any of the 50 topics by checking the tweet features in the tweets. With that, a HIRKM results file was created which contained the ranked tweets based on the topic.

trec_eval 9 was used to compare those results to the golden standard and figure 5.3 illustrates the evaluation measures for that file.

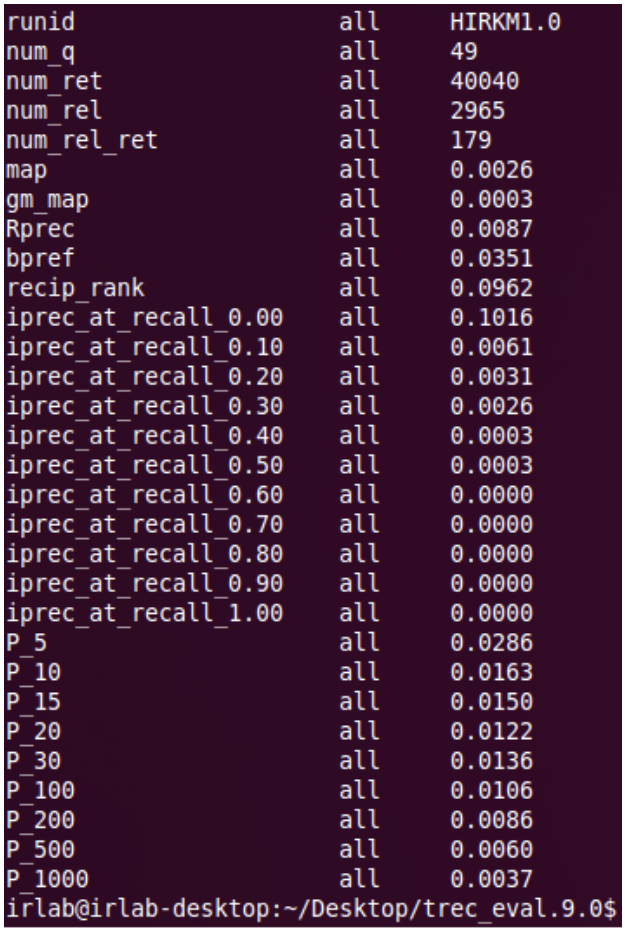


Figure 5.3: trec_eval 9 evaluation measures for HIRKM without Terrier and BM25 as a baseline

As it clearly shows, the results comparing to the golden standard ended up being very low. In fact, only 179 tweets were found relevant to the golden standard as it shows in Figure 5.3. So, the results were very low and an understanding was reached that this method was not the right approach to rank the tweets.

By having these bad results using only that method, a new approach was needed to find a fitting method to rank tweets based on their authority. A new approach strategy was made by checking the possibility of having a well-known ranking method as a baseline to help HIRKM with the ranking. Then, re-ranking the baseline results by fine-tuning the results with the tweet features to get even

better results based on the HIRKM method. Next section will give further details regarding this new approach which is the second run used for this thesis.

5.3.2 HIRKM Run Using Terrier and BM25

Before the second run was generated, it was planned to have a ranking method from Terrier IR system as a baseline for HIRKM. As it was mentioned before in previous chapters, BM25 is main ranking method that was chosen for this thesis and will be set as baseline for HIRKM.

With BM25 results file already available, it was all about rearranging the baseline results by using additional criteria from the HIRKM method. to get new ranking results for the tweets. This kind of adjustment of BM25 will cause re-ranking the tweet results which is based on the tweet features that are part of the HIRKM method. In order to do that, a program was set to look for the 3 main tweet features of the HIRKM method in all the BM25 tweet results. When any of the tweet features is detected, it changes the value of the tweet which will also change its ranking in the results file. When all that is done, a new results ranking file will be created based on the HIRKM method.

Trec_eval 9 was used again to compare the new re-ranking results to the golden standard and figure 5.4 illustrates the evaluation measures of the new file.

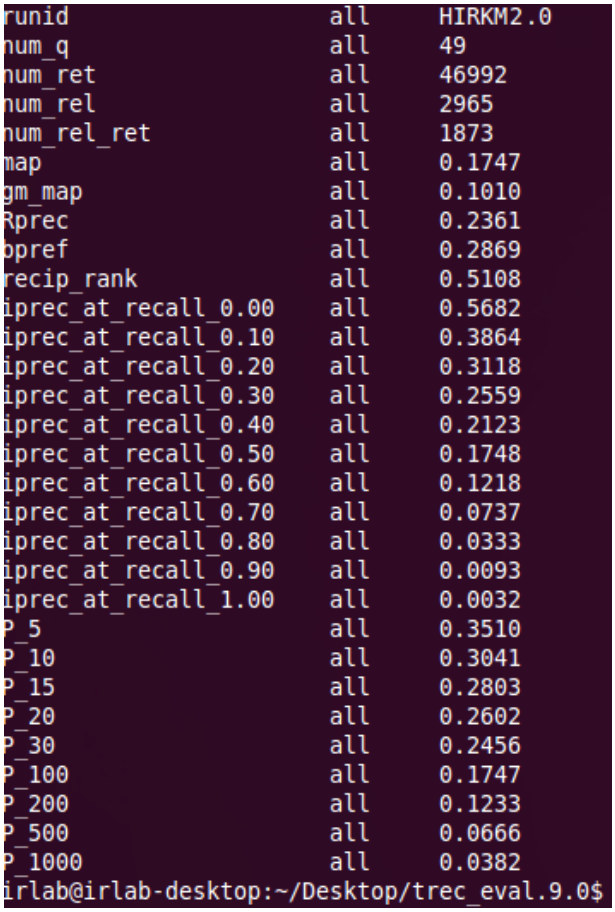


Figure 5.4: trec_eval 9 evaluation measures for HIRKM with the use of Terrier and BM25 as a baseline

By looking at these results, it clearly shows how well this approach did comparing to the first run. Also, the results were found reasonable as any ranking method would normally get. So, the second run was found successor to the first run and its results were selected as HIRKM's ranking results for this study.

Next section will analyze and discuss these results with the baseline results to check their efficiency by comparing them to BM25 ranking results from Terrier IR system.

5.4 Analysis and Discussion

The two runs that were conducted in this study were HIRKM1.0 and HIRKM2.0. The reason for conducting these runs was to determine which one of the proposed methods would prove to have a batter performance and efficiency when comparing them to the golden standard in terms of ranking the tweets. Table 2 will explain the difference between the proposed runs.

Run	Description
HIRKM1.0	It was generated using the tweet features scores of the tweets.
HIRKM2.0	It was generated by taking the baseline results that were generated using Terrier IR system and BM25 weighting model, and re-ranking them based on the tweet features scores.

Table 2: The Description of the Proposed Runs

By examining the results of HIRKM1.0 and HIRKM2.0 in Figure 5.3 and Figure 5.4 in the previous section of this chapter, it clearly indicates that HIRKM2.0 was superior by massive results. The reasons for that will be explained as follows.

HIRKM1.0, the first version of the proposed method, was built from scratch without the aid of any ranking methods. So basically, all the tweets in the TREC 2011 microblogging dataset were giving specific ranking values based on HIRKM's tweet features only. For the golden standard, it is unknown how TREC designed the highest results to rank the tweets and what kind of ranking methods was used to help building it. But still, the golden standard is all about ranking the tweets to the best way possible based in all 50 topics that were provided by TREC. So, the golden standard was designed for only one job which is to rank the tweets perfectly as much as possible based on the query topics. For HIRKM

however, it was mentioned several times earlier in this thesis that HIRKM is not only about ranking the tweets but it is about recognizing influential Twitterers as its second goal. So when comparing HIRKM1.0 ranking results to the golden standard, it was logical why the results were so low because HIRKM has two goals that needed to be achieved unlike the golden standard which again its only goal is to rank the tweets. So a conclusion was reached that focusing on the tweet ranking should be the main priority at first in order to get good evaluation results with golden standard. HIRKM1.0 lacks an important step to give it the right balance to be compared with the golden standard. By giving it a lot of thoughts, it was decided later on that the first step to achieve that approach is by having a ranking technique as a starting point to help HIRKM with the tweet ranking. That is when BM25 was decided to be the baseline of the second run which is called HIRKM2.0.

The second run, HIRKM2.0, was performed to attempt improving the performance of BM25 ranking results as shown in Figure 5.2. To reach that goal, the tweet features in the BM25 ranked tweets needs to be identified in order for HIRKM to work. As it mentioned in chapter 4, all the tweets been stored in a MySQL database and all the tweet features been identified there as well. For that, a strategy was needed to find a way to connect the BM25 ranked tweets to the tweets in the MySQL database. By looking back at section 4.4, it illustrates how the ranking format of the results file will look like and it also explains the fields of that file. The tweet id is the field that was needed to establish the connection between the BM25 results file and the database. The reason for that is the tweet id has been already stored in the database and it was already available in the results file as well. With it, the all the tweet features that belong to the BM25 ranked tweets can be identified so later they can be adjusted for HIRKM2.0 to be generated. Figure 5.5 will illustrate the performance of HIRKM1.0 and HIRKM2.0 runs in comparison with the baseline.

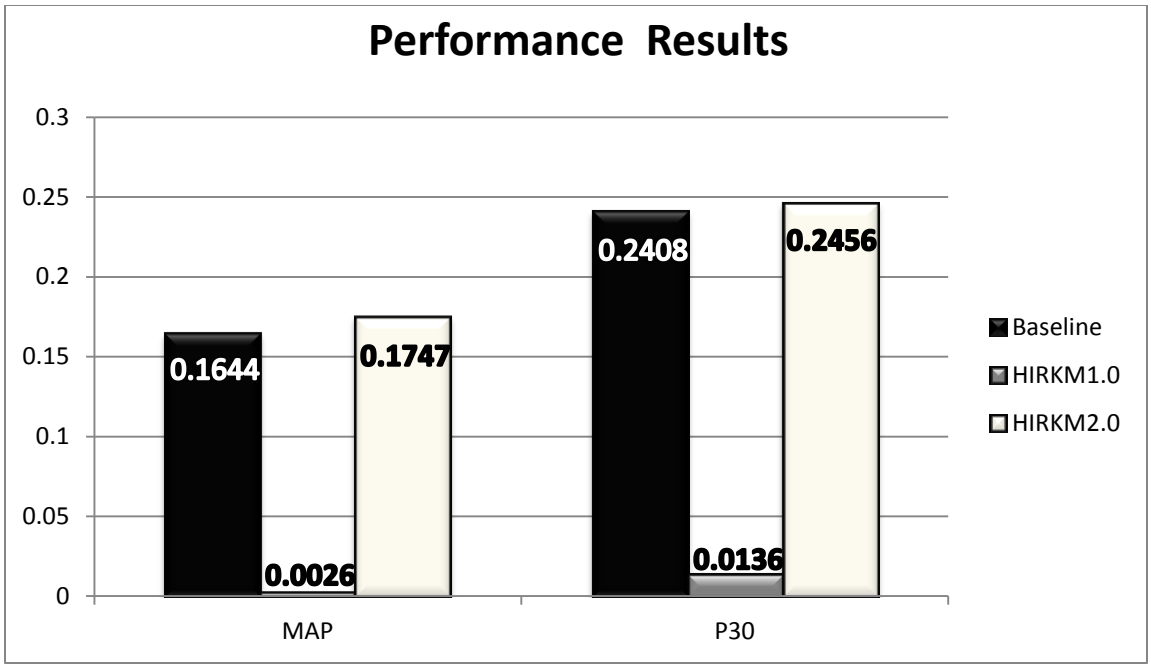


Figure 5.5: Performance Comparison of the Baseline, HIRKM1.0 and HIRKM2.0 Runs

After re-ranking the BM25 results with HIRKM, Figure 5.5 shows a slight improvement for HIRKM2.0 in both evaluation methods by comparing to the baseline. MAP performance measure improved by 1.03% and the P30 measure increased by only 0.48%. With these results, HIRKM2.0 outperforms the BM25 baseline by low percentage. For the first run however, HIRKM1.0 results were really low by comparing it to the baseline as shown also in Figure 5.5. In fact, the results show how much difference can be for HIRKM by having a baseline or not. The evaluated results of both runs are displayed in Table 3 along with the baseline results.

Run	MAP	P30
Baseline	0.1644	0.2408
HIRKM1.0	0.0026	0.0136
HIRKM2.0	0.1747	0.2456

Table 3: The Evaluated Results of the Proposed Runs and the Baseline

Further tests were performed later after submitting the final results of both runs along with the baseline. These tests were executed on a smaller version of the TREC 2011 microblogging dataset to check how HIRKM will perform when adjusting different values to the three main tweet features of the method. The smaller version of the dataset contained only 430,396 randomly selected tweets out of the 16,123,041 tweets from the original dataset. The results of these tests differ for each run depending on the value that is given to each tweet feature. Out of three runs, there was only one run that successfully performed better than the baseline. This run was based on HIRKM2.0 which the MAP performance measure improved by 8.17% and the P30 measure increased by 5.36%. A general understanding was reached that HIRKM can perform better if each tweet feature is given the right value to acquire the best performance possible for the method. However, in order to find the right value of each tweet feature then more tests are needed until the right value is found. Also, similar tests must go through the original dataset in order to obtain accurate results of the HIRKM method.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Through series of experiments and studies on Twitter, this thesis attempts to perform an accurate ranking method to rank the tweets based on their authority. In addition, a second goal for this thesis was to find the influential Twitterers based on the new proposed ranking system that this method of this thesis proposed. This thesis succeeded on achieving both of these goals. However, there are some conditions that need attention regarding the influential Twitterers goal. Both of these goals will be explained in details in this section.

As the results in chapter 5 presented, HIRKM2.0 was successful on outperforming the baseline which caused a better results for the new re-ranking scores of the tweets. However, the improved results were not great as they were expected to be. Different views were considered regarding this matter and how to find a way to improve these results to even more. The proposed method, HIRKM, already proved with the help of the tweet features that it can ranked the tweets based on their authority. But were the chosen three features enough for that task? That is an important question that needs to be considered. The reason why

the hash tags, URLs and retweets were only chosen for this method is due to the limitation that TREC restricted for the 2011 microblogging dataset. There are other important tweet futures that can be used to improve HIRKM even more. But the only way to obtain these features is through Twitter's API which was not an available option for this thesis. With the dataset that was used in this study, all the available features in the tweets were used and found helpful for HIRKM. So to answer the previous question: Yes, the three tweet features were enough to improve the baseline ranking. However, HIRKM can perform even better if it is possible to add the other tweet features that are not available in the dataset. Here is an explanation of how the three chosen tweet features supported HIRKM:

1. Hash tags: Checking the content of the tweets for relevant hash tags helped recognizing if tweets are relevant to the query topics.

2. URLs: Checking how popular the URLs are by examining their sources will indicate how authoritative these sources are.

3. Retweets: Checking if the tweets are from an original source/author or not.

In brief, the hash tags helps recognizes topic relevancy, URLs helps indicating the web authoritative sources and finally retweets helps checking the originality of the tweets.

Moving to the second goal, how to find the influential Twitterers. It was tough to reach that goal with the current microblogging dataset. Based on the information in chapter 5, the golden standard goal is to rank the tweets perfectly as much as possible based on the 50 topics. Recognizing influential Twitterers was not an objective to golden standard. In order to reach that goal using this microblogging dataset, an assumption was made that all the authors of the tweets in the golden standard were influential Twitterers. So the BM25 baseline results ranked relevant tweets written by influential Twitterers. When these results were re-ranked by HIRKM2.0, the tweets became more authoritative which also means that the authors became more influential than the ones that

were introduced in the baseline as well. The reason this assumption was made is because HIRKM already proved how important the tweet features are in terms of ranking tweets. By having an author whom writes original tweets using hash tags for topic relevancy and using URLs from authoritative web sources is applicable to be called influential Twitterer.

6.2 Future Work

Numbers of certain factors were noticed that can significantly improve the performance of HIRKM and develop it to the better in the future. This section of this thesis will explain some of these factors and what they can do to improve the results of the proposed method.

In this thesis, pre-processing the TREC 2011 microblog dataset did not take part for this study. This step felt to be unnecessary because removing any kind of characters from the tweet can cause problems for HIRKM to identify the tweet features. A tweet can only contain 140 characters so every character within the tweet is important. However, if the pre-processing the dataset can be done without harming any of the tweet features then this step is worth considering being part of the future plans for HIRKM.

Using other tweet features from other sources like Twitter's API can also as well improve the performance of HIRKM. The only problem with this step is not a lot of people have unlimited access to Twitter's API. This type of access is very limited and cannot be given to anyone to avoid privacy issues. Here are some of the tweet features that are not available in the dataset but it can help the performance of HIRKM: Number of retweets, number of favorite tweets, number of the Twitterer followers and their followees.

Lastly, readjusting the current tweet features values of HIRKM's tweet features to acquire the best performance. This can be accomplished by running different HIRKM runs by giving each tweet feature a different value in each of those runs. This part already been explained in the end of chapter 6 and how the

results can improve by finding the right value to each tweet feature. However, this step can be time consuming and may take weeks to accomplish the best performance and that is why it was only performed on a smaller version of the dataset. Time was a big concern for this study so that is why it was decided to move this step as one of the parts of the future work in this thesis.

Bibliography

[1] I. Celik, F. Abel and P. Siehndel, "Towards a Framework for Adaptive Faceted Search on Twitter," in *Dynamic and Adaptive Hypertext*, Eindhoven, 2011.

[2] J. Weng, E.-P. Lim, J. Jiang and Q. He, "TwitterRank: Finding Topic-sensitive Influential Twitterers," in *WSDM*, New York, 2010.

[3] B. Suh, Hong Lichan, P. Pirolli and E. H. Chi, "Want to be Retweeted? Large Scale Analytics on Factors Impacting Retweet in Twitter Network," in *IEEE*, Minneapolis, 2010.

[4] S. Petrovic, M. Osborne and V. Lavrenko, "RT to Win! Predicting Message Propagation in Twitter," in *ICWSM*, Barcelona, 2011.

[5] d. boyd, S. Golder and G. Lotan, Tweet, Tweet, Retweet: Conversational Aspects of Retweeting on Twitter, Kauai: HICSS-43. IEEE, 2010.

[6] I. Soboroff, D. McCullough, J. Lin, C. Macdonald, I. Ounis and R. McCreadie, Evaluating Real-Time Search Over Tweets, Dublin: ICWSM, 2012.

[7] R. M. McCreadie, I. Soboroff, J. Lin, C. Macdonald, I. Ounis and D. McCullough, On Building a Reusable Twitter Corpus, Portland: SIGIR, 2012.

[8] L. Hong and B. . D. Davison, "Wanted: A Unified Model for Search in Social Media," in *Third ACM International Conference on Web Search and Data Mining (WSDM)*, New York, 2010.

[9] J. Jeon, W. B. Croft and J. H. Lee, "Finding Similar Questions in Large Question and Answer," in *CIKM '05 Proceedings of the 14th ACM international conference on Information and knowledge management*, New

York, 2005.

[10] H. Kwak, C. Lee, H. Park and S. Moon, "What is Twitter, a Social Network or a News Media?," in *The 19th international conference on World wide web (WWW '10)*, North Carolina, 2010.

[11] G. Stringhini, G. Wang, M. Egeley, C. Kruegel, G. Vigna, H. Zheng and B. Y. Zhao, "Follow the Green: Growth and Dynamics in Twitter Follower Markets," in *Internet Measurment Conference (IMC'13)*, Barcelona, 2013.

[12] G. Stringhini, M. Egele, C. Kruegel and G. Vigna, "Poultry markets: on the underground economy of twitter followers," in *ACM workshop on Workshop on online social networks (WOSN '12)*, Helsinki, 2012.

[13] R. LI, K. H. Lei, R. Khadiwala and K. C.-C. Chang, "TEDAS: a Twitter Based Event Detection and," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference*, Washington, 2012.

[14] M. Beaulieu, M. Gatford, X. J. Huang, S. Robertson, S. Walker and P. Williams, "Okapi at TREC-5," in *In Proceedings of TREC-5*, 1997.

[15] "Wikipedia, The Free Encyclopedia," Wikimedia Foundation Inc., (January 2014). [Online]. Available: http://en.wikipedia.org/wiki/Foreign_key.

[16] "Wikipedia, The Free Encyclopedia," Wikimedia Foundation Inc., January) (2014. [Online]. Available: http://en.wikipedia.org/wiki/Gold_standard_(test).

[17] "TREC Microblog Track," Text REtrieval Conference, [Online]. Available: https://sites.google.com/site/microblogtrack/2011-guidelines.

[18] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald and C. Lioma, "Terrier: A High Performance and Scalable Information," in *In Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

[19] "Terrier IR Platform," Terrier, [Online]. Available: http://terrier.org/.

[20] "Configuring Indexing in Terrier," Terrier, [Online]. Available: http://terrier.org/docs/v3.5/configure_indexing.html.

[21] "The Free Dictionary By Farlex," [Online]. Available: http://www.thefreedictionary.com/Data+preprocessing.

[22] S. K. R and R. Krishnamoorthi, "Data Preprocessing and Easy Access Retrieval of Data through Data Ware House," in *Proceedings of the World Congress on Engineering and Computer Science (WCECS 2009)*, San Francisco, 2009.

[23] "Tokeniser," Terrier, [Online]. Available: http://terrier.org/docs/v3.5/javadoc/org/terrier/indexing/tokenisation/Tokeniser.html.

[24] "English Tokeniser," Terrier, [Online]. Available: http://terrier.org/docs/v3.5/javadoc/org/terrier/indexing/tokenisation/EnglishTokeniser.html.

[25] "Wikipedia, The Free Encyclopedia," Wikimedia Foundation Inc., (January 2014). [Online]. Available: http://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

[26] S. P. Harter, "A Probabilistic Approach to Automatic Keyword Indexing Part II," *Journal of the American Society for Infomration Science,* vol. 26, no. 5, pp. 280-289, 1975.

[27] E. M. Ruiz, "Experiments on Genomics Ad Hoc Retrieval," in *Proceedings of the 14th Text Retrieval Conference*, 2005.

[28] E. S. Robertson and S. Walker, "Some Simple Effective Approximations to the 2-Position Method for Probabalistic Weighted Retrieval," in *17th Annual International ACM SIGIR Conference on Research and Development in Infomration Retrieval (SIGIR'94)*, 1994.

[29] " Find a Tech Definition," WhatIs.com, [Online]. Available: http://whatis.techtarget.com/definition/unique-identifier-UID.

[30] "Why MySQL?," MySQL, [Online]. Available: http://www.mysql.com/why-mysql/.

[31] "trec_eval," TREC, (July 2009). [Online]. Available: http://trec.nist.gov/trec_eval/.

[32] K. M. Svore and C. J. C. Burges, "A Machine Learning Approach for Improved BM25 Retrieval," in *Microsoft Research, Microsoft, Redmond, Technical Report MSR-TR-2009-92*, 2009.

[33] "Class BM25," Terrier, [Online]. Available: http://terrier.org/docs/current/javadoc/org/terrier/matching/models/BM25.html.

[34] X. Zhou, X. J. Huang and B. He, "Enhancing ad-hoc relevance weighting using probability density estimation," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval (SIGIR'11)*, 2011.

[35] J. Zhao, X. J. Huang and Z. Ye, "Modeling Term Associations for Probabilistic Information Retrieval," to appear in *ACM Transactions on Information Systems (TOIS)*. ACM Publisher. April 2014.

[36] X. Yin, X. J. Huang, Z. Li and X. Zho, "A Survival Modeling Approach to Biomedical Search Result Diversification Using Wikipedia," in *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2013.

[37] X. J. Huang, J. Miao and B. He, "High Performance Query Expansion Using Adaptive Co-training," Information Processing & Management: An International Journal (IPM), 2013.

[38] Z. Ye, X. J. Huang and J. Miao, "A Hybrid Model for Adhoc Information Retrieval," in *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval (SIGIR'12)*, 2012.

[39] J. Miao, X. J. Huang and Z. Ye, "Proximity-based Rocchio's Model for Pseudo Relevance Feedback," in *Proceedings of the 35th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'12)*, 2012.

[40] B. He, X. J. Huang and X. Zhou, "Modeling Term Proximity for Probabilistic Information Retrieval Models," Information Sciences Journal, no. 0020-0255, p. 32, 2011.

[41] X. Yin, X. J. Huang and Z. Li, "Mining and Modeling Linkage Information from Citation Context for Improving Biomedical Literature Retrieval," Information Processing & Management: An International Journal (IPM), p. 32, 2010.

[42] Q. Hu and X. J. Huang, "Passage Extraction and Result Combination for Genomics Information Retrieval," Journal of Intelligent Information Systems (JIIS), vol .34, no. 0925-9902, p. 23, 2010.

Appendix A

TREC Topics

This thesis used the topics given by TREC 2011 Microblog. Below is the complete list of all topics. There were 50 topics but only 49 of them were found relevant for both the baseline and the proposed method of this thesis.

01 BBC World Service staff cuts
02 2022 FIFA soccer
03 Haiti Aristide return
04 Mexico drug war
05 NIST computer security
06 NSA
07 Pakistan diplomat arrest murder
08 Phone hacking British politicians
09 Toyota Recall
10 Egyptian protesters attack museum
11 Kubica crash
12 Assange Nobel peace nomination
13 Oprah Winfrey half-sister
14 Release of "The Rite"

15 Thorpe return in 2012 Olympics
16 Release of "Known and Unknown"
17 White Stripes breakup
18 William and Kate fax save-the-date
19 Cuomo budget cuts
20 Taco Bell filling lawsuit
21 Emanuel residency court rulings
22 Healthcare law unconstitutional
23 Amtrak train service
24 Super Bowl, seats
25 TSA airport screening
26 US unemployment
27 Reduce energy consumption
28 Detroit Auto Show
29 Global warming and weather
30 Keith Olbermann new job
31 Special Olympics athletes
32 State of the Union and jobs
33 Dog Whisperer Cesar Millan's techniques
34 MSNBC Rachel Maddow
35 Sargent Shriver tributes
36 Moscow airport bombing
37 Giffords' recovery
38 Protests in Jordan
39 Egyptian curfew
40 Beck attacks Piven
41 Obama birth certificate
42 Holland Iran envoy recall
43 Kucinich olive pit lawsuit
44 White House spokesman replaced
45 Political campaigns and social media

46 Bottega Veneta

47 Organic farming requirements

48 Egyptian evacuation

49 Carbon monoxide law

50 War prisoners, Hatch Act

Appendix B

Created Hash Tags

This thesis used Hash Tags that were created in a .txt file to help recognizes topic relevancy. Below is the complete list of all the 479 created hash tags and they are followed by the topic number.

01 #BBC #BBCStaff #BBCWorldService #StaffCuts #BBCStaffCuts #BBCWorldServiceStaffCuts

02 #WorldCup #WorldCup2022 #FIFA #FIFAWorldCup #FIFAWorldCup2022 #WorldCupDraw #WC2022 #Qatar #WorldCupQatar #WorldCup2022Qatar #WorldCupQatar2022 #Qatar2022 #Soccer #Football

03 #Haiti #HaitiAristide #AristideReturns #AristideReturn #Aristide #JeanBertrandAristide #HaitiAristideReturns

04 #Mexico #MexicoDrugWar #Drug #Drugs #MexicoDrugs #War #DrugWar

05 #NIST #NISTSecurity #ComputerSecurity #Security #ComputerSecurity #Technology

06 #NSA #NationalSecurityAgency #NationalSecurity #Security #USA #USASecurity

07 #Pakistan #PakistanDiplomat #PakistanMurder #PakistanArrest #Polotics #Murder #DiplomatArrest #DiplomatMurder #PakistanDiplomatArrestMurder

08 #Britan #UK #Hacking #PhoneHacking #BritishPhoneHacking #BritishPoliticians #Polotics #Spying #Spy #Poloticians #Hack #PhoneHack #BrithishPhoneHack

09 #Toyota #ToyotaRecall #ToyotaCars #ToyotaCar #Car #Cars #Recall #CarRecall #Toyota2011 #Toyota2011Recall #ToyotaCarRecall

10 #Egypt #EgyptProtesters #EgyptProtest #ArabSpring #Spring #Protest #Protesters #EgyptProtestersAttackMuseum #Museum #EgyptMuseum #HosniMubarak #Mubarak #EgyptianArmy #EgyptArmy #Army #Cairo #TahrirSquare

11 #Kubica #KubicaCrash #Crash #RobertKubica #RobertKubicaCrash #F1 #Formula1 #FormulaOne #RondeDiAndoraRally #Andora #RondeDiAndora #Rally #RallyCrash #AndoraRally #AndoraRallyCrash

12 #Nobel #NobelPrize #JulianAssange #Assange #NobelPeacePrize #PeacePrize #NobelNomitation #NobelPrizeNomination #NobelPeacePrizeNomination #AssangeNobelPeaceNomination #WikiLeaks

13 #Oprah #OprahWinfrey #OprahHalfSister #OprahSister

14 #TheRite #Movie #Movies #HollyWood #AnthonyHopkins

15 #Olympics #Olympics2012 #Thorpe #ThorpeReturns #London2012 #London #LondonOlympics2012 #London2012Olympics #IanThorpe #IanThorpeReturns

16 #KnownAndUnknown #DonaldRumsfeld #Book #Books #Memoir #USA #NewYorkTimes #USMilitary #Military

17 #WhiteStripes #WhiteStripesBreakup #WhiteStripesSplitUp #JackWhite #MegWhite #JackAndMegWhite #Breakup #SplitUp #Music #MusicBand

18 #PrinceWilliam #KateMiddleton #WilliamAndKate #RoyalWedding #TheRoyalWedding #London #Fax #SaveTheDateFax #Wedding #BuckinghamPalace #April29

19 #Cuomo #GovernorCuomo #AndrewCuomo #CuomoBudgetCuts #NewYork #NY #NewYorkBudgetCut

20 #TacoBell #TacoBellLawsuit #Food #TacoBellFilling #MeatFilling #BeefFilling #FakeBeef #FakeMeat #Lawsuit #FastFood #FastFoodChain #FastFoodRestaurant #TacoMeatFilling #TacoBeefFilling #Beef #Meat #TacoBellMeat #TacoBellBeef

21 #EmanuelResidencyCourtRuling #ResidencyCourtRuling #ResidencyCourtRulings #EmanuelResidencyCourtRulings #RahmEmanuel #Emanuel #Chicago #ChicagoMayor #SupremeCourtRuling #Illinois #IllinoisSupremeCourt #SupremeCourt #CourtRuling #CourtRulings

22 #HealthCare #HealthCareLaw #HealthcareLawUnconstitutional #USA #HealthcareUnconstitutional #Obama #ObamaCare

23 #Amtrak #AmtrakTrainService #Train #TrainService #AmtrakService #AmtrakTrain

24 #SuperBowl #SuperBowl2011 #CowboysStadium #Football #AmericanFootball #SuperBowlSeats #SuperBowlXLV #NFL #Unsafe #Safety #Failure #GreenBayPackers #Packers #Cowboys #PittsburghSteelers #Steelers

25 #TSA #TSAScreening #TSAAirport #TSAAirportScreening #AirportScreening #Privacy #TransportatingSecurityAdministration #TSAProgram #TSAPreProgram #TSAPreScreening

26 #USA #Unemployment #USAUnemployment #USUnemployment #Job #Jobs #Work #UnemploymentRate

27 #Energy #EnergyConsumption #ReduceEnergyConsumption #ReduceEnergy #Electricity

28 #Detroit #DetroitAutoShow #AutoShow #NAIAS #Car #Cars #Motors #USA #NorthAmerica #NorthAmericanInternationalAutoShow #Michigan #CoboCenter

29 #GlobalWarming #Weather #World #Earth #Climate

30 #KeithOlbermann #KeithOlbermannNewJob #KeithOlbermannJob #NewJob #TBS #MSNBC #TV #Countdown #Baseball

31 #Olympics #SpecialOlympics #SpecialOlympicsAthletes #Olympics2012 #Olympics2012London #OlympicsLondon2012 #Athletes #SpecialAthletes #London2012 #London #LondonOlympics2012 #London2012Olympics

32 #StateOfTheUnion #StateOfTheUnionAndJobs #Job #Jobs #Employment #Unemployment #Obama #WhiteHouse #USA #Economy

33 #DogWhisperer #Dog #Dogs #DogTraining #DogTrainer #CesarsWay #CesarMillan #Animals #CesarMillan #CesarMillansTechniques #DogWhispererCesarMillansTechniques

34 #MSNBC #NBC #RachelMaddow #MSNBCRachelMaddow #TheRachelMaddowShow #TV

35 #SargentShriver #Shriver #SargentShriverTributes #RIP #RIPSargentShriver #RIPShriver

36 #Moscow #MoscowAirport #MoscowBombing #MoscowAirportBombing #Domodedovo #DomodedovoAirport #DomodedovoAirportBombing #Airport #DomodedovoInternationalAirport #UUDD

37 #Giffordsrecovery #GabrielleGiffords #Giffords #GiffordsReturns #Recovery

38 #Jordan #Protests #ArabSpring #Spring #JordanProtests #Protesters #Amman

39 #Egypt #EgyptianCurfew #Curfew #HosniMubarak #Mubarak #EgyptianArmy #EgyptArmy #Army #Cairo #TahrirSquare #ArabSpring #Spring

40 #GlennBeck #Beck #FrancesPiven #FrancesFoxPiven #Piven #TheBlaze #TV

41 #Obama #BarackObama #ObamaBirthCertificate #WhiteHouse #BirthCertificate #Hawaii #USA #Honolulu

42 #Holland #Netherlands #TheNetherlands #Iran #Envoy #Recall #EnvoyRecall #HollandIranEnvoyRecall #Tehran #SahraBahrami #ZahraBahrami #Bahrami #EvinPrison

43 #Kucinich #DennisKucinich #Clevland #USA #OlivePit #OlivePitLawsuit #KucinichOlivePitLawsuit #Lawsuit #Sandwich #Olive

44 #WhiteHouse #WhiteHouseSpokesman #USA #Washington #WhiteHouseSpokesmanReplaced #SpokesmanReplaced #Spokesman #JayCarney #Carney #RobertGibbs #Gibbs #JoeBiden #Biden

45 #Politics #Campaign #Campaigns #PoliticalCampaigns #SocialMedia #Twitter #Facebook #PoliticalCampaignsAndSocialMedia #Elections #Vote #Voters #Technology

46 #Bottega #Veneta #BottegaVeneta #Fashion #Clothes #Shoes #Handbags #Designer #Luxury #Gucci #Italy #Shop #Shopping #Vicenza #Leather #Jewelry #Fragrance #Bags #Accessories

47 #Organic #Frame #FramingRequirements #OrganicFarmingRequirements #OrganicFarming #Agriculture #OrganicAgriculture

48 #Egypt #Evacuation #EgyptianEvacuation #ArabSpring #Spring #Cairo #HosniMubarak #Mubarak #EgyptianArmy #EgyptArmy #Army #TahrirSquare

49 #Carbon #Monoxide #CarbonMonoxide #CarbonMonoxideLaw #Law

50 #War #Prison #Prisoners #WarPrisoners #HatchAct #WarPrisonersHatchAct

Appendix C

MySQL Tables

MySQL tables were created for this thesis in order to store some information that was needed to conduct the research contained within. Below are the SQL needed to create all the tables in (**ahmed_thesis**) MySQL database.

C.1 tweetfeatures Table

```
CREATE TABLE `tweetfeatures` (
  `tweetFeatureID` bigint(17) unsigned NOT NULL AUTO_INCREMENT,
  `tweet_id` bigint(17) unsigned NOT NULL,
  `featureType` int(1) unsigned NOT NULL COMMENT '0 = hastag\n1 = URL\n2 = Retweet',
  `featureText` varchar(255) NOT NULL,
  `featureValue` double DEFAULT NULL,
  PRIMARY KEY (`tweetFeatureID`),
```

```
  UNIQUE KEY `tweetFeatureID_UNIQUE` (`tweetFeatureID`),
  KEY `tweet_id` (`tweet_id`),
  CONSTRAINT `tweetfeatures_ibfk_1` FOREIGN KEY (`tweet_id`) REFERENCES `twitterdataset` (`tweet_id`)
) ENGINE=InnoDB AUTO_INCREMENT=5107788 DEFAULT CHARSET=latin1;
```

C.2 twitterdataset Table

```
CREATE TABLE `twitterdataset` (
  `tweet_id` bigint(17) unsigned NOT NULL,
  `username` varchar(255) NOT NULL,
  `status` int(3) unsigned DEFAULT NULL,
  `created_at` datetime DEFAULT NULL,
  `tweet` mediumtext,
  PRIMARY KEY (`tweet_id`),
  UNIQUE KEY `tweet_id_UNIQUE` (`tweet_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Appendix D

Programming Code

The following java code was used as part of the experimentation process throughout this thesis. Only the main files are included, and minor code changes are required in order to receive the data needed for each run. In addition, some changes to the code have been made to protect sensitive information, such as usernames and passwords.

D.1 Databse Package

D.1.1 DataInsertion Class

```
//The DataInsertion class is part of the Database package
package database;
//Import statements to external sources necessary for the class to function
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import org.apache.commons.lang3.StringEscapeUtils;
```

```
import hirkm.sql.MySQL;

//The DataInsertion class
public class DataInsertion
{
	//Global variables accessible to entire class
	private static String databaseUsername = "USERNAME",
	databasePassword = "PASSWORD",
	databaseConnectivity = "jdbc:mysql://localhost/ahmed_thesis";

	//The main method
	public static void main(String[] args)
	{
	//Create a local file variable linking to a file located on our storage medium
	File datasetFile = new File("./ExternalFiles/statistics");

	//Call the readFile method to read our file
	readFile(datasetFile);
	}

	/*
	 * The readFile method
	 *
	 * Description:
	 * The method reads a file that was passed into it, then calls the insertToDatabase
	method to take the contents of the file and put it into a database
	 *
	 * Parameters:
	* fileToRead - File parameter which is needed to read a file
	*/
	private static void readFile(File fileToRead)
	{
	//Local variable
	String line;

	//Attempt to do the following
	try
	{
	//Create a BufferedReader instance and a FileReader instance in order to read the
	file
	BufferedReader bufferedReader = new BufferedReader(new
	FileReader(fileToRead));

	//Create a connection to our database
	Connection connection = MySQL.connectToDatabase(databaseUsername,
	databasePassword, databaseConnectivity);
```

```
//Perform the following until the file we are reading has no more lines to read
while((line = bufferedReader.readLine()) != null)
//Call the insertToDatabase method to insert the line that was read
insertToDatabase(connection, line);

//Commit our database changes and close the connection
connection.commit();
connection.close();
}
//Perform the following if a FileNotFoundException was encountered
catch(FileNotFoundException e)
{
//Show the stack trace for the error in the console
e.printStackTrace();

//Stop the program
System.exit(0);
}
//Perform the following if a IOException was encountered
catch(IOException e)
{
//Show the stack trace for the error in the console
e.printStackTrace();

//Stop the program
System.exit(0);
}
//Perform the following if a SQLException was encountered
catch(SQLException e)
{
//Show the stack trace for the error in the console
e.printStackTrace();

//Stop the program
System.exit(0);
}
}

/*
 * The insertToDatabase method
 *
 * Description:
 * The method takes a given line from the file, tokenizes it to appropriate    tokens,
does formatting on the date, and stores it in the database
 *
```

```
 * Parameters:
 * databaseConnection - The connection needed to store our data
 * lineToInsert - The line that was read from our file that needs to be
tokenized, formatted and then inserted to our database
 */
private static void insertToDatabase(Connection databaseConnection, String
lineToInsert)
{

//Local variables
String[] tokens = lineToInsert.split("\t");
String[] dateTokens = tokens[3].split(" ");
String formattedDate;

//If the number of tokens that was created for the date was not equal to 1
(meaning that it was not null), perform the following
if(dateTokens.length != 1)
//Format the date to a proper MySQL datetime entry
formattedDate = "\"" + dateTokens[5] + "-" +
getNumericMonth(dateTokens[1]) + "-" + dateTokens[2] + " " +     dateTokens[3]
+ "\"";
//If the number of tokens that was created for the date was equal to 1
(meaning that it was null), perform the following
else
//Make the date equal to null (what it already comes as from the file)
formattedDate = tokens[3];

//Escape the tweet in order to avoid any encapsulation errors in Java
tokens[4] = StringEscapeUtils.escapeJava(tokens[4]);

//Generate the insert statement to be used to enter our line to the     database
String insertStatement = "INSERT INTO twitterDataset (tweet_id,  username,
status, created_at, tweet) VALUES (" + tokens[0] + ", \"" +           tokens[1] +
"\", " + tokens[2] + ", " + formattedDate + ", \"" + tokens[4] +      "\");";

//Try to perform the following
try
        {
        //Create a PreparedStatement and execute our insert statement on
        the database
        PreparedStatement preparedStatement =
databaseConnection.prepareStatement(insertStatement);
        preparedStatement.executeUpdate();
        preparedStatement.close();
        }
        //Perform the following if a SQLException error was encountered
```

```
        catch(SQLException e)
        {
        //Show the stack trace on the console
        e.printStackTrace();

        //Stop the program
        System.exit(0);
        }
}

/*
 * The getNumericMonth method
 *
 * Description:
 * This converts the 3 character month value into a double digit month value
 *
 * Parameters:
 * textualMonth - The 3 character month representation text
 */
private static String getNumericMonth(String textualMonth)
{
//Local variable
String formattedDate = "";

//Determine which month was submitted, and assign our local variable the
corresponding double digit month code
if(textualMonth.equals("Jan"))
formattedDate = formattedDate + "01";
else if(textualMonth.equals("Feb"))
        formattedDate = formattedDate + "02";
else if(textualMonth.equals("Mar"))
        formattedDate = formattedDate + "03";
else if(textualMonth.equals("Apr"))
        formattedDate = formattedDate + "04";
else if(textualMonth.equals("May"))
        formattedDate = formattedDate + "05";
else if(textualMonth.equals("Jun"))
        formattedDate = formattedDate + "06";
else if(textualMonth.equals("Jul"))
        formattedDate = formattedDate + "07";
else if(textualMonth.equals("Aug"))
        formattedDate = formattedDate + "08";
else if(textualMonth.equals("Sep"))
        formattedDate = formattedDate + "09";
else if(textualMonth.equals("Oct"))
        formattedDate = formattedDate + "10";
```

```
        else if(textualMonth.equals("Nov"))
                formattedDate = formattedDate + "11";
        else if(textualMonth.equals("Dec"))
                formattedDate = formattedDate + "12";
        //If no month was detected, perform the following
        else
        {
        //Show this error in the console
        System.err.println("NO MONTH FOUND");

        //Stop the program
        System.exit(0);
        }

        //Return the double digit month representation
        return formattedDate;
        }
}
```

D.1.2 ExtractTweetFeatures Class

```
//The ExtractTweetFeatures class is part of the Database package
package database;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import hirkm.ranks.AlexaRank;
import hirkm.ranks.PageRank;
import hirkm.sql.MySQL;
import hirkm.http.Requests;
import hirkm.inputs.DateAndTime;

//The ExtractTweetFeatures class
public class ExtractTweetFeatures
{
```

```
//Global variables accessible to entire class
private static final String DATABASE_USERNAME = "USERNAME",
DATABASE_PASSWORD = "PASSWORD",
DATABASE_CONNECTIVITY =
"jdbc:mysql://127.0.0.1/ahmed_thesis?autoReconnect=true";
private static final int START_ID = -1,
END_ID = 16123041;
private static String urlText, featureID;

//Set the date where the program begins to be run
private static final Date startDate = new Date();

//The extractHashtag method
private static void extractHashtag() throws InstantiationException,
IllegalAccessException, ClassNotFoundException
{
//Try to perform the following
try
{
//Create a database connection
Connection connection =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);
//Tell the user a database connection was made
System.out.println("Database connection established for extracting Hashtags.");

//Inform the user that the extraction of hashtags is beginning
System.out.println("Beginning to extract hashtags from tweets.");

//Prepare the query that will get the tweets from the database
String query = "SELECT tweet_id, tweet FROM twitterDataset;";

//Get the query results from the database
ResultSet resultSet = MySQL.getSelectResultsFromDatabase(connection, query);

//Loop through all the results returned from the database
while(resultSet.next())
{
//Perform the following if the tweet is not null
if(resultSet.getString("tweet") != null)
{
//Set the regular expression pattern to look for that will find hashtags in the tweet
Pattern pattern = Pattern.compile("(#[^0-9][a-zA-Z0-9_]+)");

//Match the established pattern with the actual tweet
Matcher matcher = pattern.matcher(resultSet.getString("tweet"));
```

```
//Perform the following for all matches found in the tweet
while(matcher.find())
{
//Set the tweet_id and actual hashtag to local variables
String tweet_id = resultSet.getString("tweet_id"),
hashTag = matcher.group();

//If the extracted hashtag is longer than 255 characters, perform the following
if(hashTag.length() > 255)
//Make the hashtag 255 charters long
hashTag = hashTag.substring(0, 255);

//Create a database connection
Connection connection2 =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);

//Create an insert statement that will put the extracted hashtag into the database
along with the tweet_id that it came from
PreparedStatement insertStatement = connection2.prepareStatement("INSERT
INTO tweetFeatures (tweet_id, featureType, featureText)" + " VALUES (?, 0,
?);");

//Put the tweet_id in the first unknown section of the insert statement
insertStatement.setString(1, tweet_id);

//Put the hashtag in the second unknown section of the insert statement
insertStatement.setString(2, hashTag);

//Execute the insert statement
insertStatement.executeUpdate();

//Close the insert statement
insertStatement.close();

//Commit the changes and close the connection
connection2.commit();
connection2.close();
}
}
}

//Close the resultset from the database
resultSet.close();
```

```
//Close the database connection
connection.close();

//Tell the user a database connection is closed
System.out.println("Database connection closed for extracting Hashtags.");

//Tell the user that the method has completed executing
System.out.println("Finished extracting hashtags from tweets.");
}
//Catch any SQLException
catch(SQLException sqlException)
{
//Inform the user that establishing a database connection failed
System.err.println("Establishing database connection failed for Hashtags!");

//Show the error
sqlException.printStackTrace();

//Get the time and how long the program was running when the exception was
caught
System.err.println(DateAndTime.getDateDifference(startDate, new Date()));

//Terminate the program
System.exit(1);
}
}

//The extractURL method
private static void extractURL() throws InstantiationException,
IllegalAccessException, ClassNotFoundException
{
try
{
//Create a database connection
Connection connection =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);

//Tell the user a database connection was made
System.out.println("Database connection established for extracting URLs.");

//Inform the user that the extraction of URLs is beginning
System.out.println("Beginning to extract URLs from tweets.");

//Prepare the query that will get the tweets from the database
String query = "SELECT tweet_id, tweet FROM twitterDataset;";
```

```
//Get the query results from the database
ResultSet resultSet = MySQL.getSelectResultsFromDatabase(connection, query);

//Loop through all the results returned from the database
while(resultSet.next())
{
//Perform the following if the tweet is not null
if(resultSet.getString("tweet") != null)
{
//Regular expression for finding URLs
String regex = "((([A-Za-z]{3,9}:(?:\\/\\/)?)(?:[-;:&=\\+\\$,\\w]@)?[A-Za-z0-9.-
]+|(?:www.|[-;:&=\\+\\$,\\w]+@)[A-Za-z0-9.-]+)((?:\\/[\\+~%\\/.\\w-_]*)?\\??(?:[-
\\+=&;%@.\\w_]*)#?(?:[.\\!\\/\\\\w]*))?|(([A-Za-z0-9_])+[.]([A-Za-z0-
9_])+[.]([A-Za-z0-9_])+)|(([A-Za-z0-9])+([.])([A-Za-z0-9])+))";

//Set the regular expression pattern to look for that will find URLs in the tweet
Pattern pattern = Pattern.compile(regex);
//Match the established pattern with the actual tweet
Matcher matcher = pattern.matcher(resultSet.getString("tweet"));

//Perform the following for all matches found in the tweet
while(matcher.find())
{
//Set the tweet_id and actual url to local variables
String tweet_id = resultSet.getString("tweet_id"),
url = matcher.group();

//If the url is less than 7 characters, perform the following
if(url.length() < 7)
//Change the string to "NOT A URL"
url = "NOT A URL";
//If the URL does not start with the string "http://", perform the following
else if(!url.substring(0, 7).equals("http://"))
//Change the string to "NOT A URL"
url = "NOT A URL";

//If the extracted URL is longer than 255 characters, perform the following
if(url.length() > 255)
//Make the URL 255 charters long
url = url.substring(0, 255);

//If the URL is a real URL and if there are no duplicates of the ':' character,
perform the following
if(!url.equals("NOT A URL") /*&& checkURLStatus(url)*/ &&
!checkForDuplicate(url, ':'))
```

```
{
//Create a database connection
Connection connection2 =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);

//Create an insert statement that will put the extracted url into the database along
with the tweet_id that it came from
PreparedStatement insertStatement = connection2.prepareStatement("INSERT
INTO tweetFeatures (tweet_id, featureType, featureText)" + " VALUES (?, 1,
?);");

//Put the tweet_id in the first unknown section of the insert statement
insertStatement.setString(1, tweet_id);

//Put the URL in the second unknown section of the insert statement
insertStatement.setString(2, url);

//Execute the insert statement
insertStatement.executeUpdate();

//Close the insert statement
insertStatement.close();

//Commit the changes and close the connection
connection2.commit();
connection2.close();
}
}
}
}

//Close the resultset from the database
resultSet.close();

//Close the database connection
connection.close();

//Tell the user that the method has completed executing
System.out.println("Finished extracting URLs from tweets.");
}
//Catch any SQLException
catch(SQLException sqlException)
{
//Inform the user that establishing a database connection failed
System.err.println("Establishing database connection failed for URLs!");
```

```
//Show the error
sqlException.printStackTrace();

//Show the time and how long the program was running before the error was
caught
System.err.println(DateAndTime.getDateDifference(startDate, new Date()));

//Terminate the execution of the program
System.exit(1);
}
}

//The checkURLStatus method
private static boolean checkURLStatus(String url)
{
//Create local variable
boolean isTrueURL = false;

//Try to perform the following
try
{
//If the URL that is checked returns HTTP code 200, perform the following
if(Requests.getResponseCode(url) == 200)
//Set the isTrueURL variable to true
isTrueURL = true;
else
//Set the isTrueURL variable to false
isTrueURL = false;
}
catch(Exception e)
{
System.err.println("Caught IOException...ignorning");
return isTrueURL;
}

//Return the isTrueURL variable
return isTrueURL;
}

//The checkforDuplicate method
private static boolean checkForDuplicate(String stringToCheck, char
characterToFind)
{
//Create local variable
int duplicateCount = 0;
```

```
//Loop through the string and perform the following
for(int i = 0 ; i < stringToCheck.length(); i++)
{
//If the string has a matching character, perform the following
if(stringToCheck.charAt(i) == characterToFind)
//Increase the duplicateCount variable by 1
duplicateCount++;
}

//If there is more than one of a given character, perform the following
if(duplicateCount > 1)
//Return true
return true;
//If there is one or less of a given character, perform the following
else
//Return false
return false;
}

//The getFullURL method
private static String getFullURL(String url) throws IOException
{
//Request the real url of any url shortener
String line =
Requests.getRequest("http://www.checkshorturl.com/expand.php?u=" + url);

//Split the code to find the URL
String[] lineArray = line.split("<td style=\"border-bottom: 1px dotted
black;width:750px;height:20px;padding:10px;\"><a href=\"");

//If the number of tokens generated from the split is more than 1, perform the
following
if(lineArray.length > 1)
{
//Split the code to find the URL
lineArray = lineArray[1].split("\" target=\"_blank\" rel=\"nofollow\">");
lineArray = lineArray[0].split("\" title=\"");

//Return the URL
return lineArray[0];
}
//If the number of tokens generated from the split is not more than 1, perform the
following
else
{
```

```
//Return the original URL
return url;
}
}

//The assignZero method
private static void assignZero() throws SQLException, InstantiationException,
IllegalAccessException, ClassNotFoundException
{
//Create a database connection
Connection connection2 =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);

//Create an update statement that will update all instances of the migre.me url
PreparedStatement updateStatement = connection2.prepareStatement("UPDATE
tweetfeatures SET featureValue = ? WHERE featureText LIKE ?;");

//Set the featureValue
updateStatement.setDouble(1, 0);

//Set the featureText
updateStatement.setString(2, urlText);

//Execute the update
updateStatement.executeUpdate();

//Close the update statement
updateStatement.close();

//Commit the changes and close the connection
connection2.commit();
connection2.close();
}

//The executeRanking runnable
private static Runnable executeRanking = new Runnable()
{
@Override
public void run()
{
//Try to perfrom the following
try
{
//Get the real URL of any shortened link
URL url = new URL(getFullURL(urlText));
```

```
//Local variables used to calculate rank
int alexaRank = AlexaRank.getAlexaRank(url.getHost().toString()),
pageRank = PageRank.getPageRank(url.getHost().toString());
double alexaRankNormalized,
pageRankNormalized = pageRank / 10.0;

//Normalize the Alexia Rank
if(alexaRank > 0 && alexaRank < 1001)
alexaRankNormalized = 1.0;
else if(alexaRank > 1000 && alexaRank < 10001)
alexaRankNormalized = 0.8;
else if(alexaRank > 10000 && alexaRank < 100001)
alexaRankNormalized = 0.6;
else if(alexaRank > 100000 && alexaRank < 1000001)
alexaRankNormalized = 0.4;
else if(alexaRank > 1000000 && alexaRank < 4000001)
alexaRankNormalized = 0.2;
else
alexaRankNormalized = 0.0;
//Show the user the regular ranks, normalized ranks, and final rank
System.out.println("AlexaRank: " + alexaRank + " | PageRank: " + pageRank +
"\nAlexaRank_Normalized: " + alexaRankNormalized + " |
PageRank_Normalized: " + pageRankNormalized + " | Final Rank: " +
((alexaRankNormalized + pageRankNormalized) / 2));

//Tell the user the system is updating the records
System.out.print("Updating records...");

//Create a database connection
Connection connection2 =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);

//If the url thats being examined is larger than 14 characters long and is a
migre.me url, perform the following
if(urlText.length() > 14 && (urlText.substring(0, 15).equals("http://migre.me") ||
urlText.substring(0, 13).equals("http://bit.ly")))
{
//Create an update statement that will update all instances of the migre.me url
PreparedStatement updateStatement = connection2.prepareStatement("UPDATE
tweetfeatures SET featureValue = ? WHERE featureText LIKE ?;");

//Set the featureValue
updateStatement.setDouble(1, ((alexaRankNormalized + pageRankNormalized) /
2));
```

```
//Set the featureText
updateStatement.setString(2, urlText);

//Execute the update
updateStatement.executeUpdate();

//Close the update statement
updateStatement.close();
}
//If the previous condition failed, perform the following
else
{
//Create a update statement that will update the URL we just got the rank for
PreparedStatement updateStatement = connection2.prepareStatement("UPDATE
tweetfeatures SET featureValue = ? WHERE tweetFeatureID = ?;");

//Put the rank value in the first unknown section of the update statement
updateStatement.setDouble(1, ((alexaRankNormalized + pageRankNormalized) /
2));

//Put the tweetFeatureID in the second unknown section of the update statement
//updateStatement.setInt(2,
Integer.parseInt(resultSet.getString("tweetFeatureID")));
updateStatement.setInt(2, Integer.parseInt(featureID));

//Execute the update statement
updateStatement.executeUpdate();

//Close the update statement
updateStatement.close();
}

//Commit the changes and close the connection
connection2.commit();
connection2.close();

Thread.currentThread().interrupt();
}
catch(Exception e)
{
e.printStackTrace();
}
}
};
```

```
//The getRanksExecutor method
private static void getRanksExecutor(boolean secondRun) throws
InstantiationException, IllegalAccessException, ClassNotFoundException,
SQLException, InterruptedException
{
//Create a executorservice with a single thread executor
ExecutorService executor = Executors.newSingleThreadExecutor();

//Execute the executor service
executor.execute(executeRanking);

//Shutdown the executor service
executor.shutdown();

//Wait exactly 4 minutes before terminating the executor service
executor.awaitTermination(4, TimeUnit.MINUTES);

//If the executor service has terminated, perform the following
if(executor.isTerminated())
{
//Tell the user the process finished
System.out.print("Finished");
}
//If the executor service did not terminate and its the first run, perform the
following
else if(!executor.isTerminated() && !secondRun)
{
//Tell the user the executor service is taking to long and the program will try again
System.out.println("It is taking to long to process.  Terminating execution and
retrying...");

//Force the executor service to shutdown
executor.shutdownNow();

//Try for a second time
getRanksExecutor(true);
}
//If the other conditions failed, perform the following
else
{
//Tell the user the executor service is taking to long and the program will assign a
rank of 0 to the url
System.out.println("It is taking to long to process.  Terminating and assigning a
rank of 0");

//Force the executor service to shutdown
```

```
executor.shutdownNow();

//Assign a value of 0 to the URL's rank
assignZero();
}
}

private static void getRanks() throws InstantiationException,
IllegalAccessException, ClassNotFoundException, SQLException,
MalformedURLException, IOException, InterruptedException
{
//Create a database connection
Connection connection =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);

//Tell the user a database connection was made
System.out.println("Database connection established...");

//Inform the user that the URL verification has begin
System.out.println("Beginning Ranking Process");

//Prepare the query that will get the URLS
String query = "SELECT * FROM tweetfeatures where featuretype = 1 AND
featureValue IS NULL AND tweetFeatureID > " + START_ID + " AND
tweetFeatureID < " + END_ID + ";";

//Get the query results from the database
ResultSet resultSet = MySQL.getSelectResultsFromDatabase(connection, query);

//Loop through all the results returned from the database
while(resultSet.next())
{
//Records when the main work started
Date start = new Date();

//Get the url from the resultSet
urlText = resultSet.getString("featureText");
featureID = resultSet.getString("tweetFeatureID");
//Inform the user that ranking has started and on what time
System.out.println("\nStarting ranking URL " + urlText + " on " + start);

getRanksExecutor(false);

//Display the time it took for the process to finish for that one url
```

```
System.out.println(ca.hirkm.inputs.DateAndTime.getDateDifference(start, new
Date()));
}

//Close the database connection
connection.commit();
connection.close();
}

//The main method
public static void main(String[] args)
{
//Create a thread for extracting URLs
Thread urlExtractionThread = new Thread()
{
//The run method for the urlExtractionThread
public void run()
{
//Try to perform the following
try
{
//Perform the extractURL method
extractURL();
}
//Catch specific exceptions
catch (Exception e)
{
//Show the error
e.printStackTrace();

//Show the time and how long the program was running when the exception was
caught

System.err.println(DateAndTime.getDateDifference(startDate, new Date()));

//Terminate the program
System.exit(1);
}

//Inform the user of when the program started and when it finished
System.out.println("urlExtractionThread ended!");

System.out.println(DateAndTime.getDateDifference(startDate, new Date()));
}
};
```

```
//Create a thread for extracting Hashtags
Thread hashtagExtractionThread = new Thread()
{
//The run method for the hashtagExtractionThread method
public void run()
{
//Try to perform the following
try
{
//Perform the extractHashtag method
extractHashtag();
}
//Catch any exceptions that may occur
catch (Exception e)
{
//Show the error
e.printStackTrace();

//Show the time and how long the program was running when the exception was
caught

System.err.println(DateAndTime.getDateDifference(startDate, new Date()));

//Terminate the program
System.exit(1);
}

//Inform the user of when the program started and when it finished
System.out.println("hashtagExtractionThread ended!");

System.out.println(DateAndTime.getDateDifference(startDate, new Date()));
}
};

//Create a thread for verifying URLs
Thread verifyURLs = new Thread()
{
//The run method for the verifyURLs method
public void run()
{
try
{
//Create a database connection
```

```
Connection connection =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);

//Tell the user a database connection was made
System.out.println("Database connection estabishled...");

//Inform the user that the URL verification has begin
System.out.println("Beginning URL verification");

//Prepare the query that will get the URLS
String query = "SELECT * FROM tweetfeatures where featuretype = 1 AND
featureValue IS NULL and tweetFeatureID < 380413;";

//Get the query results from the database
ResultSet resultSet = MySQL.getSelectResultsFromDatabase(connection, query);

//Loop through all the results returned from the database
while(resultSet.next())
{
String text = resultSet.getString("featureText");

System.out.println(text);

//If the URL does not work, perform the following

//if(!checkURLStatus(resultSet.getString("featureText")))
if(!checkURLStatus(text))
{
//Create a database connection
Connection connection2 =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);

//Create a delete statement that will delete the URL that do not work
PreparedStatement deleteStatement = connection2.prepareStatement("DELETE
FROM tweetFeatures WHERE tweetFeatureID = ?");

//Put the tweetFeatureID in the first unknown section of the delete statement
deleteStatement.setString(1, resultSet.getString("tweetFeatureID"));

//Execute the delete statement
deleteStatement.executeUpdate();

//Close the insert statement
deleteStatement.close();
```

```
//Commit the changes and close the connection
connection2.commit();
connection2.close();
}
else
{
//Create a database connection
Connection connection2 =
MySQL.connectToDatabase(DATABASE_USERNAME,
DATABASE_PASSWORD, DATABASE_CONNECTIVITY);
//Create a update statement that will update the URL feature row to indicate it has
been verified
PreparedStatement updateStatement = connection2.prepareStatement("UPDATE
tweetfeatures SET featureValue = \"99\" WHERE featureText LIKE ?");

//Put the featureText in the first ? in the updateStatement
updateStatement.setString(1, text);

//Execute the update statement
updateStatement.executeUpdate();

//Close the update statement
updateStatement.close();

//Commit the changes and close the connection
connection2.commit();
connection2.close();
}
}

//Close the ResultSet
resultSet.close();

//Commit and close the database connection
connection.commit();
connection.close();

//Inform the username the database connection is closed and the thread finshed
System.out.println("Database connection closed");
System.out.println("verifyURLs thread ended");

System.out.println(DateAndTime.getDateDifference(startDate, new Date()));
}
catch(Exception e)
{
```

```
//Show the error to the user
e.printStackTrace();
}
}
};

//Get the PageRank and Alexia Rank for the URLS
Thread getRanksThread = new Thread()
{
//The run method for the getRanksThread
public void run()
{
try
{
getRanks();
}
catch(InstantiationException e)
{
e.printStackTrace();
}
catch(IllegalAccessException e)
{
e.printStackTrace();
}
catch(ClassNotFoundException e)
{
e.printStackTrace();
}
catch(SQLException e)
{
e.printStackTrace();
}
catch(MalformedURLException e)
{
e.printStackTrace();
}
catch(IOException e)
{
e.printStackTrace();
}
catch(InterruptedException e)
{
e.printStackTrace();
}
}
};
```

```
        //Start the threads
        //verifyURLs.start();
        //urlExtractionThread.start();
        //hashtagExtractionThread.start();
        getRanksThread.start();
        }
}
```

D.2 HIRKM Package

D.2.1 AlexaRankNotRankingException Class

```
package hirkm;

public class AlexaRankNotRankingException extends IllegalStateException

{
        public AlexaRankNotRankingException()
        {
                super();
        }

        public AlexaRankNotRankingException(String message)
        {
                super(message);
        }
}
```

D.2.2 HashTag Class

```
package hirkm;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import hirkm.sql.MySQL;

public class HashTag
```

```
{
        public static void extractHashtag(String databaseUsername, String
        databasePassword, String databaseConnectivity) throws InstantiationException,
        IllegalAccessException, ClassNotFoundException, SQLException
        {
        //Create a database connection
        Connection connection = MySQL.connectToDatabase(databaseUsername,
        databasePassword, databaseConnectivity);

        //Tell the user a database connection was made
        System.out.println("Database connection established for extracting Hashtags.");

        //Inform the user that the extraction of hashtags is beginning
        System.out.println("Beginning to extract hashtags from tweets.");

        //Prepare the query that will get the tweets from the database
        String query = "SELECT tweet_id, tweet FROM twitterDataset;";

        //Get the query results from the database
        ResultSet resultSet = MySQL.getSelectResultsFromDatabase(connection, query);

        //Loop through all the results returned from the database
        while(resultSet.next())
        {
        //Perform the following if the tweet is not null
        if(resultSet.getString("tweet") != null)
        {
        //Set the regular expression pattern to look for that will find hashtags in the tweet
        Pattern pattern = Pattern.compile("(#[^0-9][a-zA-Z0-9_]+)");

        //Match the established pattern with the actual tweet
        Matcher matcher = pattern.matcher(resultSet.getString("tweet"));

        //Perform the following for all matches found in the tweet
        while(matcher.find())
        {
        //Set the tweet_id and actual hashtag to local variables
        String tweet_id = resultSet.getString("tweet_id"),
        hashTag = matcher.group();

        //If the extracted hashtag is longer than 255 characters, perform the following
        if(hashTag.length() > 255)
        //Make the hashtag 255 charters long
        hashTag = hashTag.substring(0, 255);
        //Create a database connection
```

```
        Connection connection2 = MySQL.connectToDatabase(databaseUsername,
        databasePassword, databaseConnectivity);

        //Create an insert statement that will put the extracted hashtag into the database
        along with the tweet_id that it came from
        PreparedStatement insertStatement = connection2.prepareStatement("INSERT
        INTO tweetFeatures (tweet_id, featureType, featureText)" + " VALUES (?, 0,
        ?);");

        //Put the tweet_id in the first unknown section of the insert statement
        insertStatement.setString(1, tweet_id);

        //Put the hashtag in the second unknown section of the insert statement
        insertStatement.setString(2, hashTag);

        //Execute the insert statement
        insertStatement.executeUpdate();

        //Close the insert statement
        insertStatement.close();

        //Commit the changes and close the connection
        connection2.commit();
        connection2.close();
        }
        }
        }

        //Close the resultset from the database
        resultSet.close();

        //Commit and close the database connection
        connection.commit();
        connection.close();

        //Tell the user a database connection is closed
        System.out.println("Database connection closed for extracting Hashtags.");

        //Tell the user that the method has completed executing
        System.out.println("Finished extracting hashtags from tweets.");
        }
}
```

D.2.3 HIRKM_Runner Class

```
package hirkm;

import java.sql.SQLException;

public class HIRKM_Runner
{
	public static void main(String[] args)
	{
	final String DATABASE_USERNAME = "USERNAME",
	DATABASE_PASSWORD = "PASSWORD",
	DATABASE_CONNECTIVITY =
	"jdbc:mysql://127.0.0.1/ahmed_thesis?autoReconnect=true";
	final int START_ID = -1,
	END_ID = 16123041;

	try
	{
	Rank.getRanks(DATABASE_USERNAME, DATABASE_PASSWORD,
	DATABASE_CONNECTIVITY, START_ID, END_ID);
	}
	catch(InstantiationException e)
	{
	e.printStackTrace();
	}
	catch(IllegalAccessException e)
	{
	e.printStackTrace();
	}
	catch(ClassNotFoundException e)
	{
	e.printStackTrace();
	}
	catch(SQLException e)
	{
	e.printStackTrace();
	}
	catch(InterruptedException e)
	{
	e.printStackTrace();
	}
	}
}
```

D.2.4 Rank Class

```
package hirkm;

import java.io.IOException;
import java.net.MalformedURLException;
import java.net.SocketException;
import java.net.URL;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import hirkm.ranks.AlexaRank;
import hirkm.ranks.PageRank;
import hirkm.sql.MySQL;

public class Rank
{
	private static String urlText, featureID, databaseUsername, databasePassword,
	databaseConnectivity;

	public static void getRanks(String databaseUsername, String databasePassword,
	String databaseConnectivity, int startID, int endID) throws SQLException,
	InstantiationException, IllegalAccessException, ClassNotFoundException,
	InterruptedException
	{
	Rank.databaseUsername 	= databaseUsername;
	Rank.databasePassword 	= databasePassword;
	Rank.databaseConnectivity = databaseConnectivity;

	//Create a database connection
	Connection connection = MySQL.connectToDatabase(databaseUsername,
	databasePassword, databaseConnectivity);

	//Tell the user a database connection was made
	System.out.println("Database connection established...");

	//Inform the user that the URL verification has begin
	System.out.println("Beginning Ranking Process");

	//Prepare the query that will get the URLS
```

```
String query = "SELECT * FROM tweetfeatures where featuretype = 1 AND
featureValue IS NULL AND tweetFeatureID > " + startID + " AND
tweetFeatureID < " + endID + ";";

//Get the query results from the database
ResultSet resultSet = MySQL.getSelectResultsFromDatabase(connection, query);

//Loop through all the results returned from the database
while(resultSet.next())
{
//Records when the main work started
Date startDate = new Date();

//Get the url from the resultSet
urlText = resultSet.getString("featureText");
featureID = resultSet.getString("tweetFeatureID");

//Inform the user that ranking has started and on what time
System.out.println("\nStarting ranking URL " + urlText + " on " + startDate);

getRanksExecutor(false, databaseUsername, databasePassword,
databaseConnectivity);

//Display the time it took for the process to finish for that one url

System.out.println(hirkm.inputs.DateAndTime.getDateDifference(startDate, new
Date()));
}

//Close the database connection
connection.commit();
connection.close();

System.out.println("Finished Ranking Process");
}

private static void getRanksExecutor(boolean secondRun, String
databaseUsername, String databasePassword, String databaseConnectivity)
throws InterruptedException, InstantiationException, IllegalAccessException,
ClassNotFoundException, SQLException
{
//Create a executor service with a single thread executor
ExecutorService executor = Executors.newSingleThreadExecutor();

//Execute the executor service
executor.execute(executeRanking);
```

```
//Shutdown the executor service
executor.shutdown();

//Wait exactly 4 minutes before terminating the executor service
executor.awaitTermination(4, TimeUnit.MINUTES);

//If the executor service has terminated, perform the following
if(executor.isTerminated())
{
//Tell the user the process finished
System.out.print("Finished");
}
//If the executor service did not terminate and its the first run, perform the
following
else if(!executor.isTerminated() && !secondRun)
{
//Tell the user the executor service is taking to long and the program will try again
System.out.println("It is taking to long to process.  Terminating execution and
retrying...");

//Force the executor service to shutdown
executor.shutdownNow();

//Try for a second time
getRanksExecutor(true, databaseUsername, databasePassword,
databaseConnectivity);
}
//If the other conditions failed, perform the following
else
{
//Tell the user the executor service is taking to long and the program will assign a
rank of 0 to the url
System.out.println("It is taking to long to process.  Terminating and assigning a
rank of 0");

//Force the executor service to shutdown
executor.shutdownNow();

//Assign a value of 0 to the URL's rank
assignZero(databaseUsername, databasePassword, databaseConnectivity);
}
}
```

```
private static void assignZero(String databaseUsername, String
databasePassword, String databaseConnectivity) throws InstantiationException,
IllegalAccessException, ClassNotFoundException, SQLException
{
//Create a database connection
Connection connection = MySQL.connectToDatabase(databaseUsername,
databasePassword, databaseConnectivity);

//Create an update statement that will update all instances of the migre.me url
PreparedStatement updateStatement = connection.prepareStatement("UPDATE
tweetfeatures SET featureValue = ? WHERE featureText LIKE ?;");

//Set the featureValue
updateStatement.setDouble(1, 0);

//Set the featureText
updateStatement.setString(2, urlText);

//Execute the update
updateStatement.executeUpdate();

//Close the update statement
updateStatement.close();

//Commit the changes and close the connection
connection.commit();
connection.close();
}

private static Runnable executeRanking = new Runnable()
{
@Override
public void run()
{
//Try to perfrom the following
try
{
//Get the real URL of any shortened link
URL url = new URL(UniformResourceLocator.getFullURL(urlText));

//Local variables used to calculate rank
int alexaRank = AlexaRank.getAlexaRank(url.getHost().toString()),
pageRank = PageRank.getPageRank(url.getHost().toString());
double alexaRankNormalized,
pageRankNormalized = pageRank / 10.0;
```

```
//Normalize the Alexia Rank
if(alexaRank > 0 && alexaRank < 1001)
alexaRankNormalized = 1.0;
else if(alexaRank > 1000 && alexaRank < 10001)
alexaRankNormalized = 0.8;
else if(alexaRank > 10000 && alexaRank < 100001)
alexaRankNormalized = 0.6;
else if(alexaRank > 100000 && alexaRank < 1000001)
alexaRankNormalized = 0.4;
else if(alexaRank > 1000000 && alexaRank < 4000001)
alexaRankNormalized = 0.2;
/*else if(alexaRank == -1)
throw new AlexaRankNotRankingException("AlexaRank is not ranking URLs
properly.");*/
else
alexaRankNormalized = 0.0;

//Show the user the regular ranks, normalized ranks, and final rank
System.out.println("AlexaRank: " + alexaRank + " | PageRank: " + pageRank +
"\nAlexaRank_Normalized: " + alexaRankNormalized + " |
PageRank_Normalized: " + pageRankNormalized + " | Final Rank: " +
((alexaRankNormalized + pageRankNormalized) / 2));

//Tell the user the system is updating the records
System.out.print("Updating records...");

//Create a database connection
Connection connection = MySQL.connectToDatabase(databaseUsername,
databasePassword, databaseConnectivity);

//If the url thats being examined is larger than 14 characters long and is a
migre.me url, perform the following
if(urlText.length() > 14 && (urlText.substring(0, 15).equals("http://migre.me")))
{
//Create an update statement that will update all instances of the migre.me url
PreparedStatement updateStatement = connection.prepareStatement("UPDATE
tweetfeatures SET featureValue = ? WHERE featureText LIKE ?;");

//Set the featureValue
updateStatement.setDouble(1, ((alexaRankNormalized + pageRankNormalized) /
2));

//Set the featureText
updateStatement.setString(2, urlText);

//Execute the update
```

```
updateStatement.executeUpdate();

//Close the update statement
updateStatement.close();
}
//If the previous condition failed, perform the following
else
{
//Create a update statement that will update the URL we just got the rank for
PreparedStatement updateStatement = connection.prepareStatement("UPDATE
tweetfeatures SET featureValue = ? WHERE tweetFeatureID = ?;");

//Put the rank value in the first unknown section of the update statement
updateStatement.setDouble(1, ((alexaRankNormalized + pageRankNormalized) /
2));

//Put the tweetFeatureID in the second unknown section of the update statement
//updateStatement.setInt(2,
Integer.parseInt(resultSet.getString("tweetFeatureID")));
updateStatement.setInt(2, Integer.parseInt(featureID));

//Execute the update statement
updateStatement.executeUpdate();

//Close the update statement
updateStatement.close();
}

//Commit the changes and close the connection
connection.commit();
connection.close();

Thread.currentThread().interrupt();
}
catch(MalformedURLException e)
{
System.out.print("The URL is not a proper URL.  Assigning a rank of 0...");

try
{
Assign Zero(Rank.databaseUsername, Rank.databasePassword,
Rank.databaseConnectivity);
}
catch(InstantiationException e1)
{
e1.printStackTrace();
```

```
}
catch(IllegalAccessException e1)
{
e1.printStackTrace();
}
catch(ClassNotFoundException e1)
{
e1.printStackTrace();
}
catch(SQLException e1)
{
e1.printStackTrace();
}
}
catch(SocketException e)
{
System.out.print("The URL throws a socket exception.  Assigning a rank of 0...");

try
{
Assign Zero(Rank.databaseUsername, Rank.databasePassword,
Rank.databaseConnectivity);
}
catch(InstantiationException e1)
{
e1.printStackTrace();
}
catch(IllegalAccessException e1)
{
e1.printStackTrace();
}
catch(ClassNotFoundException e1)
{
e1.printStackTrace();
}
catch(SQLException e1)
{
e1.printStackTrace();
}
}
catch(IOException e)
{
e.printStackTrace();
}
catch(SQLException e)
{
```

```
        e.printStackTrace();
        }
        catch(InstantiationException e)
        {
        e.printStackTrace();
        }
        catch(IllegalAccessException e)
        {
        e.printStackTrace();
        }
        catch(ClassNotFoundException e)
        {
        e.printStackTrace();
        }
        }
        };
}
```

D.2.5 UniformRecourceLocator Class

```
package hirkm;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import hirkm.http.Requests;
import hirkm.sql.MySQL;

public class UniformResourceLocator
{
        public static void extractURL(String databaseUsername, String
        databasePassword, String databaseConnectivity) throws InstantiationException,
        IllegalAccessException, ClassNotFoundException, SQLException
        {
        //Create a database connection
        Connection connection = MySQL.connectToDatabase(databaseUsername,
        databasePassword, databaseConnectivity);

        //Tell the user a database connection was made
        System.out.println("Database connection established for extracting URLs.");
```

```
//Inform the user that the extraction of URLs is beginning
System.out.println("Beginning to extract URLs from tweets.");

//Prepare the query that will get the tweets from the database
String query = "SELECT tweet_id, tweet FROM twitterDataset;";

//Get the query results from the database
ResultSet resultSet = MySQL.getSelectResultsFromDatabase(connection, query);

//Loop through all the results returned from the database
while(resultSet.next())
{
//Perform the following if the tweet is not null
if(resultSet.getString("tweet") != null)
{
//Regular expression for finding URLs
String regex = "((([A-Za-z]{3,9}:(?:\\/\\/)?)(?:[-;:&=\\+\\$,\\w]@)?[A-Za-z0-9.-
]+|(?:www.|[-;:&=\\+\\$,\\w]+@)[A-Za-z0-9.-]+)((?:\\/[\\+~%\\/.\\w-_]*)?\\??(?:[-
\\+=&;%@.\\w_]*)#?(?:[.\\!\\/\\\\w]*))?|(([A-Za-z0-9_])+[.]([A-Za-z0-
9_])+[.]([A-Za-z0-9_])+)|(([A-Za-z0-9])+([.])([A-Za-z0-9])+))";

//Set the regular expression pattern to look for that will find URLs in the tweet
Pattern pattern = Pattern.compile(regex);

//Match the established pattern with the actual tweet
Matcher matcher = pattern.matcher(resultSet.getString("tweet"));

//Perform the following for all matches found in the tweet
while(matcher.find())
{
//Set the tweet_id and actual url to local variables
String tweet_id = resultSet.getString("tweet_id"),
url = matcher.group();

//If the url is less than 7 characters, perform the following
if(url.length() < 7)

//Change the string to "NOT A URL"
url = "NOT A URL";
//If the URL does not start with the string "http://", perform the following

else if(!url.substring(0, 7).equals("http://"))

//Change the string to "NOT A URL"
url = "NOT A URL";
```

```
//If the extracted URL is longer than 255 characters, perform the following
if(url.length() > 255)

//Make the URL 255 charters long
url = url.substring(0, 255);

//If the URL is a real URL and if there are no duplicates of the ':' character, perform the following
if(!url.equals("NOT A URL") /*&& checkURLStatus(url)*/ &&
!checkForDuplicateURLs(url, ':'))
{

//Create a database connection
Connection connection2 = MySQL.connectToDatabase(databaseUsername, databasePassword, databaseConnectivity);

//Create an insert statement that will put the extracted url into the database along with the tweet_id that it came from
PreparedStatement insertStatement = connection2.prepareStatement("INSERT INTO tweetFeatures (tweet_id, featureType, featureText)" + " VALUES (?, 1, ?);");

//Put the tweet_id in the first unknown section of the insert statement
insertStatement.setString(1, tweet_id);

//Put the URL in the second unknown section of the insert statement
insertStatement.setString(2, url);

//Execute the insert statement
insertStatement.executeUpdate();

//Close the insert statement
insertStatement.close();

//Commit the changes and close the connection
connection2.commit();
connection2.close();
}
}
}
}

//Close the resultset from the database
resultSet.close();
//Commit and close the database connection
connection.commit();
```

```
connection.close();

//Tell the user that the method has completed executing
System.out.println("Finished extracting URLs from tweets.");
}

private static boolean checkForDuplicateURLs(String stringToCheck, char
characterToFind)
{

//Create local variable
int duplicateCount = 0;

//Loop through the string and perform the following
for(int i = 0 ; i < stringToCheck.length(); i++)
{

//If the string has a matching character, perform the following
if(stringToCheck.charAt(i) == characterToFind)
//Increase the duplicateCount variable by 1
duplicateCount++;
}

//If there is more than one of a given character, perform the following
if(duplicateCount > 1)

//Return true
return true;

//If there is one or less of a given character, perform the following
else

//Return false
return false;
}

private static boolean checkURLStatus(String url) throws IOException
{

//Create local variable
boolean isTrueURL = false;

//If the URL that is checked returns HTTP code 200, perform the following
if(Requests.getResponseCode(url) == 200)
//Set the isTrueURL variable to true
isTrueURL = true;
```

```
else

//Set the isTrueURL variable to false
isTrueURL = false;

//Return the isTrueURL variable
return isTrueURL;
}

public static String getFullURL(String url) throws IOException
{

//Request the real url of any url shortener
String line =
Requests.getRequest("http://www.checkshorturl.com/expand.php?u=" + url);

//Split the code to find the URL
String[] lineArray = line.split("<td style=\"border-bottom: 1px dotted
black;width:750px;height:20px;padding:10px;\"><a href=\"");

//If the number of tokens generated from the split is more than 1, perform the
following
if(lineArray.length > 1)
{

//Split the code to find the URL
lineArray = lineArray[1].split("\" target=\"_blank\" rel=\"nofollow\">");
lineArray = lineArray[0].split("\" title=\"");

//Return the URL
return lineArray[0];
}

//If the number of tokens generated from the split is not more than 1, perform the
following
else
{

//Return the original URL
return url;
}
}

public void verifyURLs(String databaseUsername, String databasePassword,
String databaseConnectivity) throws InstantiationException,
IllegalAccessException, ClassNotFoundException, SQLException, IOException
```

```
{

//Create a database connection
Connection connection = MySQL.connectToDatabase(databaseUsername,
databasePassword, databaseConnectivity);

//Tell the user a database connection was made
System.out.println("Database connection estabishoshed...");

//Inform the user that the URL verification has begin
System.out.println("Beginning URL verification");

//Prepare the query that will get the URLS
String query = "SELECT * FROM tweetfeatures where featuretype = 1 AND
featureValue IS NULL and tweetFeatureID < 380413;";

//Get the query results from the database
ResultSet resultSet = MySQL.getSelectResultsFromDatabase(connection, query);

//Loop through all the results returned from the database
while(resultSet.next())
{
String text = resultSet.getString("featureText");

System.out.println(text);

//If the URL does not work, perform the following
//if(!checkURLStatus(resultSet.getString("featureText")))
if(!checkURLStatus(text))
{
//Create a database connection
Connection connection2 = MySQL.connectToDatabase(databaseUsername,
databasePassword, databaseConnectivity);

//Create a delete statement that will delete the URL that do not work
PreparedStatement deleteStatement = connection2.prepareStatement("DELETE
FROM tweetFeatures WHERE tweetFeatureID = ?");

//Put the tweetFeatureID in the first unknown section of the delete statement
deleteStatement.setString(1, resultSet.getString("tweetFeatureID"));

//Execute the delete statement
deleteStatement.executeUpdate();

//Close the insert statement
deleteStatement.close();
```

```
            //Commit the changes and close the connection
            connection2.commit();
            connection2.close();
            }
            else
            {

            //Create a database connection
            Connection connection2 = MySQL.connectToDatabase(databaseUsername,
            databasePassword, databaseConnectivity);

            //Create a update statement that will update the URL feature row to indicate it has
            been verified
            PreparedStatement updateStatement = connection2.prepareStatement("UPDATE
            tweetfeatures SET featureValue = \"99\" WHERE featureText LIKE ?");

            //Put the featureText in the first ? in the updateStatement
            updateStatement.setString(1, text);

            //Execute the update statement
            updateStatement.executeUpdate();

            //Close the update statement
            updateStatement.close();

            //Commit the changes and close the connection
            connection2.commit();
            connection2.close();
            }
            }

            //Close the ResultSet
            resultSet.close();

            //Commit and close the database connection
            connection.commit();
            connection.close();

            //Inform the username the database connection is closed and the thread finshed
            System.out.println("Database connection closed");
            System.out.println("verifyURLs thread ended");
            }
}
```