5-2015

Tweetement: Pseudo-relevance Feedback for Twitter Search

Kanatbay Bektemirov
University of Arkansas, Fayetteville

TWEETEMENT: PSEUDO-RELEVANCE FEEDBACK FOR TWITTER SEARCH

An Undergraduate Honors College Thesis
in the
Department of Computer Science and Computer Engineering
College of Engineering
University of Arkansas
Fayetteville, AR

by

Kanat Bektemirov
bektemir@uark.edu

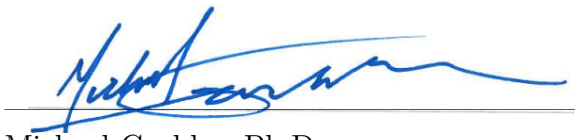
April 2015
University of Arkansas

This thesis is approved.

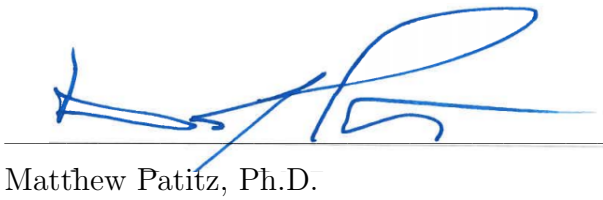
Thesis Advisor:

Susan Gauch, Ph.D., Chair

Thesis Committee:



Michael Gashler, Ph.D.



Matthew Patitz, Ph.D.

Abstract

Microblogging platforms such as Twitter let users communicate with short messages. Due to the messages' short content and the users' tendency to type short queries while searching, it is particularly challenging to locate useful tweets that match user queries. The fundamental problems of word mismatch due to ambiguity are especially acute. To solve this problem, this thesis explores and compares multiple automatic query expansion methods that involve the most frequent hashtags and keywords. We built a Web service that provides real-time Twitter Search results incorporating automatic query expansion. Six pseudo-relevance feedback methods were studied and the numbers indicate that results without query expansion perform just as well as results with query expansion. However, the expanded queries find different relevant tweets than the original query, indicating, from multiple methods, that combining the results is a fruitful area for future investigations.

Keywords: *microblog, Twitter Search, query expansion, pseudo-relevance feedback, Web service*

Contents

1 Introduction **1**
1.1 Background 1
1.2 Motivation 1
1.3 Goals 2
1.4 Overview 2

2 Related Work **5**

3 Approach **7**
3.1 Web Service 7
3.2 Data Retrieval 10
3.3 Query Expansion 11
3.4 Performance Measure 13

4 Evaluation **15**
4.1 Data Selection 15
4.2 Results & Discussion 17

5 Conclusion **19**

References **20**

1 Introduction

1.1 Background

Twitter (`https://twitter.com/`) is an online social networking and microblogging service that enables users to send 140-character messages called tweets. With 288 million monthly active users and 500 million tweets sent per day[1], Twitter contains a wealth of information about people, products, companies, sports teams, events, and other topics. In addition to using the service as a medium to share information, Twitter is also used to find recent tweets about a particular topic or event. Twitter Search is a prominent feature of the service, serving 2.1 billion search queries per day. Twitter Search does a decent job of displaying results for user's search query; however, it mainly relies on a keyword-based search that may not contain the most interesting results. Additionally, the Twitter Search REST API[2] tends to work less effectively than the search feature on Twitter.com. For example, Twitter.com search handles auto-corrects, has more up-to-date indices, and overall has more indexed tweets. Since the Twitter Search REST API does not behave exactly like the search feature on Twitter.com, throughout this paper, the term Twitter Search refers to the Twitter Search REST API.

1.2 Motivation

Twitter is a platform through which millions of users have live conversations about various topics in real-time. To give some context, 5,700 tweets are sent per second on average, with a record of 10,300 tweets per second during the 2014 FIFA World Cup Final[3]. Twitter Search performs a keyword-based search that tends to retrieve a greater number of results and includes tweets that may not be relevant to the intent of the search. If a user searches for *"James"* on Twitter, it is unpredictable whether the user sees results about LeBron James (NBA player), James Franco (actor), James Rodriguez (soccer player), or other James-related tweets. Although Twitter Search does a good job of ranking results based on current trends and conversations, we are interested if the search results through the use of query expansion via pseudo-relevance feedback are more interesting than the top Twitter Search results.

[1] About Twitter, Inc. (`https://about.twitter.com/company`)
[2] Representational state transfer (REST) application programming interface (API)
[3] Facebook, Twitter Set Usage Records for World Cup Final (`http://on.wsj.com/1mArJUn`)

```
[GET]  https://api.twitter.com/1.1/search/tweets.json
            ?q=James&result_type=popular&lang=en

@James_Yammouni: #JamesFollowSpree IM DOING A FOLLOW SPREE!
        IM GOING TO FOLLOW AS MANY OF YOU UNTILL I CANT ANYMORE!!
@James_Yammouni: It's my day in Poland today ... Woman's dayyyyyyyy
@James_Yammouni: FOLLOW SPREE IN 10 MINUTES! #FOLLOWjames
@James_Yammouni: Hold you tight straight through the daylight,
        I'm right here when you gonna realise ... that I'm your cure...
@James_Yammouni: Just woke up and my hearts racing and I'm out of
        breath I think I was having a nightmare
```

Figure 1.1: Top 5 Twitter Search results for *James*.

1.3 Goals

Due to increasing trends like form autocompletions, we believe users do not always type complete queries. Although it is nearly impossible to show the user exactly what he or she finds interesting, we are interested if we can improve the search experience on Twitter. We would like to take the results retrieved by the original Twitter Search query submitted via its REST API[4], text mine these results, do query expansion with new terms, and display the most interesting conversations that are happening now. This thesis consists of two parts: 1) building a Web service on top of Twitter Search that serves search queries; 2) exploring and comparing multiple query expansion techniques to find which, if any, lead to users rating the results as more interesting. Furthermore, the reader should note that the outcomes of these experiments largely depend on the current implementation of the Twitter Search API.

1.4 Overview

The Web service built for this project can be found at: `http://tweetement.com` (or `http://tweetement0.appspot.com`). The website handles user input, places the query in a background job, and displays the final results. The technology stack consists of: AngularJS JavaScript framework, Bootstrap front-end framework, Google Cloud Platform, and the Twitter API. More technical details are discussed in Chapter 3.

[4]GET search/tweets (`https://dev.twitter.com/rest/reference/get/search/tweets`)

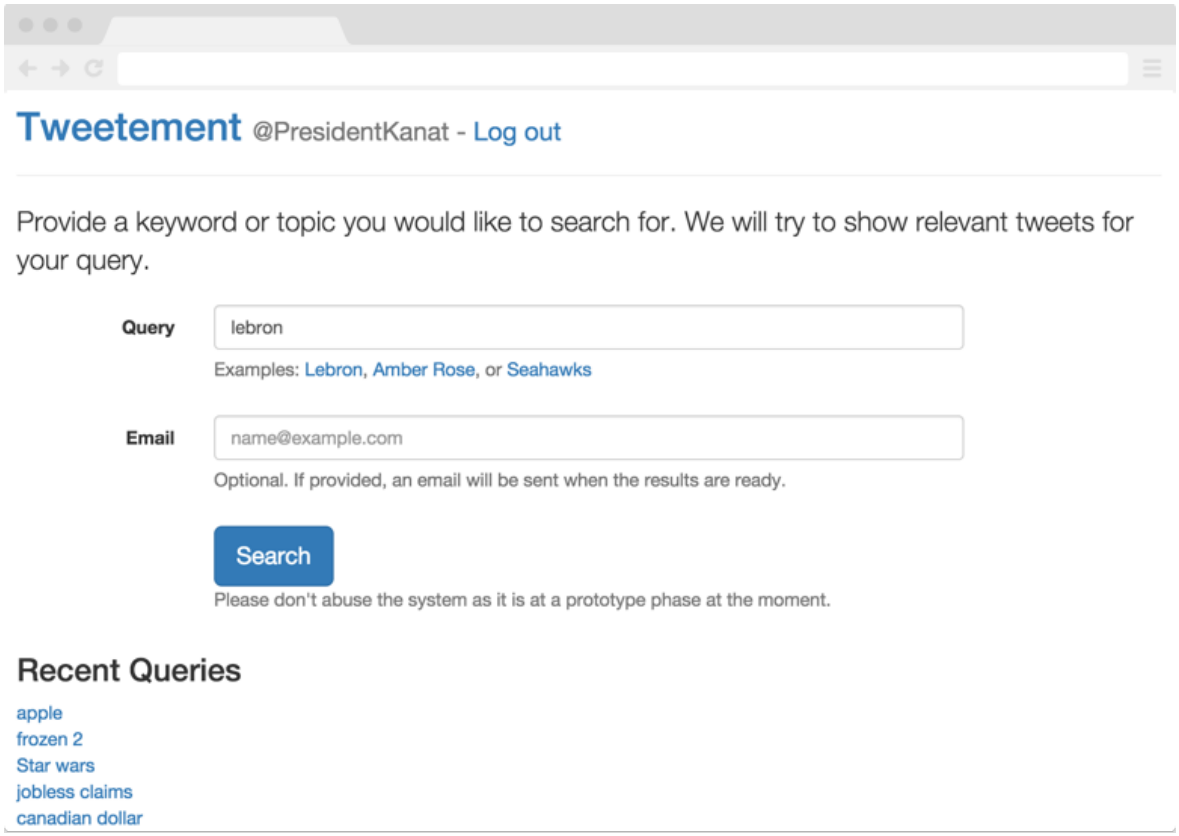


Figure 1.2: Tweetement main screen.

Query Expansion

Query expansion is the process of adding additional terms to the existing search query to generate an expanded search query, with the objective of retrieving more relevant and interesting search results. For example, the query term *"NCAA"* might be expanded to *"NCAA Football"* or *"NCAA March Madness"*, and one is more appropriate than the other depending on the time of the year or current trends. A well-formed query expansion technique helps the end user formulate better search queries that lead to better search results. Therefore, it is important to pick the right expansion terms.

Pseudo-relevance feedback is a general technique used for automatic query expansion. The idea is to assume some top-N results are accurate or relevant to the initial query and pick expansion terms from these results. This process is called automatic because of the implicit user interaction. Pseudo-relevance feedback generally leads to improved retrieval in search engines. However, pseudo-relevance feedback has obvious drawbacks such as *query drift*. Query drift happens when the focus of the search topic shifts to an unintended topic caused by improper expansion[8]. Chapter 2 talks more about query expansion.

In this paper, we explore whether or not query expansion via pseudo-relevance feedback produces more interesting search results than Twitter Search. Six query expansion methods were explored and their results were shown to the user in random order. The user then rated each result (tweet) as either *Interesting, Neutral,* or *Not Interesting*. Chapter 3 discusses these methods in detail. Chapter 4 discusses the performance outcomes of the methods.

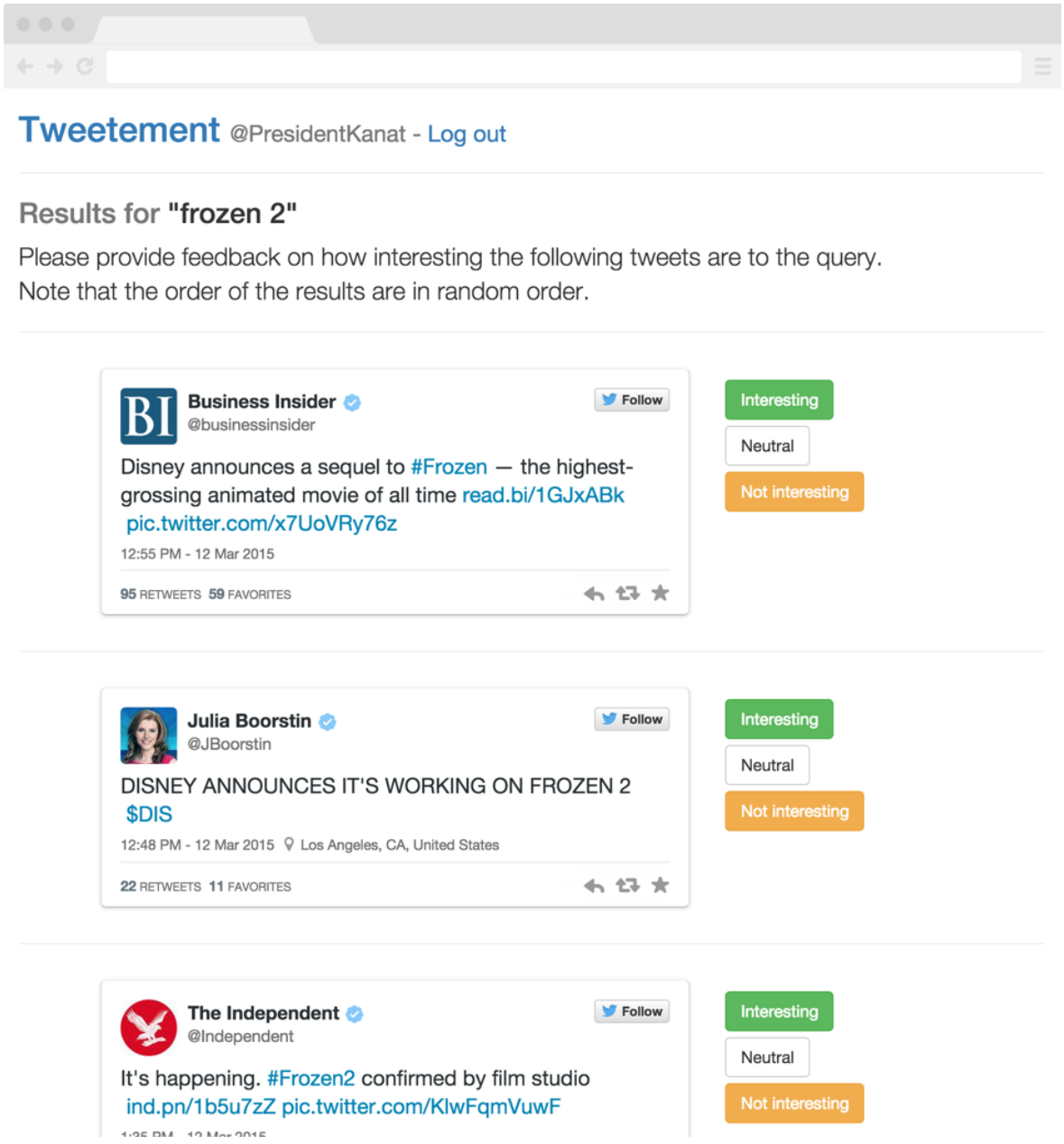


Figure 1.3: Search results page with expanded queries.

2 Related Work

With the increasing amount of information on the Internet, the number of search queries performed on various Web search engines continues to grow. These queries are generally very short; therefore, straightforward keyword matching across documents may not always yield relevant results. Word mismatch refers to the issue in which users often use different words to describe concepts in their queries than authors use to describe the same concepts in their documents[1]. The primary problems in ambiguity stem from hyponymy (one word has multiple meanings) and synonymy (multiple words have the same meaning). A number of techniques for dealing with word mismatch have long been studied.

A popular approach for dealing with word mismatch is query expansion. The goal of query expansion is to expand the original query with other words that best capture the user's intent, or that produce a more useful query that is more likely to retrieve more relevant documents[2]. The technique is particularly effective when the user query is vague, short, or lacks useful keywords pertaining the intended topic. This process can be manual via explicit relevance feedback, or automatic via pseudo-relevance feedback.

Relevance feedback is a technique designed to improve search results based on the user's feedback. Relevance feedback is an iterative process where the user assesses the relevance of documents returned in response to an initial, 'tentative' query[3]. The user goes through the initial query results and marks whether each document is relevant or not. Based on the user's feedback, a new query is submitted with the initial query, expanded with terms taken from the results marked as relevant. This procedure, known as explicit feedback, is repeated until the user is satisfied with the results. Relevance feedback can be effective; however, the technique puts a huge burden on the user. Pseudo-relevance feedback (also known as implicit feedback, local feedback, blind feedback, or adhoc) is a technique designed to effectively guess what the user might find interesting without having the user explicitly mark the search results. In pseudo-relevance feedback, the top results are assumed to have a higher precision and that keywords in those results are assumed to represent the intended search topic. It has proven to be an effective technique in numerous studies and experiments[1].

The concept of microblogging is fairly new and many traditional information retrieval techniques do not apply as nicely. It is observed that people search Twitter to find temporarily relevant information (e.g current events) and information related to people. Additionally, it appears people repeat Twitter queries to monitor the as-

sociated search results over time, while they change and develop Web engine queries to learn about a topic[4]. Researchers have been exploring various automatic query expansion techniques on Twitter and other microblogging sites.

Lau, Li, and Tjondronegoro performed an experiment using the TREC '11 dataset in which they proposed term-based and pattern-based features with distributed weights to retrieve information from Twitter. The flow is to (1) use q retrieve first 100 relevant tweets, (2) sort them based on time, R, (3) form training set using R, (4) form expanded queries, Q, using terms from R, (5) use Q to retrieve 1000 tweets and sort based on time, and (6) display top 30 tweets. For term-based feature, they used a standard TF-IDF weighting scheme. For topical or pattern based feature, they adopted the Frequent Pattern Mining approach to find the closed pattern. The baseline run using terms frequency feedback without weighting outperformed runs with weighted terms and patterns. It is unclear how the baseline run would compare to a run without feedback[5].

Efron performed a similar experiment in which the author found that the baseline run with terms frequency feedback was slightly more effective than the baseline run without feedback. Additionally, the author proposed a method of pseudo-relevance feedback based on hashtags. Using data over a 24-hour period using Twitter's Streaming API and 29 topic queries, the author found that retrieval using query expansion with hashtags gave a marginal improvement over the two baseline runs[6].

Bandyopadhyay, Mitra, and Majumder took a slightly different approach for Twitter search query expansion. In their experiment, the authors managed to improve retrieval effectiveness by using external corpora as a source for query expansion terms. Specifically, the authors used the now deprecated Google Search API to retrieve page titles from Google Search and used those to expand queries. The five most frequent word-level n-grams ($n = 1, 2, 3$) were added to the original query. One of the runs, $R3$, contained the results that were retrieved using the Google Search API with title words ($n = 1$) sorted in descending order by their frequencies. The most frequent five words were used to perform a new retrieval. Interestingly, the new query did not include the original query – this process is known as query reformulation, as opposed to query expansion. This technique resulted in significant improvements over the baseline run for the top fifty results per query[7].

Current published papers we have discovered all used a TREC dataset or some other pre-fetched dataset to perform queries on. However, we performed our experiment with actual Twitter users and real-time Twitter Search results which may be less subject to data generalization.

3 Approach

We built the Web service *Tweetement* for this project. This chapter describes the architecture of Tweetement, data retrieval process from Twitter Search API, the six query expansion methods that were explored, and the way we measured performance. Source code of the project is open-source and is hosted on GitHub at the following URL:

`https://github.com/Bekt/tweetement`

3.1 Web Service

Tweetement is a Web service that handles user input, performs query expansion via pseudo-relevance feedback, and displays the results of the six methods in random order where the user can provide feedback for each result (tweet). The service was built in a way such that it is easily scalable and maintainable. The following is the flow of the application:

1. User submits a search query.
2. The query is put in a queue to process.
3. The query is popped from the queue, automatic query expansion is performed, and the results are stored in the datastore.
4. User views the search results and optionally provides feedback for each result.

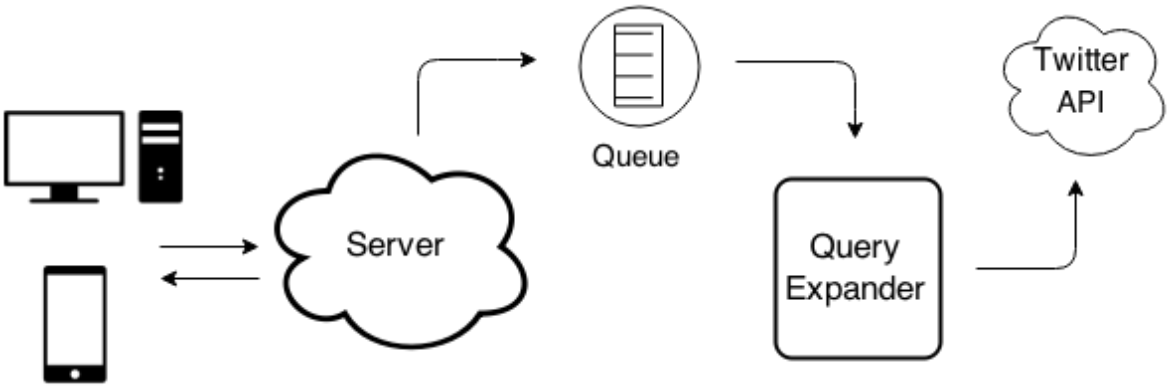


Figure 3.1: Tweetement architecture.

The front-end of the website was built in a Single Page Application (SPA) fashion to provide fluid user experience. In order to accomplish SPA, open-source Bootstrap and AngularJS frameworks were used. Bootstrap is a popular open-source front-end framework that focuses on responsive design so that the layout of the website adjusts per user's device – mobile or desktop. AngularJS is a popular open-source JavaScript framework that makes building SPAs easy by providing a MVC-like architecture on the client-side. Twitter's `widget-js` JavaScript library was used to embed tweets into the search results page.

The website was hosted on Google App Engine and used features such as the Task Queue API and the NDB Datastore API. App Engine makes it simple to build, deploy, and maintain scalable applications on Google's own infrastructure. Since each of the six techniques could make multiple network calls to the Twitter Search API, it was best to put the queries in a background queue instead of making the client's request wait for a long period of time. The Task Queue API was used to schedule background jobs when the user submitted a query. The Query Expander then picked up the job to perform query expansion. All data were stored in schemaless object datastore, NDB, which comes with convenient features such as automatic caching. The back-end of the application was written in Python.

User authentication was handled via Twitter OAuth for two reasons. First, we did not want to roll out our own user authentication over unencrypted connections. Second, we followed Twitter's recommended practice to communicate properly with the Twitter API. Without Twitter authentication, the application had a rate limit of 450 API requests per 15 minutes for all users; whereas with Twitter Authentication, the rate-limit was extended to 180 API requests per user every 15 minutes. `webapp2` sessions and `simpleauth` libraries were used to implement user authentication. The author found a major bug in `simpleauth` while working on Tweetement and submitted a patch accordingly.

On the search results page, the user was shown the top-5 results from the original search query plus the top-5 results from each of the query expansion methods. Thus, the total number of unique tweets for the search query could be up to 35. All tweets were shown in random order. The application provided a feedback system where the user could rate each search result as *Interesting*, *Neutral*, or *Not Interesting*. The user could change his/her rating at any time.

The application provided REST APIs for the first-party client and other third-parties to interact with. The following methods were implemented. All methods accepted and returned JavaScript Object Notation (JSON) objects.

```
[POST]  /api/enqueue
  Submit a search query.
Parameters:
  (string) query: search query
  (string) email: email to notify when the request completes (optional)
Returns:
  (int) qid: enqueued query ID

[GET]  /api/result
  Retrieve results for a particular query.
Parameters:
  (int) qid: query ID
Returns:
  (int) qid: query ID
  (string) query: original query
  (string) status: qtatus of the job. Working | Cancelled | Done
  (date) created: created datetime of the job
  (date) updated: updated datetime of the job
  (list) hashtags: top-10 hashtags used for query expansion
  (list) keywords: top-10 keywords used for query expansion
  (list) status_ids: tweet IDs of the expanded search result

[POST]  /api/feedback
  Provide feedback for a particular result.
Parameters:
  (int) qid: query ID
  (int) sid: tweet ID
  (int) score: user's rating (0, 1, or 2)
Returns:
  None

[GET]  /api/scores
  Retrieve feedback submitted by the current user for
  results of a particular query.
Parameters:
  (int) qid: query ID
Returns:
  (list) items:
      (int) qid: query ID
      (int) uid: user ID
      (int) sid: tweet ID
      (int) score: user's rating (0, 1, or 2)
```

Figure 3.2: Publicly available API methods.

3.2 Data Retrieval

Since Tweetement was built on top of the Twitter Search API, it is important to note that our experimental results are heavily dependent on the quality of the responses received from the Twitter Search API. Furthermore:

> [The Twitter Search API] allows queries against the indices of recent or popular tweets and behaves similarly to, but not exactly like the Search feature available in Twitter mobile or web clients, such as Twitter.com search.
>
> ... it's important to know that the Search API is focused on relevance and not completeness. This means that some tweets and users may be missing from search results.[1]

Furthermore, we observed that Twitter Search API behaved unpredictably at times. For example:

- Ranking of the results were not consistent. That is, if two exact same queries were submitted simultaneously, the returned tweets were not always the same or have the same ranking. This affects two things:
  (a) The top-5 results shown to the user, affecting the average precision calculations. Two sets of five identical tweets could have different average precision scores because of ordering.
  (b) Since the results were not always the same, this could affect the order of the most occurred hashtags and keywords.
- When requesting *popular* tweets only, regardless of the value of the `count` parameter in the API request, sometimes there were only up to 10 results in the response, even for trending topics.

The Query Expander connected to the Twitter Search API by first requesting *popular* result types for a query. If there were not enough results, the Query Expander filled up the remaining spots with *mixed* result types for the query. The difference is that `popular` returns only the most popular results in the response, whereas `mixed` returns both popular and real-time results in the response. As a result, the results we showed to the users were sorted by popularity in a way. Note that we limited the search results to English tweets only by providing the `lang='en'` parameter.

[1] The Search API (`https://dev.twitter.com/rest/public/search`)

3.3 Query Expansion

We were interested to see if we could provide improved search results by performing automatic query expansion via pseudo-relevance feedback. That is, we started off by first retrieving 200 search results for the user's original query, q, from the Twitter Search API. Let's denote this initial results set, R, and assume these results are somewhat relevant. The question now is how to expand the user's query. We explored a few ways to automatically generate additional query terms by analyzing the results in R.

The users of Twitter categorize their messages or put more emphasis on a certain topic by using *hashtags* in their tweets. Hashtags are a big part of the Twitter culture that help users have conversations about a common topic across the world.

The ten most frequently occurring hashtags, h, and keywords, k, were extracted from R. Only unique tokens in each tweet and non-stop words were taken into consideration for this process. Tokens were lowercased and any punctuation was removed. The original query tokens were also excluded from h and k.

Around 900 Twitter-specific stop words were generated based on a 1.5 million tweets dataset.[2] During query expansion, tokens that appear in this list are ignored. The original query, however, could still contain such tokens. The complete list is available under the file `stoplist.txt` and the script that was used to generate the list is accessible under the file `scripts/twitter_stopwords.py`.

The user was shown the list of tweets that were retrieved by expanded search queries in random order. The top-5 results for the original query were also included to compare the baseline method.

We studied six different ways to generate additional query terms. To compare the performances of these methods, we denoted *method 0* as the baseline method where the search query was performed with the original query. Methods 1-5 were two-pass techniques and method 6 was a three-pass technique. We expected method 1, method 2, or method 6 to perform the best. No kind of ranking arrangement was done for methods 1-5.

Method 1: Top hashtag

$$\text{New query: } q + h[0]$$

A new search query was performed with the original query concatenated with the

[2]Twitter Sentiment Analysis Training Corpus
(`http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/`)

most occurring hashtag. The resulting tweets all included the tokens q and/or $h[0]$ in the body of the tweet. The first five results were shown to the user.

Method 2: Top keyword

$$\text{New query: } q + k[0]$$

A new search query was performed with the original query concatenated with the most occurring keyword. The resulting tweets all included the tokens q and/or $k[0]$ in the body of the tweet. The first five results were shown to the user.

Method 3: Top hashtag + top keyword

$$\text{New query: } q + h[0] + k[0]$$

A new search query was performed with the original query concatenated with the most occurring hashtag and keyword. The resulting tweets all included the tokens q, $h[0]$, and/or $k[0]$ in the body of the tweet. The first five results were shown to the user.

Method 4: Top two hashtags

$$\text{New query: } q + h[0] + h[1]$$

A new search query was performed with the original query concatenated with the two most occurring hashtags. The resulting tweets all included the tokens q, $h[0]$, and/or $h[1]$ in the body of the tweet. The first five results were shown to the user.

Method 5: Top two keywords

$$\text{New query: } q + k[0] + k[1]$$

A new search query was performed with the original query concatenated with the two most occurring keywords. The resulting tweets all included the tokens q, $h[0]$, and/or $h[1]$ in the body of the tweet. The first five results were shown to the user.

Method 6: Most top-10 hashtags and keywords occurrences

$$\text{New query: } q + h[0] \ OR \ q + k[0]$$

A new search query was performed with the above string. The special OR operand is a boolean operand that Twitter Search recognizes. In a way, we combined methods 1 and 2.

The top 200 results were retrieved with the expanded query. From there, each tweet was given a score. A tweet had a higher score if it contained hashtags or keywords that occurred in h or k. The five results with the highest scores were shown to the user.

```
scores = dict()
for tweet in results:
  for token in unique_tokens(tweet.message):
      if token in h or token in k:
          scores[tweet.id] += 1
```

Figure 3.3: Scoring results by maximal hashtag and keyword occurrences.

3.4 Performance Measure

Average Precision (AP) is a popular and standard evaluation metric in the information retrieval field that takes rank order into consideration. As a result, average precision is sensitive to the order of the results. When working with small number of results (as we are in this thesis), changes to the ranking of search results may have significant impact on the average precision calculation.

$$AP = \frac{1}{N} \sum_{k=1}^{N} P(k) \times rel(k)$$

N : *number of results shown to the user*
$rel(k)$: *indicates whether result k is relevant or not (1 or 0)*
$P(k)$: *precision of result k.*

$$P(k) = \frac{|\text{relevant results up until } k|}{k}$$

The average precision equation was slightly modified so that it took into account the user's feedback: (0) *Not Interesting*, (1) *Neutral*, and (2) *Interesting*. Since each result contained at least the original query, we assumed every result is relevant. We refer to this as average weighted precision.

$$AP' = \frac{1}{N} \sum_{k=1}^{N} P'(k)$$

$$P'(k) = \frac{\text{sum of feedback scores up until result } k}{k \times max\_weight}$$

$$max\_weight = 2 \text{ (Interesting)}$$

Mean average precision is used to compute the mean of average precision scores for a set of queries.

$$MAP = \frac{1}{Q} \sum_{q=1}^{Q} AP(q)$$

Similarly, mean average weighted precision is used to compute the mean of average weighted precision scores for a set of queries.

$$MAP' = \frac{1}{Q} \sum_{q=1}^{Q} AP'(q)$$

4 Evaluation

Over 30 people participated as volunteers and provided feedback for their search queries results. Most of the participants were students at the CSCE Department at the University of Arkansas or members of the *HH Data Hackers* Facebook group[1]. The age demographic was early 20's. In this chapter, we analyze the responses from these volunteers and discuss some of the observations.

It should be noted that the computations are volatile due to the small sample size, selection bias, the subjectivity of users' feedback, and the quality of the responses from the Twitter Search API.

4.1 Data Selection

There were a total of 116 queries performed by 32 users. However, not all queries were suitable for meaningful analysis. The 116 queries were filtered down into 25 queries with the following criteria:

1. At least 5 hashtags and 5 keywords in the initial results set. (106 queries)
   This filtered out queries that did not have enough initial results. For example, the query *Kanat Bekt* only returned one result.
2. Each of the query expansion methods had at least 5 results. (42 queries)
   This filtered out queries that did not have enough results for one or more query expansion methods. Unfortunately, this filtered out some good queries such as those that had more than 5 results for all methods except one or two. Although it would have been helpful to include some of these queries, we did not include them in our analysis as this would have introduced more variables in our computations.
3. Each query had 100% feedback from the user. (28 queries)
   This filtered out queries that did not have user feedback for all of their results. The reason for this is that we did not want to "guess" the scores of the results the user left unrated. This reduced ambiguity and allowed us to achieve more objective results.
4. Maximum of three queries per user. (25 queries)
   Although the user did not know which results belonged to which methods, this filter reduced bias that the user might have had towards a particular method.

[1] `https://www.facebook.com/groups/datahackers/`

q	M0	M1	M2	M3	M4	M5
0	Seahawks	Seahawks #seahawks	Seahawks seattle	Seahawks #seahawks seattle	Seahawks #seahawks #nflcombine	Seahawks seattle marshawn
1	net neutrality	net neutrality #nonetneutrality	net neutrality fcc	net neutrality #nonetneutrality fcc	net neutrality #nonetneutrality #netneutrality	net neutrality fcc fccs
2	macbook	macbook #applewatch	macbook apple	macbook #applewatch apple	macbook #applewatch #applelive	macbook apple event
3	apple watch	apple watch #apple	apple watch economist	apple watch #apple economist	apple watch #apple #applewatch	apple watch economist iphone
4	Yeezy 750 boost	Yeezy 750 boost #adidas	Yeezy 750 boost adidas	Yeezy 750 boost #adidas adidas	Yeezy 750 boost #adidas #kanyewest	Yeezy 750 boost adidas size
5	LCS	LCS #lcs	LCS elements	LCS #lcs elements	LCS #lcs #lcsbigplays	LCS elements lolesports
6	ferguson	ferguson #ferguson	ferguson police	ferguson #ferguson police	ferguson #ferguson #darrenwilson	ferguson police justice
7	boston bombing	boston bombing #news	boston bombing marathon	boston bombing #news marathon	boston bombing #news #bostonmarathon	boston bombing marathon trial
8	house of cards	house of cards #houseofcards	house of cards houseofcards	house of cards #houseofcards houseofcards	house of cards #houseofcards #hoc	house of cards houseofcards netflix
9	samsung	samsung #business	samsung galaxy	samsung #business galaxy	samsung #business #cbc	samsung galaxy samsungs
10	Cardinals Baseball	Cardinals Baseball #stlcards	Cardinals Baseball stlcards	Cardinals Baseball #stlcards stlcards	Cardinals Baseball #stlcards #baseball	Cardinals Baseball stlcards spring
11	amazon gift cards	amazon gift cards #amazon	amazon gift cards itunes	amazon gift cards #amazon itunes	amazon gift cards #amazon #itunes	amazon gift cards itunes xbox
12	swagbucks	swagbucks #swagbucks	swagbucks swag	swagbucks #swagbucks swag	swagbucks #swagbucks #swagcode	swagbucks swag bucks
13	Roald Dahl	Roald Dahl #worldbookday	Roald Dahl worldbookday	Roald Dahl #worldbookday worldbookday	Roald Dahl #worldbookday #roalddahl	Roald Dahl worldbookday roalddahl
14	iphone	iphone #spideyonpix	iphone apple	iphone #spideyonpix apple	iphone #spideyonpix #iphone	iphone apple spideyonpix
15	Dallas Cowboys	Dallas Cowboys #cowboys	Dallas Cowboys nfl	Dallas Cowboys #cowboys nfl	Dallas Cowboys #cowboys #nfl	Dallas Cowboys nfl dez
16	snow-storm	snowstorm #snowstorm	snowstorm skids	snowstorm #snowstorm skids	snowstorm #snowstorm #snow	snowstorm skids snow
17	icc cwc	icc cwc #cwc15	icc cwc cwc15	icc cwc #cwc15 cwc15	icc cwc #cwc15 #pakvuae	icc cwc cwc15 pakvuae
18	ac milan	ac milan #milan	ac milan liverpool	ac milan #milan liverpool	ac milan #milan #deals	ac milan liverpool el
19	peyton manning	peyton manning #broncos	peyton manning broncos	peyton manning #broncos broncos	peyton manning #broncos #peytonmanning	peyton manning broncos nfl
20	Marijuana	Marijuana #marijuana	Marijuana legal	Marijuana #marijuana legal	Marijuana #marijuana #cannabis	Marijuana legal dc
21	net neutrality	net neutrality #netneutrality	net neutrality fcc	net neutrality #netneutrality fcc	net neutrality #netneutrality #internet	net neutrality fcc netneutrality
22	samsung	samsung #galaxys6edge	samsung galaxy	samsung #galaxys6edge galaxy	samsung #galaxys6edge #galaxys6	samsung galaxy s6
23	lean engi-neering	lean engineering #jobs	lean engineering manufacturing	lean engineering #jobs manufacturing	lean engineering #jobs #engineering	lean engineering manufacturing manager
24	Arkansas Basketball	Arkansas Basketball #arkansas	Arkansas Basketball kentucky	Arkansas Basketball #arkansas kentucky	Arkansas Basketball #arkansas #basketball	Arkansas Basketball kentucky sec

Figure 4.1: All qualifying queries. Method 6 is not included because it is just M1 concatenated with M2 with an *OR* operand.

4.2 Results & Discussion

The average weighted precision (AP'@5) scores were calculated for each method of each query.

q	M0	M1	M2	M3	M4	M5	M6
0	1.000	1.000	1.000	0.565	0.980	1.000	1.000
1	1.000	0.693	1.000	1.000	0.980	1.000	0.980
2	1.000	0.500	1.000	0.500	0.327	0.743	1.000
3	0.743	0.375	1.000	1.000	0.643	1.000	1.000
4	0.837	0.268	0.837	0.268	0.000	0.447	0.000
5	1.000	1.000	1.000	1.000	0.872	1.000	1.000
6	0.663	0.157	0.613	0.157	0.000	0.575	0.447
7	0.000	0.085	0.000	0.157	0.000	0.000	0.000
8	0.040	0.128	0.257	0.128	0.613	0.000	0.663
9	0.693	0.293	0.693	0.293	0.293	1.000	1.000
10	0.843	0.653	0.653	0.653	1.000	1.000	1.000
11	0.782	0.543	1.000	1.000	1.000	1.000	1.000
12	1.000	1.000	1.000	1.000	1.000	1.000	0.128
13	0.318	0.628	0.628	0.628	0.698	0.698	0.693
14	0.872	0.420	0.543	0.227	0.803	0.227	0.515
15	1.000	1.000	1.000	0.857	0.857	0.922	1.000
16	0.980	0.852	1.000	1.000	0.922	0.543	1.000
17	0.728	0.857	0.807	0.857	0.520	0.520	0.500
18	0.960	0.663	0.752	0.535	0.605	0.935	0.520
19	1.000	1.000	1.000	1.000	0.387	1.000	1.000
20	1.000	0.793	1.000	1.000	0.852	0.565	0.922
21	0.922	0.955	0.832	1.000	0.733	1.000	0.603
22	0.623	0.662	0.513	0.563	0.615	0.585	0.453
23	1.000	1.000	1.000	0.955	0.980	1.000	1.000
24	0.872	0.578	0.872	0.578	0.500	0.578	0.877
MAP′	**0.795**	0.644	**0.800**	0.677	0.647	0.734	0.732

Figure 4.2: Average weighted precision scores for the 25 queries from section 4.1.

Figure 4.2 suggests that mean weighted average precision scores for method 0 (baseline) and method 2 were relatively high. We can safely conclude that, in general, users find the top-5 results returned from Twitter Search interesting. Method 2, which adds the top keyword as an additional query term, also tends retrieve mostly interesting top-5 results. Since the MAP' was already so high for the baseline method, it is challenging to achieve a score that is significantly higher than that.

Method 0 yielded 100% AP' for 9 out of 25 queries. For methods 5 and 6, it was 11 out of 25 queries, while for method 2, the number was 12 out 25 queries. However, method 0 had only 3 queries for which the average weighted precision was less than or equal to 60%. For methods 5 and 6, it was 10 and 8 queries, respectively, while for method 2, the number was 4 queries.

A few factors affect the average weighted precision scores such as the rank of each result and the score the user gave for the result. Since we compared our results against the baseline method, we looked into how method 0 results overlap with the results methods 1-6.

	M0	M1	M2	M3	M4	M5	M6
Mean	1.000	0.792	0.560	0.864	0.936	0.824	0.888

Figure 4.3: Percentage of new results compared to method 0 results.

Figure 4.3 suggests method 2 overall has 56.0% new results compared to method 0, while method 4 has 93.6% new results. This might explain why method 2's MAP' was so high and similar to the MAP' of method 0. Method 4 discovered the most new content the user might not have otherwise discovered. However, the MAP' score of method 4 was relatively low.

Since the average weighted precision score takes rank orders into consideration, the non-overlapping results might have suffered from the orderings of the overlapping results. Therefore, the MAP' scores for only non-overlapping results were calculated for methods 1-6. The MAP' score of method 0 is the same as in Figure 4.2.

	M0	M1	M2	M3	M4	M5	M6
MAP′	0.795	0.666	0.805	0.680	0.647	0.748	0.721

Figure 4.4: Mean average weighted precision scores for non-overlapping results only.

Interestingly, the MAP' scores for all the methods except for methods 4 and 6 improved when overlapping results (with method 0) were removed. Method 2, which had MAP' of 80.0% with 56.0% new results, outperformed the baseline method when the overlapping results were excluded, bumping the MAP' score to 80.5% with 100.0% new results. This suggests that method 2 discovered interesting results the user might not have otherwise discovered using Twitter Search. There were no interesting patterns in the queries for us to categorize them into meaningful groups. There were no evident factors that explain the increase or decrease in average precision scores, with or without the non-overlapping results.

5 Conclusion

Automatic query expansion has proved to perform well in modern Web search engines. We explored a few options on how to apply the same concept on a microblogging environment. This thesis focused on building a Web service that provides an improved search experience for the microblogging platform Twitter. The experiment shows that performance measure without automatic query expansion is as good as performance measure with automatic query expansion for a small number of results.

We found out that retrieval efficiency was high for both expanded queries and non-expanded queries. The pseudo-relevance feedback method which expanded the original query with the most occurring keyword slightly outperformed the baseline method (non-expanded query).

We have also confirmed that the same method had the least percentage of originality compared to the baseline method. The pseudo-relevance feedback method which expanded the original query with the two most occurring hashtags yielded the highest percentage of originality compared to the baseline method. Additionally, excluding those results that appeared in the baseline method results slightly improved retrieval efficiency for 4 out of the 6 automatic query expansion methods.

Since removing the overlapping results caused slight improvement in retrieval effectiveness, future work could be to design a novel pseudo-relevance feedback that would exclude results that appear in the baseline method's results. Furthermore, combining the results of the six methods could lead to improved retrieval results. In that case, the results that appear in multiple methods would have a higher rank.

References

[1] Jinxi Xu and W. Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval* (SIGIR '96). ACM, New York, NY, USA, 4-11. DOI=10.1145/243199.243202 http://doi.acm.org/10.1145/243199.243202

[2] Claudio Carpineto and Giovanni Romano. 2012. A Survey of Automatic Query Expansion in Information Retrieval. *ACM Comput. Surv* 44, 1, Article 1 (January 2012), 50 pages. DOI=10.1145/2071389.2071390 http://doi.acm.org/10.1145/2071389.2071390

[3] White, Ryen W., Ruthven, Ian and Jose, Joemon M. The Use of Implicit Evidence for Relevance Feedback in Web Retrieval in Crestani, Fabio and Girolami, Mark and van Rijsbergen, Cornelis Joost. ed. *Advances in Information Retrieval*, Springer Berlin Heidelberg, 2002, 93-109. DOI=10.1007/3-540-45886-7_7 http://dx.doi.org/10.1007/3-540-45886-7_7

[4] Jaime Teevan, Daniel Ramage, and Merredith Ringel Morris. 2011. #TwitterSearch: a comparison of microblog search and web search. In *Proceedings of the fourth ACM international conference on Web search and data mining* (WSDM '11). ACM, New York, NY, USA, 35-44. DOI=10.1145/1935826.1935842 http://doi.acm.org/10.1145/1935826.1935842

[5] Cher Han Lau, Yuefeng Li, and Dian Tjondronegoro. 2011. Microblog Retrieval Using Topical Features and Query Expansion. In *E. M. Voorhees & L. P. Buckland (eds)* (TREC). National Institute of Standards and Technology (NIST).

[6] Miles Efron. 2010. Hashtag retrieval in a microblogging environment. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (SIGIR '10). ACM, New York, NY, USA, 787-788. DOI=10.1145/1835449.1835616 http://doi.acm.org/10.1145/1835449.1835616

[7] Ayan Bandyopadhyay, Prasenjit Majumder, and Mandar Mitra, 2012. Query expansion for microblog retrieval. *Int. J. Web Science*, Vol. 1, No. 4, pp.368-380.

[8] Mandar Mitra, Amit Singhal, and Chris Buckley. 1998. Improving automatic query expansion. *In Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval* (SIGIR '98). ACM, New York, NY, USA, 206-214. DOI=10.1145/290941.290995 http://doi.acm.org/10.1145/290941.290995