A DATA COLLECTION SYSTEM FOR RUMOR DETECTION

by

Ye Wang

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

Summer 2017

A DATA COLLECTION SYSTEM

FOR RUMOR DETECTION

by

Ye Wang

Approved: _____

Hui Fang, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Kenneth E. Barner, Ph.D.
Chair of the Department of Electrical and Computer Engineering

Approved: _____

Babatunde A. Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____

Ann L. Ardis, Ph.D.
Senior Vice Provost for Graduate and Professional Education

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Prof. Fang for the constant support of my study and related research, for her patience, motivation, and immense knowledge. She not only guided me how to do research, but also taught me analytical and critical thinking skills.

I would also like to thank my colleagues Peilin Yang, Yue Wang, Kuang Lu and Hao Xu who have helped me during my research.

Last, but not least, I would like to thank my family, for the continuous support and encouragement throughout my time in graduate school.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Nowadays, a lot of unsubstantiated and unverified information, named rumors, are created and propagated through the Internet because of the easiness of posting information online and lack of supervision. These rumors may cause users' confusion and social unrest. To prevent the negative influences, rumor detection which employs machine learning has been well studied. And almost all of these machine learning based methods rely on a large rumor dataset, which makes a large collection of rumor related data highly desired. However, current rumor collection methods are partially manual and usually specific for a single platform.

In this thesis, we propose a rumor collection system to automatically collect rumor related data from both search engine and social media. It mainly consists of two parts. First, instead of using user input as the search query, a query generator is proposed to avoid directly using user input as the search query, which may result in the fail of search. It can generate a set of queries based on the user's input. After that, a novel rumor crawler is built to collect rumor related data by using the generated queries.

To validate our rumor collection system, experiments are taken on the Tweets from January 2016 to March 2017. The result of 50 different rumors shows that, compared with current widely used Twitter Search API, our system can crawl more rumor with an average increasement of 3.589 times. Furthermore, for some rumors, our system is still effective when Twitter Search API returns no results.

Chapter 1

INTRODUCTION

In the last decade, the Internet has become a major source for news. A study conducted by the Pew Research Center[1] in 2016 has identified the Internet as the most important source for the news for people under the age of 30 in the US and the second most important source overall after television. However, there are also amounts of rumors (unsubstantiated and unverified information) propagated on the Internet. An example of such phenomenon is the 2010 Chile earthquake where the propagation of related rumors on the Internet caused chaos and social unrest among the news consumers. To clean the environment of Internet and reduce the harm of rumors, automatic rumor detection methods which take the advantage of big data and machine learning are very helpful. Rumor collection is a pre-requirement for such work. However, current rumor collection methods are partially manual and have limitations which make them unable to collect a complete data collection. In this thesis, we propose and implement a framework to automatically collect rumor related data on the Internet.

In Section 1.1, we briefly discuss the motivation of our work and the strengths and shortcomings of existing methods. Section 1.2 introduced our rumor collection framework and the contributions of the thesis. Finally, Section 1.3 provides an outline of this thesis.

[1] http://www.journalism.org/2016/07/07/pathways-to-news/

1.1 Motivation

Rumor, especially malicious rumor, can cause a dramatically negative influence on the Internet. Detecting and verifying rumors are highly needed to rebuild the environment of Internet without chaos and confusion. Generally, with the help of big data and machine learning, the detection process can be done automatically. However, the performance of algorithms in machine learning highly rely on a large, comprehensive, and accurate training dataset. Thus, collecting of rumor related data is always the first fundamental step for related research.

On the Internet, there are many different types of news sources, such as social media, news websites, general websites and personal blogs. Within these sources, the most influential ones are social media and news websites considering the amount of time people spend on these websites. Meanwhile, the lack of supervision on these online platforms makes rumors spread easier and faster. In this regard, we pick news websites and social media to collect rumor related data. For news websites, since the framework and organization of various news websites could be totally different, the current methods are partially manual and usually for specific websites, topics or domains, which make the process painful, time-consuming and unscalable. For social media, based on the data from eBizMBA [2], Twitter is one of the most popular social media, after Facebook and Youtube. The Representational State Transfer (REST) Application program interfaces (APIs) of Twitter are well developed and widely used which enable others to do the collecting of data easily. Twitter Search API is widely used for Twitter collection task, but there are two limitations of the Twitter Search API. For one thing, only around 3000 tweets can be returned in maximum for a specific request. For another, only tweets within roughly 7 days can be accessed. These limitations greatly hurt the completeness of the data collection which is very important for many pieces of research. As a result, a general framework that can automate the entire process and collect the data more completely is in pressing need. It will save a

[2] http://www.ebizmba.com/articles/social-networking-websites

great amount of time for researchers and enable them to focus on the research part. Also, it will greatly lower the threshold of doing related research.

1.2 Rumor Collection Framework

In this thesis, we aim to construct a framework to collect rumor related data on the Internet automatically. Figure 1.1 shows the general pipeline of our system. As can be seen, the system has three major parts, Query Generator, Rumor Crawler and Online Judgement.

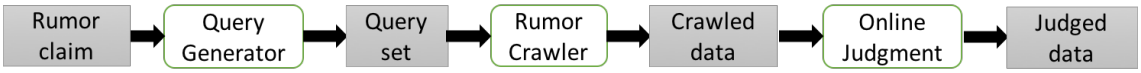

Figure 1.1: The pipeline of the rumor collection system. The inputs and outputs are shown in gray and the subsystems in green.

The input of Query Generator is the rumors that are collected from Snops.com. We find that directly using a rumor claim as the search query will result in few results, typically for Twitter since a tweet is limited to 140 characters. The search query is the input of a user put into a web search engine to satisfy the information needs. As a result, a Query Generator is proposed. It is typically optimized for cases that the original queries are relatively long and documents are short. By using linguistic and statistic features, the system can extract keywords from the rumor claim and construct a set of short queries based on it. As a result, the recall of the rumor data collecting process increased dramatically.

After the query sets of rumors have been generated, multiple crawlers for different rumor sources are built. We pick Google, Google News and Twitter as the sources of data. Google is the largest search engine where users can find rumor related web pages by search. Google News provides and aggregates up-to-date news from sources all over the world. Twitter is one of the largest social media platforms and provides powerful and versatile APIs to access the public data while many other mainstream platforms do not. We extended an existing Google crawler framework to scrape data

from both Google and Google News. For Twitter, to overcome the time and amount limitation of Twitter API, we introduced a novel Twitter crawler by using Twitter advanced search and Selenium. Twitter advanced search is a web-based Twitter search engine, which enables users to specify the date of searched tweets. It perfectly overcomes the limitation of search API. After this, we extract data from the browser with Selenium, a browser automation toolkit.

In addition, to evaluate the performance of the framework, based on an existing online evaluation system, an online judgment system is constructed and deployed to enable users to judge unlabeled rumor related data without bounding to locality. The input of this system is the data collected by the Rumor Crawler. Since labeling data is laborious, time-consuming, a ranking function is used to evaluate the relevance of a specific result and the rumor. If too much data are crawled for a specific rumor, the least relevant data will be removed automatically to increase the efficiency.

The primary goal of our research is to develop a new automatic rumor collection framework. Our research contributions are as follows.

1. A combined automatic system for rumor collection is proposed, shown to be effective in the experiment.

2. A query generation algorithm that shortens the size of a query based on word distance is proposed to increase the recall of rumor collection.

3. To collect historical tweets, a novel tweets collection method based on the simulation of user's browsing behavior is proposed.

1.3 Thesis Overview

In Chapter 2, we review current approaches for query extraction and reduction, rumor detection, and rumor collection. Chapter 3 describes the query generation subsystem. Chapter 4 introduces the rumor crawling subsystem. Chapter 5 describes the experiment as well as discuss the results obtained by the new rumor collection system. In Chapter 6, we summarize the thesis and outline directions for future work.

Chapter 2

RELATED WORK

2.1 Automatic Extraction of Keywords

Automatic extraction of keywords have exhibited their potential to improve the efficiency and accuracy of searching in natural language processing (NLP) and information retrieval (IR) tasks[11] Terse queries that contain only a small selection of keywords from a more verbose description of the actual information is more efficient during a search [2, 3].

Corpus-oriented statistics of individual words is a foundational method and they believe that keywords definitely occur much more times than other single words. For example, Andrade et al. extract biological information directly from scientific literature by comparing word frequency distribution and their relative accumulation [1]. However, operating only on single words dramatically limit the accuracy of keywords extraction. Co-occurrence distribution is used to show the importance of a term in the document [24]. Also, since tweets are short and most words only occur once which makes the word frequency based method impractical.

To avoid these drawbacks, features-based linguistics approaches have been proposed. In this method, features can be anything to get more linguistic knowledge, such as Named entities, Part-of-speech taggers [27], length of words, the position of words [45], previous or next word [44], Term Frequency $*$ Inverse Document Frequency of the word (TF $*$ IDF) [44, 16]. For example, in [9], a lexical chain that holds a set of semantically related words of a text is applied to extract the keywords.

Intuitively, linguistics approaches heavily rely on the feature itself and generally, one feature is not scalable enough to dramatically increase the accuracy for almost all

database. Therefore, nowadays, supervised machine learning has been widely applied in keywords extraction.

Using supervised machine learning for an automatic extraction of keywords was first proposed by Turney [36] and later expanded by some other researchers [44, 13, 16].

Query expansion (QE) is the process of reformulating a seed query to improve retrieval performance in information retrieval operations. Leverage Pseudo-Relevance Feedback (PRF) for Query Expansion (QE) is widely used in research of microblog search [12, 22, 8, 43, 38, 21], which assumes that most of the frequent terms in the pseudo-relevance documents are useful. Many researcher aims at using semantics to enhance microblog search [40, 47, 15].

However, machine learning based approaches are highly time-consuming and need a large dataset. As far as we know, there is no appropriate up-to-date social event dataset available now. In this paper, linguistic methods work well for our data collection task and we may move to machine learning methods in the future.

2.2 Rumor Detection

The study of rumors is a classic topic in social sciences. Case studies on specific rumors are conducted to reveal the features of their context and the common characteristics between rumors.

The problem of rumor detection can be cast as binary classification tasks [5]. The extraction and selection of discriminative features significantly affect the performance of the classifier. We studied many features used for rumor detection. Mei et al. addressed the problem of rumor detection in microblogs and explore the effectiveness of 3 categories of features: content-based, network-based, and microblog-specific memes for correctly identifying rumors [28]. Also, some semantic features i.e. opition words [33], Speech Act Verbs [35, 14, 45, 46], N-grams, and syntactic features i.e. punctuations [37], abbreviations [7], dependency sub-trees [37, 23, 26], Twitter specific characters [37, 45]are used for classification.

Some models reveal how rumors have spread after an event. Some studies focus on the behavior of Twitter users under an emergency situation [25, 31, 34]. Wu et al. utilized a message propagation tree where each node represents a text message to classify whether the root of the tree is a rumor or not [39].

2.3 Rumor Collection

Most of Twitter related researches use Twitter API to collect tweets. Some researchers maintain a large-scale dataset by keeping storing data retrieved with Streaming API which provides around 1% sampling of total tweets and then specific regular expression is used to filter out tweets related to a specific rumor [28]. Hashtags are also widely used to identify topics [37, 29]. Some studies collected data using Twitter's Search API, which is free to use and provides tweets published in the past 7 days [4, 17, 20]. The synthetic data set is another possible way to generate a large data set(generated using Watts and Strogatz model [3, 26]).

These limitations constrained the research in this field and inspired us to propose a new rumor collection system.

2.4 Focused Crawling

Focused crawling was first introduced by Soumen et al[6]. to crawl topic-specific web pages. Considering the large amount of information on the Internet, in order to save hardware and network resources, a focused web crawler analyzes the crawled pages to select only related documents and ignores the rest. It mainly has two components, a classifier to evaluate the relevance of hypertext documents regarding the focused topic, and a distiller to identify nodes as access points to pages is also. Ahmed [32] improved the distiller by using an Optimized Nave Bayes (ONB) classifier

An optimized focused web crawler that learns from the information collected by the knowledge base within one domain or category is proposed by Rungsawang and Angkawattanawit[30]. Liu, Milios and Korba[18] presented a framework for focused web crawler based on Maximum Entropy Markov Models (MEMMs)

In our system, we focus on a variety of rumors instead of a specific domain or topic, which makes focused crawling not suitable. Also, focused crawling is not applicable on social media. As a result, instead of using focused crawling, we crawl rumor related data by doing search on both search engine and social media. To improve the efficiency of crawling, a query generator is proposed to optimize the search queries.

Chapter 3

QUERY GENERATOR

The general pipeline of the rumor collection system is shown in Figure 1.1, as can be seen in that figure, the system is composed of three main subsystems for query generation, rumor collection, and online judgment. This chapter will describe in detail the query generation subsystem.

Generally, the input of rumor crawling subsystem is the raw rumor claim. The rumor claim is usually long and may contain many meanless words. As one can well imagine, if it is directly used to search on Google or Twitter, the result will not be good, especially for Twitter considering that a tweet can only have 140 characters in maximum. To improve the performance and increase the recall of the crawler, a Query Generator is necessary to generate terse queries based on user input.

The Query Generator takes a rumor claim provided by the user as the input. Then by using linguistic and statistic techniques i.e. Named entity recognition, Part-of-speech tagging, Stop words to extract keywords from the rumor claim and generate a set of queries automatically. The first part of this chapter describes the necessaries of building a Query Generator subsystem and the second part describes the approaches explored and how to integrate multiple methods to solve the problem.

3.1 Why Necessary?

Before introducing our Query Generator in detail, let's review some characteristics of collecting rumors on the Internet. First of all, a query of the rumor is a pre-requirement to collect rumor related information online. In our system, Twitter and Google are chosen as the sources of rumor related data. To collect the data, Twitter Crawler and Google (News) Crawler are built respectively. For the Twitter crawling

task, there are mainly two choices to implement it. One option is based on the Twitter search API, which can return tweets based on the request query. Another one is by using Twitter Advanced Search, which is a web-based search engine. Both methods need a query as the input to get related tweets. As can be seen, how to set the search query will make a great difference to the amount and the precision of returned tweets. For example, "Cancer Is Caused by a Deficiency of Vitamin B17" is a wildly spread rumor in recent years. If we search this rumor claim directly on Twitter, only 7 tweets are returned. Instead, if we use "Cancer B17", which is the keywords in the rumor claim, as the search query, there are thousands of tweets returned. For widely used general search engines, like Google, users typically interact with the system through natural language queries to address a broad range of information needs. If we search the same rumor claim mentioned above on Google, even though, compared with Twitter, more results will be returned since the huge amount of information available on the Internet and also possible optimization is done by Google. However, when we take a deep look at the result, we find that the results are extremely biased and most of them support this rumor even though it is a rumor which has been verified as false.

In summary, there are two main drawbacks of using the raw rumor claim as a query. First, the rumor claims are usually long, with an average length of over eight words in our case, since a specific rumor is usually detailed and unable to be described in several words. Using the claim to search will greatly decrease the recall of our Twitter crawler. Second, for both Google and Twitter, since the claim is a description of the rumor, when it is used as a search query, the results from the search engine are more likely to support the claim. As a result, the results presented to the users are biased which may mislead users' judgment on the veracity of the information.

Based on these observations, instead of using rumor claims as the input of Rumor crawler, we decide to generate a set of queries based on the rumor claims that contain fewer words, which is the functionality of our query generator subsystem.

3.2 Candidate Rumor List

To start the generation of queries, we need to get a list of rumor claims as the input. In the real system, the input should come from real users. Considering that the system has not been finished yet at this moment, we need to get the rumor list in another way. A straight forward way to get a list of rumors is collecting rumors on the Internet and record them manually. However, it is time-consuming and also non-scalable.

Instead, we picked Snopes.com as our source to get canditates rumors. It includes a section called "Fact checking" that covers rumors from different fields e.g. Politics, Medical, Fake news. Detailed analysis of the rumors is provided as well, which can help us to understand them. The crawling process is done automatically by building a web crawler, which is more efficient than manual work.

In order to automate the crawling process, a web crawler is developed by using Python and BeautifulSoup4[1]. The result is stored in a CSV file. Crawled data include the rumor claim, publish date, rating, and origin(explanation), as shown in Table 3.1.

3.3 Keywords Extraction

In this section, a variety of keywords extraction methods based on the knowledge of Natural Language Processing and Information Retrieval, are introduced. Keywords, a subset of words or phrases from a text that can describe the meaning of the text, will be extracted from the original query based on different features.

3.3.1 Named Entity Recognition

Named Entity Recognition (NER) is a technique used to label words and phrases in a text which are the names of things, such as person names, company names and location names. In many cases, a rumor claim may be related to a specific person, location or company. These name entities can be used to identify a specific topic or rumor, which makes them irreducible during query reduction.

[1] https://pypi.python.org/pypi/beautifulsoup4

Rumor Claim	Cancer is caused by a deficiency of vitamin "B17", a condition that can be remedied with nutritional supplements.
Rating	False
Origin(Explain)	Discussion of the cancer-fighting properties of a chemical (various referred to as amygdalin, laetrile, or "vitamin B17") has been a fixture of the so-called alternative medicine movement for decades. ..
Published	Jan 13th, 2017
Sources	United States, Food and Drug Administration. "Laetrile, the Commissioner's decision" 1978. PDQ Integrative, Alternative, and Complementary Therapies Editorial Board. "Laetrile/Amygdalin (PDQ®)". 21 December 2016. Lerner, Irving. "Laetrile: A Lesson in Cancer Quackery" CA Cancer J Clin. 1981. Ellison NM, et al. "Special report on Laetrile: the NCI Laetrile Review. Results of the National Cancer Institute's retrospective Laetrile analysis." N Engl J Med. 7 September 1978. Moertel CG, et al. "A clinical trial of amygdalin (Laetrile) in the treatment of human cancer." N Engl J Med. 28 January 1982. Milazzo, S and Horneber, M. "Laetrile treatment for cancer". The Cochrane Database of Systematic Reviews. 28 April 2015 Mercola.com. "New Film "Second Opinion" Exposes the Truth About a 40-Year Long Cover-Up of Laetrile Cancer Treatment" 18 October 2014

Table 3.1: An example of crawled data from Snopes.com

Figure 3.1 is an example output of NER. In this case, by using NER, "WikiLeaks" and "ISIS" are tagged as ORGANIZATION while "Hillary Clinton" is tagged as PERSON. These words identified by NER is critical in this case. If we use "Hillary Clinton ISIS WikiLeaks" as a query to search on Google or Google News, a lot of related information will be presented on the first page. Clearly, the usage of NER can be useful in some cases.

However, NER does not always work since that NER only focus on named entities like person, location and organization. If the rumor is not related to the either of these name entities, no entities will be returned. For example, "Bees have been classified as an endangered species". There are no name entities in this text. As a result, NER is possibly invalid.

Nowadays, NER systems have been created by using linguistic techniques as well as statistical models, i.e. machine learning. However, statistical NER systems typically require a large amount of training data which need to be judged manually.

This limitation makes many researchers hard to train a NER model by themselves. Fortunately, there are models already available such as GATE, Stanford NLP, and OpenNLP. By comparing, Stanford NER is chosen in our system.

Stanford NER[2] is a Java implementation of a Named Entity Recognizer powered by Standford University. Figure 3.1 is an example output by using Stanford NER. Based on the output, we can construct a query "WikiLeaks Hillary Clinton ISIS". In

<Input> : Emails released by WikiLeaks confirm Hillary Clinton sold weapons to ISIS
<Output> : ORGANIZATION: WikiLeaks
PERSON: Hillary Clinton
ORGANIZATION: ISIS

Figure 3.1: An example output of Stanford NER

addition, considering that a person could be mentioned and referred with only the first name or last name, we can optimize the method by only take the first name or last name as part of the reconstructed query. So, the query turns to be "WikiLeaks Hillary ISIS" or "WikiLeaks Clinton ISIS".

3.3.2 Part-of-Speech Tagging

In the English language, words can be considered as the smallest units of meaning. Based on the use and functions of words, they can be categorized into several types or parts of speech. There are 8 major parts of speech in English grammar: noun, pronoun, verb, adverb, adjective, conjunction, preposition, and interjection. Within these parts, noun and verb are those most likely contains keywords. A noun is a word that names a specific object such as person, thing, animal or place. Though some of the Noun phrases can be detected and tagged by using the NER technique mentioned above, the range of Noun phrases is larger than NER. As a result, Noun phrases are

[2] https://nlp.stanford.edu/software/CRF-NER.shtml

sometimes necessary as well, especially when NER tagger returns nothing. Meanwhile, a verb is a word used to describe an action, such as run, listen, swim. Considering that a specific rumor is about "someone did something", the action "did something" is important to identify this rumor instead of just uses the Noun phrases involved. Based on the analysis of the formation of a rumor, noun and verb are chosen to be extracted considering the relatively high probability to contain keywords.

To extract the Noun and Verb phrases from the text, a Part-Of-Speech Tagger (POS Tagger) is introduced, which is a piece of software that reads the text in some language and assigns parts of speech to each word (and another token), such as noun, verb, adjective, etc. In our system, We extract noun and verb phrases from the text by using Stanford POS-Tagger [3], as shown in Figure 3.2.

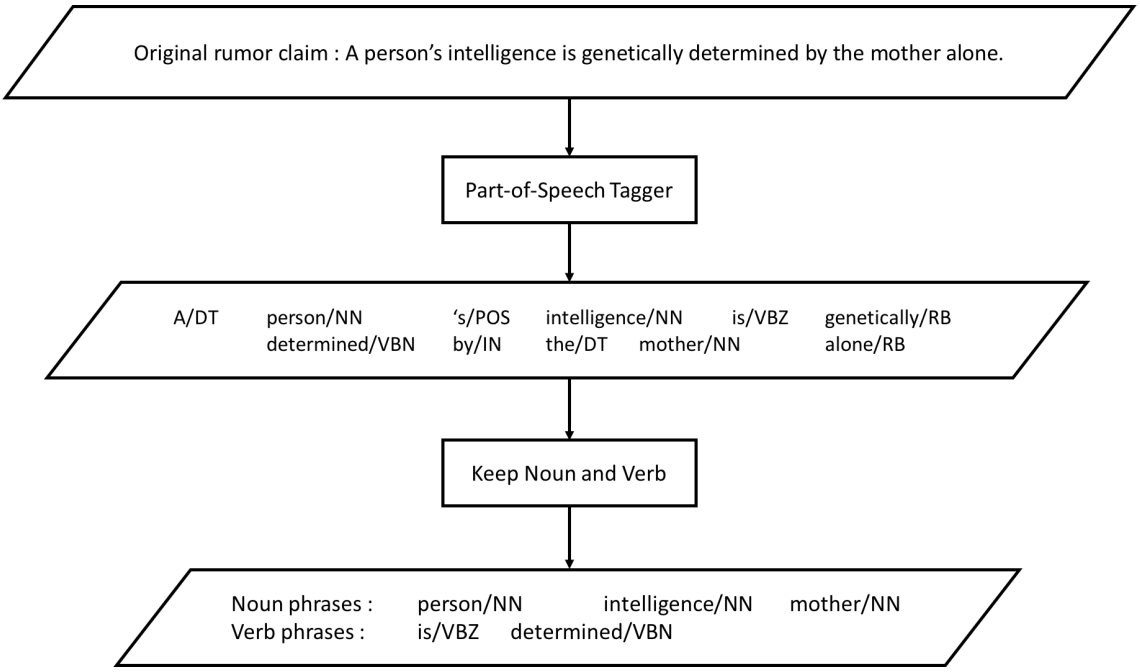

Figure 3.2: An example output of Part-of-Speech Tagger

Based on the noun phrases we get, a possible query could be "person intelligence

[3] https://nlp.stanford.edu/software/tagger.shtml

mother". By using verb phrases, "person intelligence is determined mother" could be another query. It is worthwhile to note that "determined" is the keyword in this rumor but "is" is too general to express the meaning of the rumor. As a result, stop words are imported in next part to solve this problem and remove meanless words from queries.

3.3.3 Stop Words Removal

Stop words usually refer to the words in a document that are some frequently used words. These words would appear to be of little value in expressing the meaning of the sentence and can be excluded from the vocabulary entirely during keywords extraction.

Removing stop words is a widely used step within natural language processing and information retrieval. Considering that the rumor claim inputted by a user is a verbal query, stop words do exist in the rumor claims, such as "a", "by","of", as shown in Figure 3.1. To remove stop words from a query, using a list of stop words is the common way to solve it since the stop words are relatively static even though it may vary for different domains. The general way to get a stop word list is to sort the terms by the frequency, and then take the most frequent terms. Some published stop word lists are also available i.e. Snowball stop word list[4], Terrier stop word list, Minimal stop word list. In our system, Snowball stop word list is used. Also, stop words could vary in different domains. So a possible optimization is constructing stop word list for different domains separately instead of using a general one.

3.3.4 Inverse Document Frequency

The term weighting function known as IDF has been extremely widely used, usually as part of a TF*IDF function in information retrieval to measure the importance of a word to a document in a collection or corpus. Since that tweets are short, the term frequency of words in a tweet is usually one. Instead of using the combination of TF and IDF, IDF is used alone to weight terms. It is based on the heuristic that

[4] http://snowball.tartarus.org/algorithms/english/stop.txt

a word is not a good discriminator if it occurs in many tweets or Google results. IDF scales down the term weights of terms with high collection frequency. It is defined as the total number of occurrences of a term in the collection divides the total number of documents in a collection, and then taking the logarithm of that quotient, as shown in the equation below. Obviously, the IDF score of a rare term is high, whereas the IDF score of a common term is more likely to be low. As a result, in order to remove common words from the original query, the terms with low IDF could be the ones to remove.

$$idf_t = log\frac{N}{df_t}$$

In order to calculate the IDF of tweets, 1,000,000 tweets are crawled by using one of the Twitter Steaming APIs, which provides low latency access to Twitters global stream of tweet data. There are three types of streams, as shown in Table 3.2. Considering that we need sampling tweets of the whole public twitter data, Public streams are suitable. To get the data, One Streaming API method, GET statuses/sample, is used. By using this API, A small random sample of all public statuses will be returned. The tweets returned by the default access level are the same, so if two different clients connect to this endpoint, they will see the same tweets.

Type	Description
Public streams	Streams of the public data flowing through Twitter. Suitable for following specific users or topics, and data mining.
User streams	Single-user streams, containing roughly all of the data corresponding with a single user's view of Twitter.
Site streams	The multi-user version of user streams. Site streams are intended for servers which must connect to Twitter on behalf of many users. Site Streams is a closed beta. Applications are no longer being accepted.

Table 3.2: Steaming API types

To use Streaming API in Python code, a Python library for accessing the Twitter APIs called Tweepy[5] is used. There are mainly three steps to use it.

[5] http://www.tweepy.org/

1. Create a class inheriting from StreamListener

```
import tweepy
# override tweepy.StreamListener
# to add logic to on_status
class MyStreamListener(tweepy.StreamListener):
    def on_status(self, status):
        print(status.text)
```

2. Using that class create a Stream object

```
myStreamListener = MyStreamListener()
myStream = tweepy.Stream(auth = api.auth,
                         listener=myStreamListener())
```

3. Connect to the Twitter API using the Stream.

```
myStream.sample(languages=['en'])
```

Once we get the IDF weighting, we try to set a score as a threshold at first. Words with a score lower than the threshold will be considered as non-keywords. However, this method does not work well because the threshold for different queries varies a lot, also some common words identified by IDF score could also contain key concepts of the query. It is hard to set a general threshold that is suitable for each case. Instead, a relatively loose threshold can be used to filter out the most common words, which is similar to the functionality of stop words.

3.3.5 Stemming

Stemmer is an automated tool which produces a base string (the word stem) and removes morphological affixes from words. Algorithmic stemmers and dictionary-based stemmers are the two types of stemmers. Despite the promise of out-performance by dictionary-based stemmers in normal cases, algorithmic stemmers still greatly used in information retrieval since having the advantage that they are available out of the box, fast, and memory-saving.

In our system, algorithmic stemmer [6] is used instead of dictionary-based stemmers because Twitter is social media and tweets contain many informal words, misspelled words, Internet slang words that usually are not covered by dictionary-based stemmers. Stemmer is used to combine words with the same word stem together to optimize the IDF based term-weighting method.

3.4 Shorten the Length of Query

By using the query generation methods we introduced, we can get a set of queries for a specific rumor based on various features i.e. NER, POS-Tagging, Stop words. The number of keywords returned from these features varies from rumor to rumor, sometimes a few and sometimes a lot. The number of terms extracted by different features, like noun phrases, verb phrases, name entities, stop words in the rumor claims varies a lot. It is possible that a query contains too many keywords so that most rumor related tweets only mentioned some of these words instead of all of them, which means that the generated query should use only a subset of candidtated keywords instead of all of them.

Search engines usually perform better with short queries than long queries. We note that the performance of a search engine is highly related to the size of search queries. If the query length is long, the returned results will be more precise but become fewer. If the query length is short, then returned results will contain more irrelevant tweets but the amount of results increase dramatically. Based on this notation, an experiment is done to figure out the relation between the size of a query and the number of returned results.

Based on the principle of the search engine, it is promised that a short search query will return more results than a longer query that includes the short query. However, what is the suitable size for a query? We need to trade off between the amount of returned results and the relevance of the returned results and the rumor. As can be seen in Figure 3.3, the average amount of results varies a lot with different query

[6] http://snowballstem.org/algorithms/

length. The number of results with a query size of 2 words is 4 times of 3 words. The amount of results with a query size of 3 words is 6 times of 4 words. When the size of a query is larger or equal to 6, nothing is returned on average. To balance the relevance and quantity, we decide to pick 3 as the size of a query.

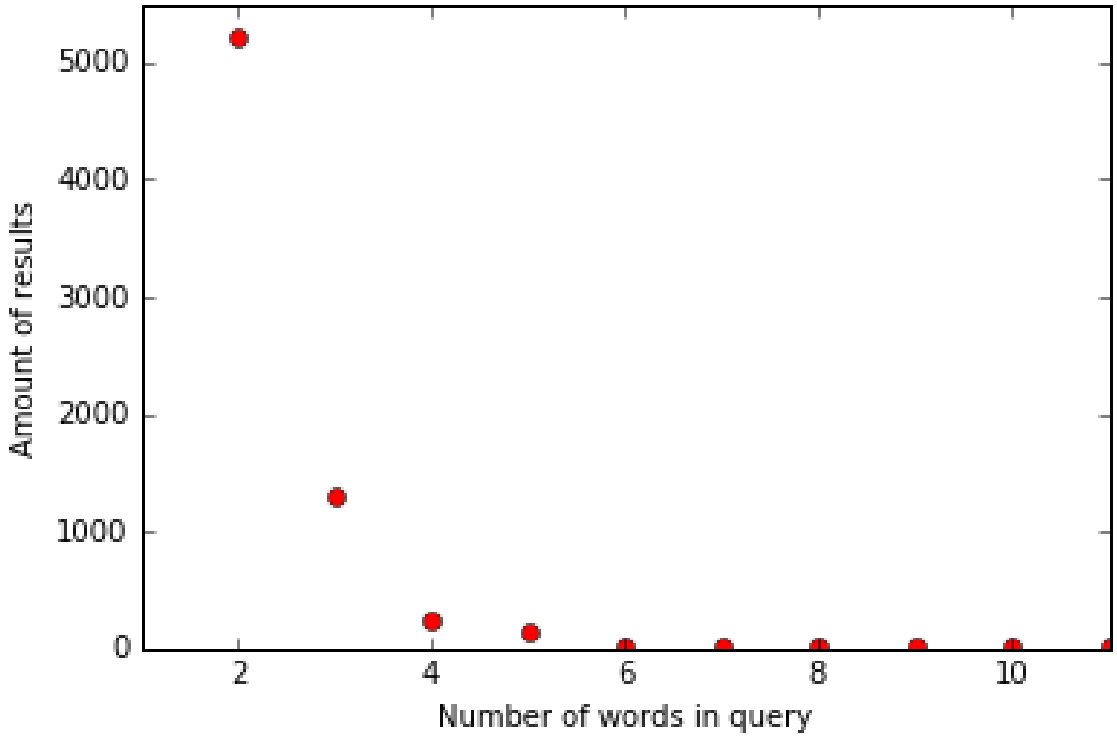

Figure 3.3: Relation between query length and returned tweets

To shorten the size of query further, we tried some methods based on information retrieval and NLP knowledge. As we mentioned before, IDF score is calculated by using 100,000 sampling tweets. Since we already get the weight of all words in the query. We can use the IDF score as a filter to pick 3 keywords from the original query. First, we sort the words in the query based on the IDF score. Then top 3 words among these keywords are picked to construct a new query. This method does not work well since the IDF score is not sufficient to distinguish the most valuable keywords from the other keyword candidates. For example, there are a lot of topics related to "Donald Trump",

which makes "Trump" a popular keyword with a relatively low IDF score. Clearly, the person who is involved in a rumor should be a keyword in the generated query. If we pick top 3 keywords based on the IDF score to construct the query, the important keyword "Trump" will not be included in the generated query. As a result, we move to a new method based on the word distance to shorten the length of a query.

Word Distance

We assume that, in English, the words appear at the beginning of a sentence are more important. Also, keywords are likely to be close to each other. Based on this observation, we proposed a new method to shorten the length of query. First, we extract keywords from the rumor claim by using features such as noun phrases and verb phrases. Then we measure the likelihood of each combination of three words based on the relative position of selected words.

For a set of words (w_1, i_1), (w_2, i_2), (w_3, i_3), w_t stand for t_{th} word in original query and i_t stand for the index of w_t in the rumor claim, $i_1 < i_2 < i_3$. To simplify the computation, i_1 is considered as the benchmark, so the distance between three words is represented as: $i_2 - i_1 + i_3 - i_1$.

A brute-force approach to solve the problem needs to consider all $\binom{N}{3}$ combinations where N is the size of original query in words and then pick the combination that has the smallest word distance as the final query. However, suppose (i_x, i_y, i_z) are not adjacent in the original query and word candidates are ordered by the relative position of keywords in the rumor claim, then there must be another subset (i_x, i_y, i_{z-1}) that has a smaller distance. So we only need to enumerate all adjacent words instead, in this case, only *n-2* possible combinations in total. Algorithm 1 shows the word distance based query reduction method.

3.5 Expand Queries with Google Related Searches

When search for a query on Google, related queries that are searched by other users will be shown in the"Related searches" section at the bottom of the result page.

input : A list of potential keywords extracted from the rumor claim
output: A new constructed query

$min\_distance \leftarrow infinite$;
$new\_query \leftarrow null$;
for *All 3 words permutation* $[(w_1, i_1), (w_2, i_2), (w_3, i_3)]$ *of the candidate words list* **do**
    // The permutations are iterated by the order of the position of words in the original query
    $cur\_distance \leftarrow i_2 + i_3 - i_1 * 2$;
    if $cur\_distance < min\_distance$ **then**
        $min\_distance \leftarrow cur\_distance$ // update min_distance
        $new\_query \leftarrow$ "w_1 w_2 w_3" // update new_query
    else
        // If one permutation has the same word distance as min_distance, it will still be passed since the position of the words lean to back
    end
end

Algorithm 1: Keyword extraction by word distance

Top related searches are terms that are most frequently searched with the term entered in the same period of time. Since Google related searches do not filter controversial topics, the bias of returned results about the rumor claim can be reduced by expanding our query set with Google related searches. Figure 3.4 shows an example of related searches of query "boil water twice" which is a query based on the rumor that "Boiling the same water twice will make your water dangerous to drink". We can see the word "reboil" appears in the related searches as a synonym of "boil twice". Also, top two results are highly related to this rumor. Based on this, we decided to use Google related searches as a method to expand the query.

3.6 Conclusion

As shown in Table 3.3, by combining the features in different ways, we generate multiple keyword extraction methods. First, the generated query set greatly increases the recall of the crawlers. Second, comparing with enumerating all possible subsets of the original query, our query generator avoid the crawling of too much irrelevant and

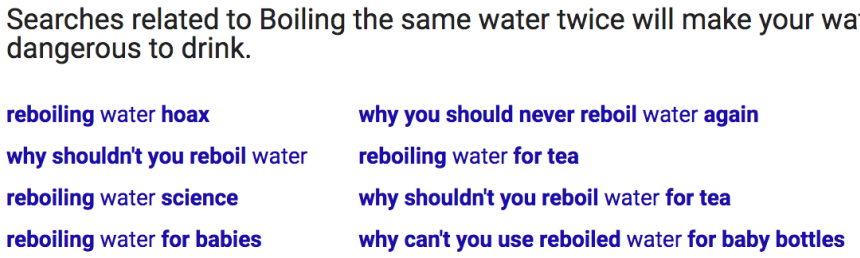

Figure 3.4: Google related searches of rumor "Boiling the same water twice will make your water dangerous to drink"

redundant data.

Rumor : Emails released by WikiLeaks confirm Hillary Clinton sold weapons to ISIS	
NOUN_VB	Emails released WikiLeaks confirm Hillary Clinton sold weapons ISIS
NOUN	Emails WikiLeaks Hillary Clinton weapons ISIS
NER	WikiLeaks Hillary Clinton ISIS
IDF_NOUN	weapons ISIS Emails
DISTANCE_NOUN_VB_3	WikiLeaks confirm Hillary
DISTANCE_NOUN_3	Hillary Clinton weapons
Google related searches	clinton's emails clinton email fox news

Table 3.3: An example output of query generator. NER stand for Name entity recognition, DISTANCE_3 stand for the word distance based query extraction method, IDF stand for the Inverse document frequency based query extraction method and NOUN_VB stand for the combination of Noun phrases and Verb phrases. Same condition for other combinations.

Chapter 4

RUMOR CRAWLER

In the previous chapter, we discuss different methods proposed to generate queries based on the rumor claims. To get rumor related data and also evaluate these query generation methods, we developed a rumor crawler subsystem. It is a key process in the rumor collection framework to collect rumor related data from the Internet. This chapter will describe in detail the Rumor Crawler subsystem.

4.1 Platforms to Collect Rumor

Nowadays, there are a lot of online platforms i.e. Google, Bing, Twitter, Instagram, Snapchat that rumors may spread inside. Among these platforms, we pick Google, Google News and Twitter as the source of data considering the popularity and user numbers. Google is the largest search engine which can return rumor related web pages from different websites based on the relevance and a page ranking algorithm when searching a rumor. Google News provides up-to-date news by aggregating the news from sources all over the world. Twitter is one of the largest social media platforms. Also, Twitter REST APIs enable developers and researchers to get access to the data easily, which are widely used for data collecting from Twitter. We will introduce them in detail in the following.

4.1.1 Google Search

Google is the most popular search engine on the Internet nowadays. It is good at returning information that matches user's information need. Usually, what users need will appear at the top of returned results. Besides, advertisements and sponsored links are clearly marked and kept separate from search results. Each search result

includes a snippet of the text which is a description of or a short extraction from the web page.

4.1.2 Google News

Google News provides comprehensive up-to-date news coverage and focuses on news. With Google Search, the results are normally ranked by relevance to the query. Somewhat differently, The ranking is date oriented and top ranked results are usually within recent news.

4.1.3 Twitter

Twitter is one of the world's largest social networks that the monthly active users of Twitter are over 300 millions[1]. It allows users to broadcast and interact with short posts called tweets, which are restricted to 140-character. Twitter users can create tweets, broadcast tweets and retweet, quote and reply other users' tweets. tweets can be sent by an app on the cell phone, desktop client or by browsing the Twitter.com website.

Academic Benefits through Using Twitter

- Twitter's wide reach is one of the main advantages. It has 300 million monthly active users and provides a platform for around 6,000 tweets in every second[2].
- The hashtags is also a good feature of Twitter, which can facilitate real-time chat through the use of hashtags. It makes Twitter a good place for information propagation and rumor spreading.
- Twitter provides REST APIs, which simplify the process to collect data from Twitter.

[1] https://www.fool.com/investing/2017/04/27/how-many-users-does-twitter-have.aspx

[2] http://www.internetlivestats.com/twitter-statistics/

- Twitter Advanced search, where users can specify operators, language, date, places, people and even sentiment during a search, gives researchers the high flexibility to satisfy the information need.

4.2 Google Crawler

To crawl data from Google, a Google crawler is built based on a previous Google crawler framework. The input of the Google crawler is the generated queries provided by the query generator mentioned at chapter 3. To scrap data from Google, we need to simulate the process of the user's searching on Google through a browser. There are mainly 5 steps during one Google search. When a user searches something in Google, an HTTP request will be sent to the target domain such as 'www.google.com', then the domain will return HTML file back to the client's browser, the browser can convert the HTML file to the real web page that user can interact with. As shown in Figure 4.1, our Google crawler simulate the process to search on Google without opening a browser. At first, we need to construct HTTP request, which simulates the action of user's search on Google, then we send the query and get the returned HTML from Google. Instead of showing the web page to the user, the crawler will extract data from the HTML file and stored the data in file system or database.

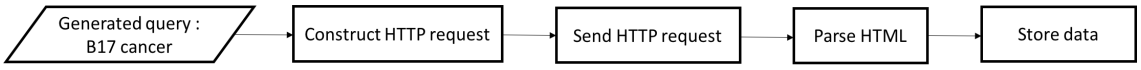

Figure 4.1: Example workflow of Google crawler

Construct HTTP Request

When a query is inserted into the search box and then the search button clicked, the query will be converted into an HTTP request, as shown in Figure 4.2. In order to construct the request, we need to figure out the request parameters used by Google. As shown in Table 4.1, we can specify the type of boolean search by using different parameters i.e search operators, language, and filter. Also, Table 4.2 shows the options

that can be used for searching a specific type of information. In order to search Google News specifically, we need to put "tbm=nws" in the request in addition.

<Original Query> : Donald Trump

< HTTP Request for Google >
https://www.google.com/search?q=Donald+Trump&hl=en

< HTTP Request for Google News>
https://www.google.com/search?q=Donald+Trump&tbm=nws&hl=en

Figure 4.2: Example HTTP request of Google crawler

Parameters	Explain
q / as_q	the search query
as_epq	matches exact phrase, same as searchphrase surrounded by quotes in searchbox
as_oq	with at least one of the search terms, same as searchterms combined with OR in searchbox
as_eq	without these searchterms, same as searchterm prefixed with - in the searchbox
hl	specifies the interface language
tbm	indicate the filter used

Table 4.1: Parameters of Google search request

Send HTTP Request

Once the HTTP request is constructed, we can send the GET request by using urllib2[3] or request[4]. The returned data is in HTML format.

[3] https://pymotw.com/2/urllib2/

[4] http://docs.python-requests.org/en/master/

Filter type	Abbreviation
Applications	app
Blogs	blg
Books	bks
Discussions	dsc
Images	isch
News	nws
Patents	pts
Places	plcs
Recipes	rcp
Shopping	shop
Video	vid

Table 4.2: Google search filter

It should be noted that Google can distinguish between human and program behavior. If sending requests frequently, the IP address may be banned by Google for hours with a 403 error returned.

Parse Google Result

BeautifulSoup4 is a Python library for pulling data out of HTML and XML files. In our system, we use it to parse the returned HTML files to extract search results and then store data in files in JSON format.

Here is an example of JSON format result:


```
{
  "title": "Onion Bunions",
  "url": "http://www.snopes.com/onion-in-your-sock-cure&sa
          =U&ved=0ahUKEwiqyOKoisPSAhUE4CYKHXRDACgQqQIIFCgA
          MAA&usg=AFQjCNGsFEwzrQYgxJVWErzGTzbUkDasRw",
  "snippet": "Like so many questionable bits of scientific
          misinformation, the claim that putting onions on
```

```
            your feet will do something unspecified that has
            to \u00a0..." ,
    "source": "snopes.com",
    "date": "Jan 24, 2017",
    "id": "1_google_news_column_1"
}
```


4.3 Twitter Crawler

To collect data from Twitter, Twitter REST APIs are widely used. Some researchers use the Twitter Streaming API, which is mentioned in chapter 3, to return a small random sample of all public statuses in real time(roughly 1%). As a result, the entire data collecting process is unpredictable and the collected data is far from complete. Also, historical data is unavailable since it is real-time data. In our system, we should be able to provide related tweets, include historical tweets, in a given time after a rumor claim is given. It makes Streaming API unsuitable for our system in the following.

Another widely used Twitter REST API is the Search API. We will introduce this API in detail.

4.3.1 Twitter Search API

Twitter Search API provides programmatic access to search Twitter data within 7 days. It can return tweets that match a specified query. The returned data is a JSON format tweet object. Within a tweet object, there are multiple key-value pairs, include entity object, user object, tweet ID, text etc.

Authorization of API Usage

To use the Twitter Search API, it is required to get authorization from Twitter. There are 4 steps to finish the authorization.

Step 1: Go to Twitter Application Management dashboard and log in.

Step 2: Create an app like https://apps.twitter.com/app/12316567/keys

Step 3: Get consumer key and secret, as shown in Figure 4.3

Step 4: Get access key and secret, as shown in Figure 4.4

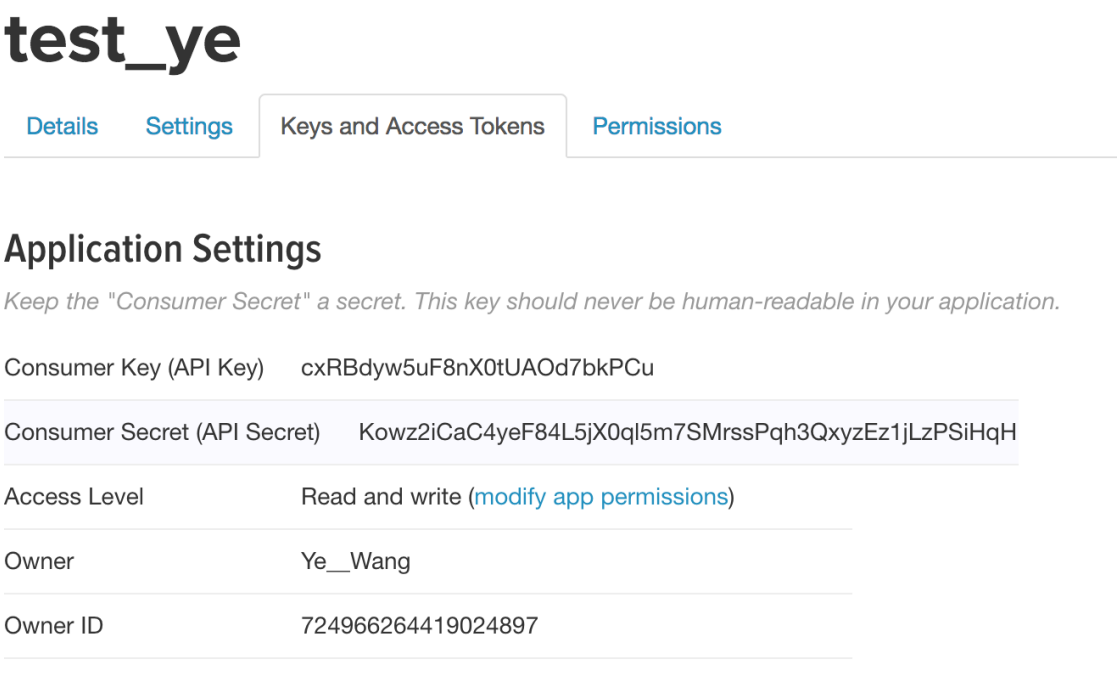

Figure 4.3: An example of consumer key and secret

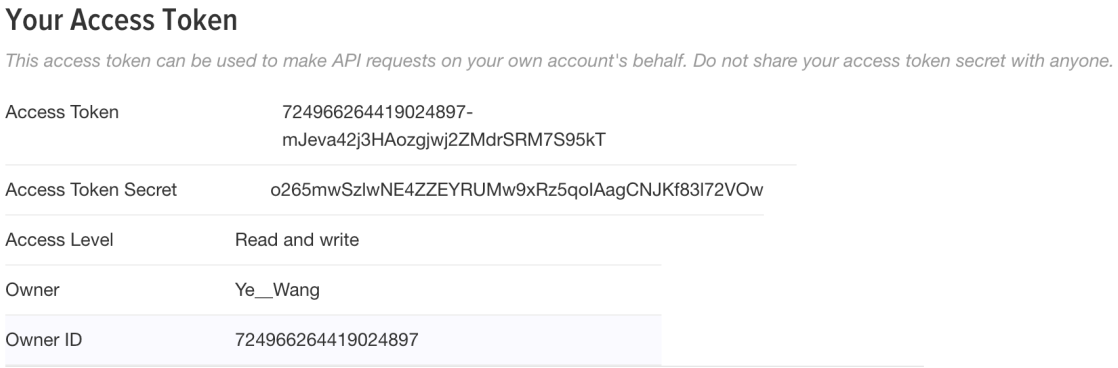

Figure 4.4: An example of access key and secret

Access API with Python

Tweepy, which is an easy-to-use Python library for accessing the Twitter API, is used to integrate API with our system. By using the key and secret we get from Twitter Application Management dashboard, the authorization can be done in a few lines Python code easily.

```
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
api.search('example search query')
```

Configure Search API Parameters

Twitter Search API return tweets that match a specified query and return a list of "SearchResult" objects. Table 4.3 shows the details of Tweepy parameters.

Parameters	Description
q	The search query string
lang	Restricts tweets to the given language, given by an ISO 639-1 code.
rpp	The number of tweets to return per page, up to a max of 100.
page	The page number (starting at 1) to return, up to a max of roughly 1500 results (based on rpp * page.
since_id	Returns only statuses with an ID greater than (that is, more recent than) the specified ID.
geocode	Returns tweets by users located within a given radius of the given latitude/longitude.
show_user	When true, prepends "<user>:" to the beginning of the tweet.

Table 4.3: Tweepy search API parameters

Limitations

However, Twitter Search API has two limitations, which place restrictions on the usage of the API in our system. As mentioned in Twitter Developer Documentation,

the Twitter Search API searches can only return a sampling of recent tweets published in the past 7 days. Considering that the propagation of a rumor can take a long period of time more than one week, the data collected by Search API is far from enough. Also, relevance, instead of completeness, is the focus of the Search API, which means that some tweets may be missing in the results.

As shown at Table 4.4, the rate limitation of Twitter Search API is another limitation. As an authorized user, the maximum rate is 180 requests / 15 minutes. One request can return 100 tweets in maximum. Theoretically, 18,000 tweets can be crawled in 15 minutes in maximum for users and 45, 000 tweets for App. For personal usage, this limitaion may be acceptable, but this limitation makes the usage of the Search API unscalable. If 100 search requests go into the system at the same time, then the API can return 450 tweets for each searched rumor on average in 15 minutes. If one search result exceed the maximum rate limitation, then no search can be done in the following 15 minutes.

Response formats	JSON
Requires authentication	YES
Rate limited?	YES
Requests / 15-min window (user auth)	180
Requests / 15-min window (app auth)	450

Table 4.4: Twitter Search API rate limitation

4.3.2 A Novel Web Crawler

As can be seen, the Twitter Search API has multiple limitations, which make it unable to crawl enough data from Twitter. In order to overcome the limitations, we move to the second method, web scraping.

Twitter Advanced Search is a web-based search engine where users can find the latest news and world events. It allows users to tailor search results to specify date

ranges, people and more. This makes it easier to find specific tweets. Considering that it is also a search engine, which is similar to Google. At beginning, we try to solve this task by sending HTTP request and resolve the returned result, which is the technique used for Google Crawler. However, the returned HTML file only contains 20 tweets in maximum. In order to get all of the returned tweets, the user need to search the query with *Twitter Advanced Search* in the browser. Then keep scrolling down the web page to the bottom to get 20 more tweets at each time until no more tweets returned. As can be seen, if there are 1,000 tweets returned, then 50 scrolling down operations are needed to get all of them. Meanwhile, the HTTP request can not simulate user's scrolling-down operation in the browser since the scrolling-down operation will triggle the browser to send another secret HTTP request to get the 20 more tweets, which is unpredictable and unable to be simulated.

To this end, a novel Twitter crawler is proposed, as shown in Figure 4.5. It uses Twitter Advanced Search as the source to get rumor related tweets. After this, Selenium, which can manipulate browser and simulate user's interaction with the browser, is used to simulate the scrolling-down operation and get all returned search results from the browser.

Twitter Advanced Search

Twitter Advanced search is available without logging in to twitter.com. Users can specify date ranges of search, which is impossible by using the Search API. As a result, it is easier to find specific tweets, especially out-of-date tweets. Also, compared with Twitter Search API, *Twitter Advanced Search* index more tweets than the Search API, especially for unpopular tweets. Table 4.5 shows the fields can be used in advanced search, we can refine search results by using any combination of the fields.

Selenium

In order to get all returned tweets from Twitter Advanced Search, we need to keep scrolling down the web page until no more tweets appear. This requirement

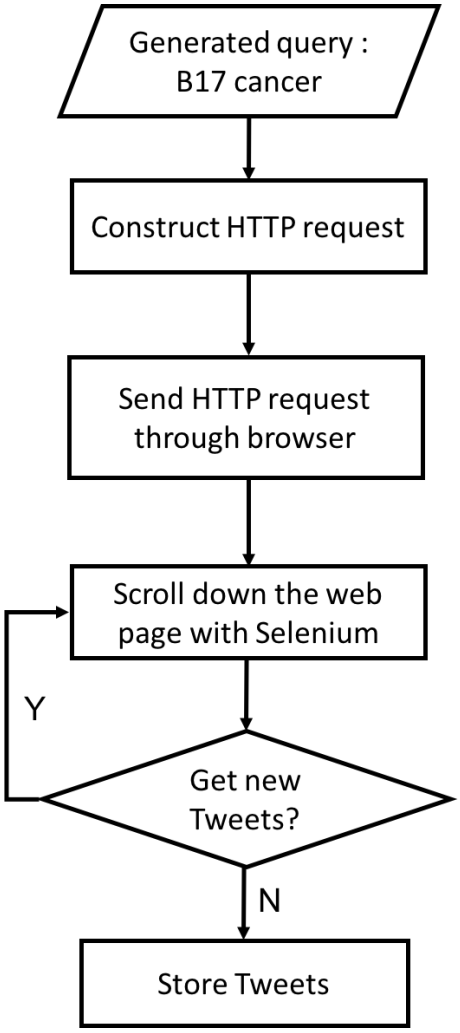

Figure 4.5: An example workflow of Twitter crawler

makes a general crawler unable to handle it. As a result, Selenium is chosen to solve the problem, which is a browser automation toolkit. Primarily, it is used to automate web applications for testing purposes. Here, we use it to do data crawling.

Selenium WebDriver provides a simple and concise programming interface which can drive a browser natively as a user either locally or on a remote machine. The supported browser includes Chrome, Firefox, InternetExplorer, Safari, and PhantomJS.

At the beginning, we choose Chrome and Firefox as the browser to get access to Advanced search, considering that Internet Explorer and Safari are platforms specified

Fileds	Example	Request format
Words		
Tweets containing all words	“Twitter” and “search”	q=Twitter%20search
Tweets containing exact phrases	“Twitter search”	q="Twitter%20search"
Tweets containing any of the words	“Twitter” or “search”	q=Twitter%20OR%20search
Tweets excluding specific words	“Twitter” but not “search”	q=Twitter%20-search
Tweets with a specific hashtag	#twitter	q=%23twitter
Tweets in a specific language	written in English	l=en
People		
Tweets from a specific account	Tweeted by “@TwitterComms”	q=from%3AtwitterComms
Tweets as replies to a specific account	in reply to “@TwitterComms”	q=to%3AtwitterComms
Tweets that mention a specific account	Tweet includes “@TwitterComms”	q=%40TwitterComms
Places		
Tweets sent from a geographic location	Tweet sent from Newark,DE within 15miles	q=near%3A"Newark%2C%20DE"%20within%3A15mi
Dates		
Tweets sent within a date range	Tweets between December 30, 2016 and January 2, 2017	q=since%3A2016-12-30%20until%3A2017-01-02

Table 4.5: Sentiment analysis of top 200 tweets

and not available in all operating system. After further tests, Chrome is chosen as the only browser used in our system because of two shortings of Firefox. First, Firefox WebDriver is unable to close the browser correctly after the task is finished while Chrome can. Second, Firefox WebDriver is slower than Chrome WebDriver.

Deploy the Crawler on Server

Considering that our system is a web-based system that can provide useful rumor related information when a rumor claim is given, our service needs to be hosted on the server instead of a personal computer. Different from a personal computer, a server usually does not have a screen, but the Twitter crawler needs a screen to open a browser visually and manipulate it with Selenium. In order to deploy the Twitter

crawler on the server side, Xvfb is used to solve it.

Xvfb or X virtual framebuffer is an in-memory display server for UNIX-like operating system. Xvfb performs all graphical operations in memory without showing any screen output. This virtual server does not require the computer it is running on to have a screen or any input device. Only a network layer is necessary. Also, in order to execute Xvfb with Python, we used a python package, PyVirtualDisplay[5], which is a Python wrapper for Xvfb. An example code of using Xvfb is shown in Figure 4.6.

```
from pyvirtualdisplay import Display

#Start Xvfb
xvfb=Display(visible=1, size=(320, 240)).start()

# Do the crawling
...

#Stop Xvfb
xvfb.stop()
```

Figure 4.6: An example code of Xvfb wrapper

[5] https://pypi.python.org/pypi/PyVirtualDisplay

Chapter 5

EXPERIMENT

We conduct experiments to evaluate our rumor collection system. First, we get a list of 50 rumor claims. After this, by using our rumor collection system, we crawled data related to these 50 rumors from both Google and Twitter.

5.1 Performance of Query Generation Methods

For each rumor claim, we generate a set of queries by using different query generation methods. These methods are based on the features extracted in chapter 3. Also, we combine some of them to improve the performance of single feature. As shown in Figure 5.1, NOUN stands for noun phrases, VB stands for verb phrases, GRS stands for tweet, NER stands for Name Entity Recognition, DIS3 stands for the word distance based query extraction method, IDF stands for the Inverse document frequency based query extraction method and NOUN_VB stands for the combination of noun phrases and verb phrases. Same condition for other combinations. It shows the number of returned tweets by using different query generation methods.

We can see NER returns more data than any other methods. The main reason is that NER based query generation method usually return a few words which are more efficient during search on Twitter. Also, with the help of Google Related Searches, numerous results are returned because the terms returned by Google Related Searches are popular queries based on the frequency of being searched, these popular queries can also effective during a search on Twitter. About shortening the size of query, both IDF based method and word distance based method contribute to more returned queries. However, based on the number of results, word distance outperforms IDF based method greatly. NOUN_DIS3 returns 5 times more results when compare with

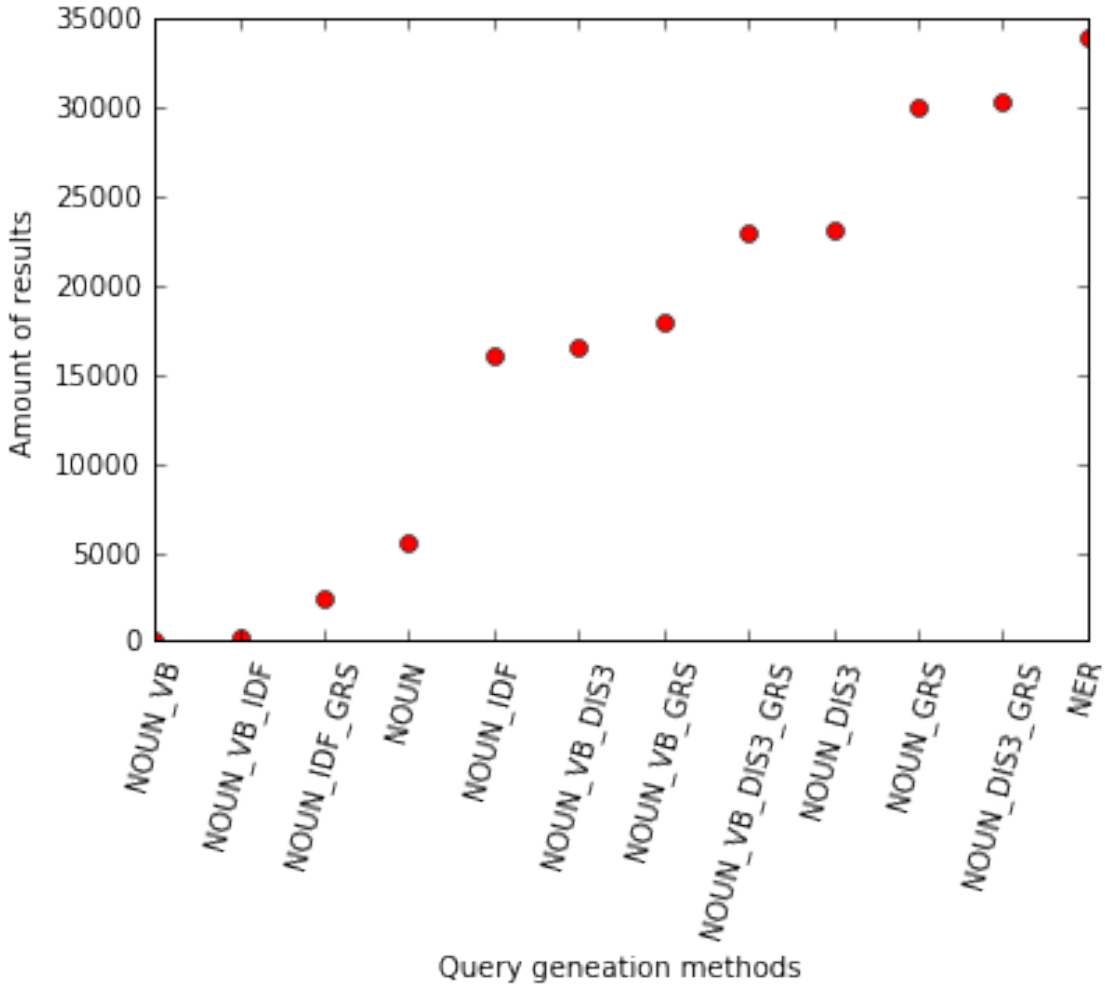

Figure 5.1: Performance of query generation methods, NOUN stands for Noun phrases, VB stands for Verb phrases, GRS stands for Google Related Searches, NER stands for Name entity recognition, DIS3 stands for the word distance based query extraction method, IDF stands for the Inverse document frequency based query extraction method, '_' stand for the combination of multiple methods

NOUN. Also, NOUN_DIS3 gets more results than NOUN_IDF. The impact of word distance based query contraction method is greater on NOUN_VB comparing with NOUN. NOUN_VB_DIS3 returns amounts of results while NOUN_VB rarely return results. A possible explanation of the different impact on NOUN and NOUN_VB is that NOUN returns relatively fewer words, sometimes, it is short enough to get sizable

results without additional contraction. Meanwhile, NOUN_VB usually returns too many words. When query contraction method applied, the returned results increase dramatically.

5.2 Compare the Number of Crawled Tweets

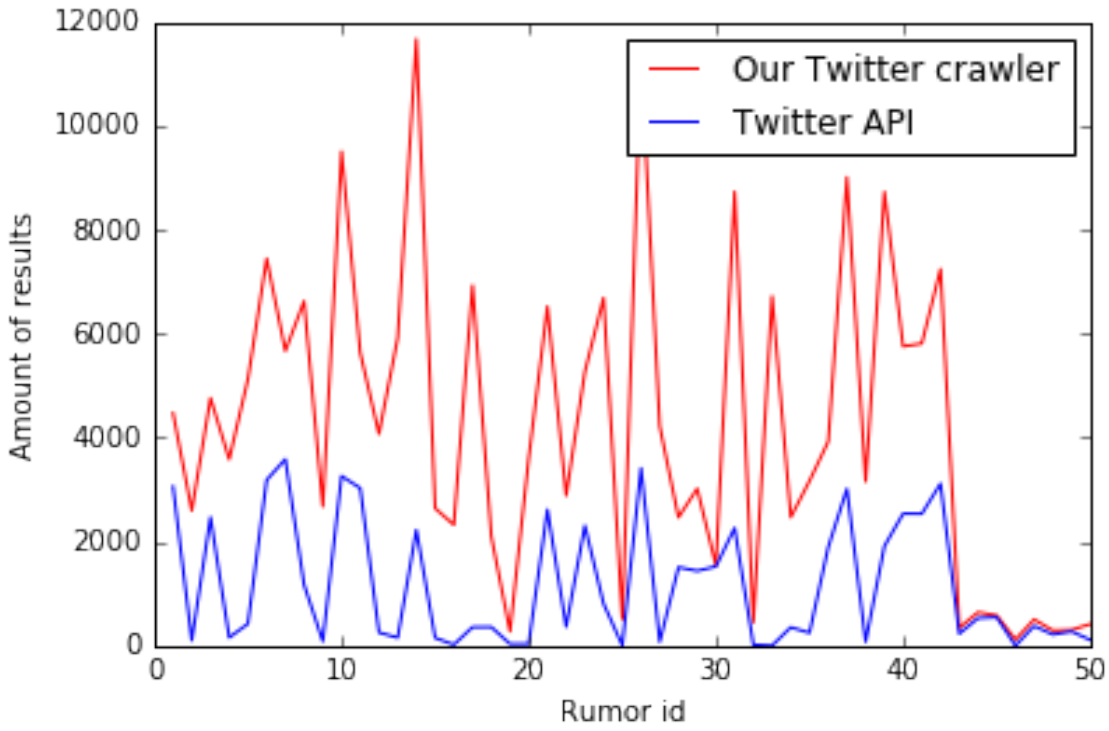

Figure 5.2: Crawled tweets by our Twitter crawler and Search API

Figure 5.2 shows the different number of tweets crawled for 50 rumors by using both our Twitter crawler and Twitter Search API. Our Twitter crawler crawls rumor related tweets monthly from January 2016 to March 2017. As can be seen, the number of results crawled by our novel Twitter crawler is always more than Twitter Search API. Furthermore, in our novel Twitter crawler, the average increasement of returned tweets is around 3.589 times compared with the Twitter Search API.

5.3 Compare the Crawled Tweets within 7 Days

Considering that Twitter API can only return results in 7 days, we compare the returned tweets in 7 days to check if our Twitter crawler can return more results than Twitter Search API in a given period of time. To easier the comparing process, we construct 10 queries that return a few tweets so that the Twitter Search API will not exceed the limitation amount, which is around 3000 tweets. Since Twitter Search API will return retweets, a reposted or forwarded message, while Advanced search does not, we filter out retweets before comparison. Figure 5.3 shows the number of tweets

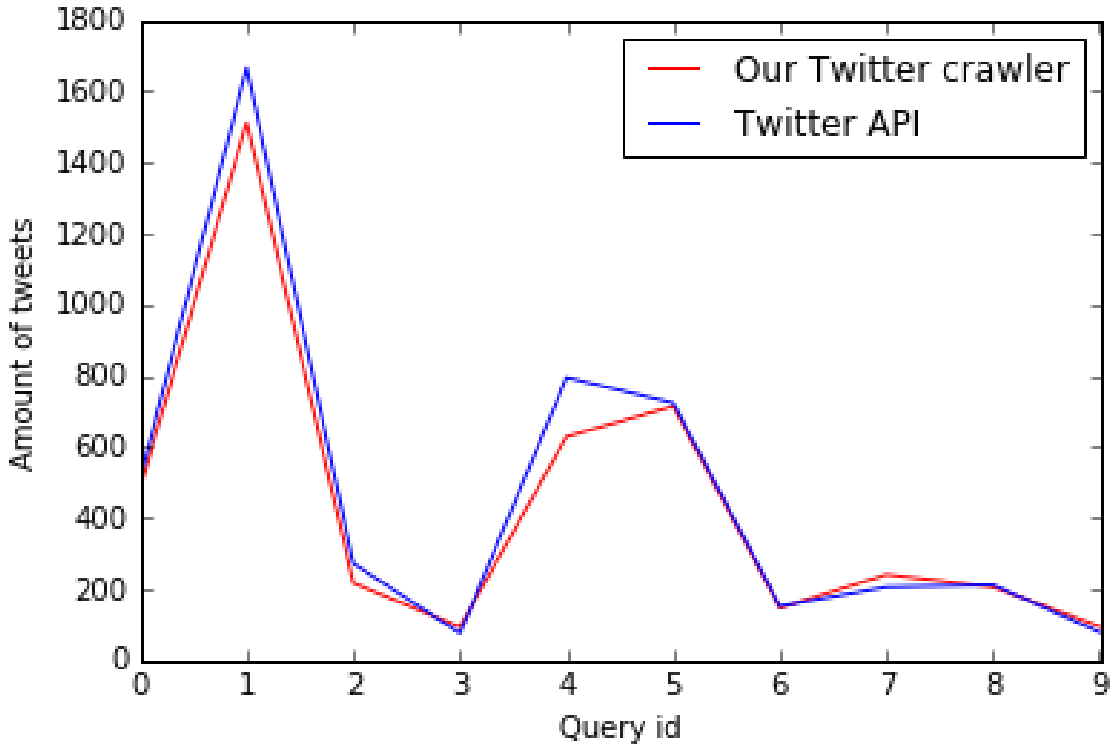

Figure 5.3: Returned tweets in 7 days

returned by these two methods. The majority of the results are same. Some data provided by Twitter Search API does not exist in the result of Advanced Search. The main reason is that Twitter Search API may return some results that do not include all

the keywords in the query. Also, some results from Advanced Search are not returned by Twitter Search API since Twitter Search API do not index all of the existing tweets.

5.4 Test the Limitations of Twitter Search API

To compare the difference between our Twitter crawler and Twitter Search API, it is necessary to test the limitations of Twitter Search API. The input is the 50 claims selected at the beginning of the experiment. By using the query generator, we generate a set of queries for each rumor and then crawl rumor related data by using Twitter search API.

5.4.1 Detect the earliest day of crawled results

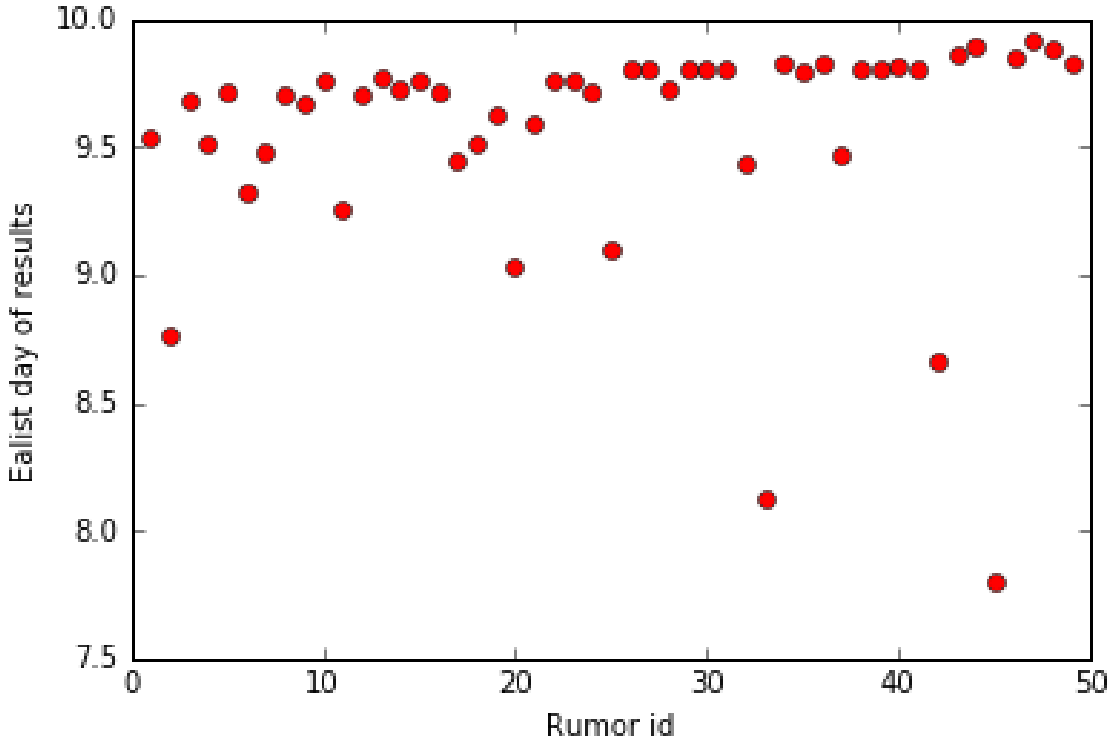

Figure 5.4: Start date of returned tweets

Based on the Twitter document[1], the Twitter Search API searches against a sampling of recent tweets published in the past 7 days. Figure 5.4 shows the detected earliest days of 50 queries. All of them are over 7 days and most of them are between 9 and 10 days.

5.4.2 Detect the date distribution of crawled results

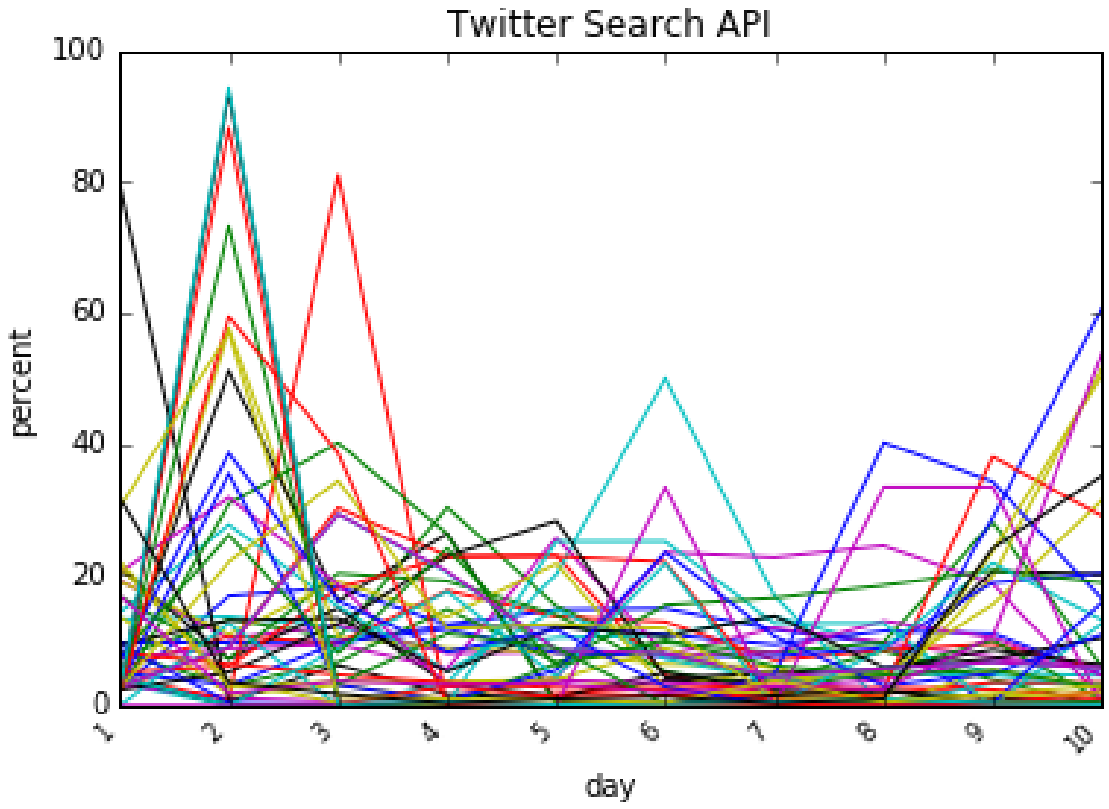

Figure 5.5: Distribution of returned tweets

Figure 5.5 shows the distribution of returned tweets over days by using Twitter Search API. It shows that the returned tweets are randomly distributed instead of preferring earlier or later tweets.

[1] https://dev.twitter.com/rest/public/search

5.5 Case Study

In order to evaluate the performance of our system and check whether crawled results are rumor related data, labeling data manually is necessary and inevitable even though it is time-consuming and painful.

There were many online judgment systems for general IR system [42, 19, 10, 41] To increase the efficiency of labeling data and make the process more user-friendly, an online judgment system is constructed and deployed based on an existing framework. First, considering the limited time we have, instead of judging all crawled data, we used a ranking function based on *TF*IDF* to retrieve 200 related data for each rumor first. Then all selected data is stored in MySQL which is the database where the web-based online judgment system can read and write data. When a user makes a judgment through the website, the result will be written to the database in real time. The online judgment system enables users to use it without bounding to locality. Also, it is easy to use. Without typing anything, a few clicks are enough to make a judgment. Furthermore, by assigning a task to multiple users, we can greatly avoid the misjudgment caused by the personal error.

The judgment system provides four options to users: believe, deny, doubt, or irrelevant. A document should be judged as 'Believe' if the document believes the rumor claim, and 'Deny' when the document does not believe the rumor. If the document is not sure about the rumor or has no opinion on the rumor, then it should be judged as 'Doubt'.

Rumor 1: Cancer is caused by a deficiency of vitamin B17, a condition that can be remedied with nutritional supplements.

As shown in Table 5.1, 86% of the Twitter are related to the rumor, which proves that our rumor collection system can return rumor related tweets effectively.

According to the analysis on Snopes.com, this rumor is a fake news. However, within these 200 tweets, most posters do believe in this rumor and do not realize that this is a fake news. This rumor is widely spread and could be very harmful.

	Amount	Percentage	Example
Believe	136	68%	You can beat cancer with vitamin B17 - just don't tell your doctor https://youtu.be/CyvSmhrlJwE via @YouTube
Deny	13	6.5%	Omg Cancer is just a vitamin B17 deficiency? And you can cure it with bicarb soda? Wake up sheeple #facebook
Doubt	23	12.5%	Cancer Caused by a Deficiency in 'Vitamin B17'?
Irrelevant	28	14%	Eat foods that have vitamin b17

Table 5.1: Sentiment analysis of top 200 tweets

Rumor 2: Clean coal technology currently makes coal a clean source of energy and helps to reduce greenhouse gas emissions.

This rumor is actually unproven. Table 5.2 shows the sentiment analysis about rumor 2. In this case, over 60 percent of the top 200 tweets are irrelevant to the rumor One possible reason is that there are over 200 relevant results but the ranking function is not good enough to pick them out from the results. So a better ranking function can be used. Another reason could be that we do not get enough relevant data, so we may need to make our Query Generator more aggressive (e.g. construct queries with fewer words). Also, it is possible that some unpopular rumors are not widely discussed on social media so that we can not find enough related rumors.

	Amount	Percentage	Example
Believe	45	22.5%	With clean-#coal technology, companies can clear regulatory hurdles & meet emissions goals http://bit.ly/2bGbLrppic.twitter.com/M81nLUkAfd
Deny	23	11.5%	@LuxSkunk clean coal uses 25% of the energy it produces to actually make "clean coal" wasteful.
Doubt	3	1.5%	@POTUS what is this magical (non existant) "clean coal technology" you speak of? #climatechange
Irrelevant	129	64.5%	@woodruffbets @thedailybeast "China uses a lot of coal, and they don't clean it."

Table 5.2: Sentiment analysis of top 200 tweets

Chapter 6

CONCLUSION

This thesis described a system for rumor collection. It consists of two parts: query generation and rumor crawling.

We proposed a query generator which can generate a set of queries based on a rumor claim. We presented multiple features i.e. NER, POS, Stop words, Stemmer, and IDF to extract keywords from the rumor claim. An algorithm based on the position of candidate keywords in the original query is proposed to shorten the size of a query. After this, Google Related Searches of generated queries are introduced to expand the query set.

We also introduced a novel rumor crawler which can crawl data from different platforms synthetically and automatically. The key idea behind our rumor crawler is simulating the operation of humans during a search. Our system overcomes the limitations of Twitter Search API by making historical tweets crawlable.

To validate our rumor collection system, extensive analysis of query generator and rumor crawler has been taken based on over 200,000 tweets. The results show that our framework can collect 3.589 times rumor related data than the widely used Twitter search API.

As part of future work, we plan to use the system to crawl more rumor related data. After this, we will use machine learning techniques to integrate these features to detect rumors.

BIBLIOGRAPHY

[1] Valencia A Andrade MA. Automatic extraction of keywords from scientific text: application to the knowledge domain of protein families. 14(7):600–607, August 1998.

[2] Niranjan Balasubramanian, Giridhar Kumaran, and Vitor R Carvalho. Exploring reductions for long web queries. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 571–578. ACM, 2010.

[3] Michael Bendersky and W Bruce Croft. Analysis of long queries in a large scale search log. In *Proceedings of the 2009 workshop on Web Search Click Data*, pages 8–14. ACM, 2009.

[4] Natalie Berry, Fiona Lobban, Maksim Belousov, Richard Emsley, Goran Nenadic, and Sandra Bucci. # whywetweetmh: Understanding why people use twitter to discuss mental health problems. *Journal of medical Internet research*, 19(4), 2017.

[5] Carlos Castillo, Marcelo Mendoza, and Barbara Poblete. Information credibility on twitter. In *Proceedings of the 20th international conference on World wide web*, pages 675–684. ACM, 2011.

[6] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific web resource discovery. In *Proceedings of the Eighth International Conference on World Wide Web*, WWW '99, pages 1623–1640, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[7] David Crystal et al. *Internet linguistics: A student guide*. Routledge, 2011.

[8] Fernando Diaz, Bhaskar Mitra, and Nick Craswell. Query expansion with locally-trained word embeddings. *arXiv preprint arXiv:1605.07891*, 2016.

[9] Gonenc Ercan and Ilyas Cicekli. Using lexical chains for keyword extraction. *Information Processing & Management*, 43(6):1705–1714, 2007.

[10] Hui Fang, Hao Wu, Peilin Yang, and ChengXiang Zhai. Virlab: A web-based virtual lab for learning and studying information retrieval models. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 1249–1250, New York, NY, USA, 2014. ACM.

[11] Kazi Saidul Hasan and Vincent Ng. Automatic keyphrase extraction: A survey of the state of the art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1262–1273, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[12] Wen Hua, Yangqiu Song, Haixun Wang, and Xiaofang Zhou. Identifying users' topical tasks in web search. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 93–102. ACM, 2013.

[13] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 216–223. Association for Computational Linguistics, 2003.

[14] Minwoo Jeong, Chin-Yew Lin, and Gary Geunbae Lee. Semi-supervised speech act recognition in emails and forums. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1250–1259. Association for Computational Linguistics, 2009.

[15] John Lafferty and Chengxiang Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–119. ACM, 2001.

[16] Chia-Jung Lee, Yi-Chun Lin, Ruey-Cheng Chen, and Pu-Jen Cheng. Selecting effective terms for query formulation. *Information Retrieval Technology*, pages 168–180, 2009.

[17] Brianna A Lienemann, Jennifer B Unger, Tess Boley Cruz, and Kar-Hai Chu. Methods for coding tobacco-related twitter data: A systematic review. *Journal of medical Internet research*, 19(3), 2017.

[18] Hongyu Liu, Evangelos Milios, and Larry Korba. Exploiting multiple features with memms for focused web crawling. In *Proceedings of the 13th International Conference on Natural Language and Information Systems: Applications of Natural Language to Information Systems*, NLDB '08, pages 99–110, Berlin, Heidelberg, 2008. Springer-Verlag.

[19] Xitong Liu, Peilin Yang, and Hui Fang. Entexpo: An interactive search system for entity-bearing queries. In *Proceedings of the 36th European Conference on IR Research on Advances in Information Retrieval - Volume 8416*, ECIR 2014, pages 784–788, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[20] Clare Llewellyn and Laura Cram. Distinguishing the wood from the trees: Contrasting collection methods to understand bias in a longitudinal brexit twitter dataset. In *ICWSM*, pages 596–599, 2017.

[21] Kuang Lu and Hui Fang. Query performance prediction and topic shift in microblog retrieval. In *Proceedings of The Twenty-Fourth Text REtrieval Conference, TREC 2016, Gaithersburg, Maryland, USA*, 2016.

[22] Kamran Massoudi, Manos Tsagkias, Maarten De Rijke, and Wouter Weerkamp. Incorporating query expansion and quality indicators in searching microblog posts. *Advances in information retrieval*, pages 362–367, 2011.

[23] Shotaro Matsumoto, Hiroya Takamura, and Manabu Okumura. Sentiment classification using word sub-sequences and dependency sub-trees. In *PAKDD*, volume 5, pages 301–311. Springer, 2005.

[24] Yutaka Matsuo and Mitsuru Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(01):157–169, 2004.

[25] Marcelo Mendoza, Barbara Poblete, and Carlos Castillo. Twitter under crisis: Can we trust what we rt? In *Proceedings of the first workshop on social media analytics*, pages 71–79. ACM, 2010.

[26] Tetsuji Nakagawa, Kentaro Inui, and Sadao Kurohashi. Dependency tree-based sentiment classification using crfs with hidden variables. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 786–794. Association for Computational Linguistics, 2010.

[27] Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. Improved part-of-speech tagging for online conversational text with word clusters. Association for Computational Linguistics, 2013.

[28] Vahed Qazvinian, Emily Rosengren, Dragomir R Radev, and Qiaozhu Mei. Rumor has it: Identifying misinformation in microblogs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1589–1599. Association for Computational Linguistics, 2011.

[29] Jacob Ratkiewicz, Michael Conover, Mark Meiss, Bruno Gonçalves, Snehal Patil, Alessandro Flammini, and Filippo Menczer. Truthy: mapping the spread of astroturf in microblog streams. In *Proceedings of the 20th international conference companion on World wide web*, pages 249–252. ACM, 2011.

[30] A. Rungsawang and N. Angkawattanawit. Learnable topic-specific web crawler. *J. Netw. Comput. Appl.*, 28(2):97–114, April 2005.

[31] Takeshi Sakaki, Fujio Toriumi, and Yutaka Matsuo. Tweet trend analysis in an emergency situation. In *Proceedings of the Special Workshop on Internet and Disasters*, page 3. ACM, 2011.

[32] Ahmed I. Saleh, Arwa E. Abulwafa, and Mohammed F. Al Rahmawy. A web page distillation strategy for efficient focused crawling based on optimized nave bayes (onb) classifier. *Applied Soft Computing*, 53:181 – 204, 2017.

[33] Philip J Stone, Dexter C Dunphy, and Marshall S Smith. The general inquirer: A computer approach to content analysis. 1966.

[34] Tetsuro Takahashi and Nobuyuki Igata. Rumor detection on twitter. In *Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS), 2012 Joint 6th International Conference on*, pages 452–457. IEEE, 2012.

[35] Elizabeth Closs Traugott. On the rise of epistemic meanings in english: An example of subjectification in semantic change. *Language*, pages 31–55, 1989.

[36] Peter D Turney. Learning algorithms for keyphrase extraction. *Information retrieval*, 2(4):303–336, 2000.

[37] Soroush Vosoughi. *Automatic detection and verification of rumors on Twitter*. PhD thesis, Massachusetts Institute of Technology, 2015.

[38] Yue Wang, Hao Wu, and Hui Fang. An exploration of tie-breaking for microblog retrieval. In *Proceedings of the 36th European Conference on IR Research on Advances in Information Retrieval - Volume 8416*, ECIR 2014, pages 713–719, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[39] Ke Wu, Song Yang, and Kenny Q Zhu. False rumors detection on sina weibo by propagation structures. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 651–662. IEEE, 2015.

[40] Peilin Yang and Hui Fang. Evaluating the effectiveness of axiomatic approaches in web track. In *Proceedings of The Twenty-Second Text REtrieval Conference, TREC 2013, Gaithersburg, Maryland, USA, November 19-22, 2013*, 2013.

[41] Peilin Yang and Hui Fang. A reproducibility study of information retrieval models. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 77–86, New York, NY, USA, 2016. ACM.

[42] Peilin Yang, Hongning Wang, Hui Fang, and Deng Cai. Opinions matter: a general approach to user profile modeling for contextual suggestion. *Information Retrieval Journal*, 18(6):586–610, 2015.

[43] Chengxiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 403–410. ACM, 2001.

[44] Chengzhi Zhang. Automatic keyword extraction from documents using conditional random fields. *Journal of Computational Information Systems*, 4(3):1169–1180, 2008.

[45] Renxian Zhang, Dehong Gao, and Wenjie Li. What are tweeters doing: Recognizing speech acts in twitter. *Analyzing Microtext*, 11:05, 2011.

[46] Renxian Zhang, Dehong Gao, and Wenjie Li. Towards scalable speech act recognition in twitter: tackling insufficient training data. In *Proceedings of the Workshop on Semantic Analysis in Social Media*, pages 18–27. Association for Computational Linguistics, 2012.

[47] Zhihua Zhang and Man Lan. Estimating semantic similarity between expanded query and tweet content for microblog retrieval. In *TREC*, 2014.