Fan Jiang. Exploring Pseudo-Relevance Feedback for Microblog Search. A Master's paper for M.S. in I.S. degree. April, 2014. 48 pages. Advisor: Jaime Arguello.

This study explored the effectiveness of a classical information retrieval (IR) approach, pseudo-relevance feedback (PRF), on improving the performance of microblog search. Factors including number of PRF iterations, term selection strategy, term weighting scheme and use of user-generated metadata were examined in order to shed light on their influence on the effectiveness of the studied approach in an environment of microblog search. An IR system implementing the studied approach was developed for experiments and experiments were conducted on an English microblog corpus composed of Twitter's microblogs, known as tweets. Search performance was evaluated using precision at thirty (P@30), mean average precision (MAP) and normalized discounted cumulative gain at thirty (NDCG@30).

As a result, it was found that pseudo-relevance feedback can significantly improve performance of microblog search. Meanwhile, it was also revealed that expanding queries with hashtags is detrimental to the search performance. Besides, it was also identified that term weighting can contribute to search performance.

Headings:

Information Retrieval

Pseudo-Relevance Feedback

Query Expansion

Microblog Search

EXPLORING PSEUDO-RELEVANCE FEEDBACK FOR MICROBLOG SEARCH

by
Fan Jiang

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

April, 2014

Approved by:

_____

Jaime Arguello

Table of Contents

Introduction **2**

Literature Review **3**

Methodology **9**

Results **21**

Discussion **27**

Future Work **31**

Conclusion **33**

References **34**

Appendix I – Implementation of Term Collector **37**

Appendix II – Implementation of Analysis Component **40**

Introduction

Microblogging is one of the most widely used forms of social media. According to the latest statistics on Twitter, the world's most popular microblogging platform, over 58 million tweets are posted and over 2.1 billion searches are performed each day in average (Cook, 2013). Sharing information, seeking information and building social connections are major features frequently used (Java et al. 2007). Among these features, an increasing need for seeking information from microblogs was discovered as the volume of microblog data scales rapidly and the excellence microblogging services, especially Twitter, have shown in reporting breaking news. However, identifying preferred information from microblogs is challenging due to its succinct expression, informal writing and potential of containing noise (Lau et al. 2012). Therefore, this study attempted to tackle these problems by applying pseudo-relevance feedback to microblog search, particularly against a corpus of tweets. Meanwhile, the study also looked into a number of parameters in the studied approach, which can significantly influence its effectiveness.

The study employed P@30, MAP and NDCG@30 as evaluation metrics, since it was assumed that microblog search users pay more attention to precision than recall. A baseline run with raw queries has been conducted for comparison. Potential limitations of this study are discussed in the last section of this paper.

Literature Review

Overview of Microblog and Microblog Search

A microblog is defined as “a weblog that is restricted to 140 characters per post but is enhanced with social networking facilities”(McFedries, 2007). An increasing number of social networking service providers, such as Facebook, Tumblr, etc., have integrated microblogging service into their platform, whereas Twitter is still the most visible microblogging platform in the world (Efron, 2011). Twitter was launched in 2006 and growing rapidly since it won South By South by South West (SXSW) conference Web Awards in 2007 (Java et al. 2007). It has around a billion registered users and over 300 million tweets published till October 2013 (Smith, 2014). Meanwhile, it was found that information seeker is one of main user categories on Twitter (Java et al. 2007). Thus, retrieving relevant information from such a large amount of data that grows at a extremely fast pace, is becoming an intriguing topic in IR area. Fortunately, Twitter makes it possible to do research on the data it owns by exporting Twitter streaming application programming interface (API), via which people can access historical tweet data. The huge volume of data and the easy access to the data make Twitter a dominant source of data for researches on microblog search. In this study, the researcher also adopted an experimental corpus developed with Twitter’s data.

In contrast to most IR areas, microblog search is a new area to be studied. Efron (2011) gave an overview of microblog search, identifying an array of central problems and proposing methods to address these problems. The author recognized two types of microblog search, *asking* and *retrieving*. *Asking* refers to posting questions as microblogs and expecting answers from the public or the author's social network, while retrieving refers to searching for relevant information from existing microblogs. The latter is more like traditional ad hoc IR and would be the focus of this study. The author also pointed out the different definitions of relevance between microblog search and other types of searches. The author also suggested that an array of issues other than text matching, such as temporality, topicality, authority and locality, play important roles in defining relevance in microblog search. However, text matching is still an influential dimension in microblog search and performance improvement in this dimension will definitely contribute to the overall search performance against microblogs. As a microblog is associated with a limited length of text, pseudo-relevance feedback can be one of the best approaches for microblog search, since it can refine queries with terms likely to be relevant in order to increase the probability of matching the text of relevant microblogs. This study would mainly focus on improving search performance in the scope of text matching and slightly dive in the influence of topicality in microblog search.

As microblog search is nascent, comparisons between microblog search and other existing IR area have been made in an effort to find commonalities between these areas and attempt to solve new problems with existing methods. Teevan et al. (2011) compared microblog search to the Web search. They discovered that people resorted to microblog search mainly for temporarily relevant information, whereas they were inclined to learn

about a topic when conducting the Web search. The difference in motivation led to entirely distinct user behaviors observed in microblog search and the Web search. Microblog searchers tended to formulate shorter, more popular queries and repeat same queries in order to monitor updates about a particular event. The Web searcher users were used to refining a query through a search to gain knowledge about a topic in depth to some extent. It indicated that search engines should adopt automatic approaches to infer users' intents and improve performance of microblog search, as users were reluctant to interacting with search engines through microblog searches. Meanwhile, it also suggested that metrics that measure precision could better simulate motivation of microblog search users than those that measure recall.

Microblogging services, such as Twitter, also provide a number of features that are potentially conducive to improving microblog search performance. Teevan et al. (2011) and Efron (2011) recognized "@" and "#" symbols as the most frequently used features in Twitter. "@" is used to refer to a user name, while "#", also known as a hashtag, is used to self-tag a tweet (Teevan et al., 2011). The "@" symbol is usually followed by an existing user name on Twitter to mention a user in a tweet. Although it is commonly seen in the body of tweets, it is less frequently included into queries against microblogs (Teevan et al., 2011). The "#" symbol is usually followed by a stream of characters in sequence describing a topic or an event. Hashtags were used to locate a clustering of tweets associated with the hashtags and supposed beneficial to microblog search. Efron (2010) looked into hashtags retrieval and attempted to improve microblog search performance by using query expansion with selected hashtags. The author harvested hashtags from the initial search results and applied a language model to select meaningful

hashtags for query expansion. As was reported in the poster, the method effectively improves performance of the search against a corpus containing 3,414,330 tweets. However, Janes (2013) reported that query expansion using hashtags harvested from search results was detrimental to the performance of ad hoc microblog search. In this study, the researcher would also look at effect of query refinement with hashtags on the performance of microblog search.

Regarding evaluation of microblog search, Efron (2010) suggested that Cranfield model for evaluating traditional TREC ad hoc IR could also be contributive to microblog search. The Cranfield model was a dominant approach to evaluate traditional IR systems and proved effective in evaluating innovative approaches (Voorhees, 2007). The model includes test collections consists of “a collection of test documents, a collection of test queries; and a collection of relevance assessments”(Borlund, 2003), and measurement of precision and recall. Because of the differences between microblog search and traditional ad hoc IR, it is doubted whether results of the Cranfield-style experiments could reflect the actual search effectiveness or not. Efron(2011) argued that the Cranfield model would still work for microblog search evaluations with several issues addressed. In spite of factors in the traditional IR context, he emphasized that a number of factors should also be taken into account, such as temporality, locality, authority, recency, etc. In this study, the Cranfield model would be employed for performance evaluation.

Pseudo-Relevance Feedback

Pseudo-relevance feedback is a classical method to improve IR performance. It automatically collects useful information from the search result and refines queries without involving any human interactions with the IR system. Pseudo-relevance feedback assumes the top *k* (*k* is an arbitrary number) documents in the search result are relevant and extracts useful information from these documents to refine queries. It was proved effective in improving performance of TREC ad hoc tasks (Manning et al., pp. 187). It was also successfully implemented in other IR environments such as the Web search (Yu et al., 2003) and Multimedia search (Yan et al., 2003).

Pseudo-relevance feedback was also implemented for microblog search. Lau et al. (2011) applied pseudo-relevance feedback to microblog search, capturing both term-based and pattern-based features to refine queries. The approach achieved perfect MAP scores of 1.00 for some queries but performed terribly in some cases. Bandyopadhyay et al. (2012) proposed a novel way to use pseudo-relevance feedback in microblog search. The proposed approach collected terms from the search result and searched for more terms from the Web with the collected terms for query expansion. Unfortunately, the approach produced bad results. However, further work might be done to improve the effectiveness of the proposed approach.

Massoudi, et al. (2011) proposed an enhanced pseudo-relevance feedback approach. In this approach, they introduced textual and microblog-specific quality indicators to select terms for query expansion and construct new queries. The textual quality indicators

included emoticons, post length, shouting, capitalization and the existence of hyperlinks. Microblog-specific quality indicators were acquired from these elements. The researchers mainly focused on reposts, followers and recency. It is worth noting that the researchers took recency into account when selecting terms for query expansion. They developed an algorithm to give higher scores to terms temporally closer to a given query and discard terms in tweets posted after the time when the query was issued. This algorithm better simulated user behavior in searching than those that treat a microblog dataset as a static corpus, as people can only query against posted microblogs and should know nothing about microblogs posted in the future. The researchers developed an experimental corpus composed by 110,038,694 tweets and created a query set with 30 topics. The proposed approach brought about a 92.8% improvement in MAP score by integrating both textual and microblog-specific quality indicators into pseudo-relevance feedback, showing significant competency in improving performance of microblog search

Methodology

Experiment Design

The experiments were designed as traditional ad hoc IR experiments, in the effort to explore the effectiveness of pseudo-relevance feedback for microblog search in various settings. Basically, the researcher investigated five parameters and their influence on the performance of microblog search. The researcher tuned these parameters and compared the resulting metric scores to the baseline run's, to observe the influence of each parameter. The parameters investigated in this study were number of PRF iterations, genre of feedback terms, number of feedback terms, number of feedback tweets and term weight distribution. There were three genres of feedback terms available in this study: selected terms, hashtags and selected terms coupled with hashtags. The selected terms refer to terms (excluding hashtags) selected from retrieved tweets at the top of search result with highest scores, whereas hashtags are hashtags selected from these tweets. A feedback tweet is defined as a retrieved tweets at the top of the search result from which feedback terms are selected.

Query terms were weighted using a simple term weighting scheme in this study. It only weighted terms in the initial queries and selected terms for query expansion. Hashtags were not weighted in this study, since hashtags were not scored and thus there was no way to differentiate hashtags. The maximum term weight value was 1 and there was no

lower bound for the value. The value of term weight ranged from 0 to 1. Terms in the initial query were assigned a perfect weight of 1, as they were assumed to be the most relevant to the topic. Term weight decreased at a specific step as the term's rank dropped. The step value was specified by the researcher and ranged from 0 to 1. It was used to degrade the contribution of terms less likely to be relevant to the original query. The formula of the term weighting scheme was shown in Figure 1.

$$w(i) = 1 - \frac{step * i}{N}$$

Figure 1 Term Weighting Formula

In this formula, *w(i)* is the weight to be assigned to the term at rank *i. step* is the step size of term-weighting decay. It can be an arbitrary value between 0 and 1 specified by the researcher. Because the value of term weight must be positive as required in the implementation, the subtrahend was normalized by the total of selected terms, which was denoted by *N* in this formula.

As can be seen in the formula, the step size brings about two effects. It controls the decay associated with each feedback term as a function of its rank and the weight associated with feedback terms compared to the original query terms, which always acquire a weight value of one. For instance, if there are feedback terms ranked from 1 to 10, the term weight will degrade in a manner shown in Figure 3.

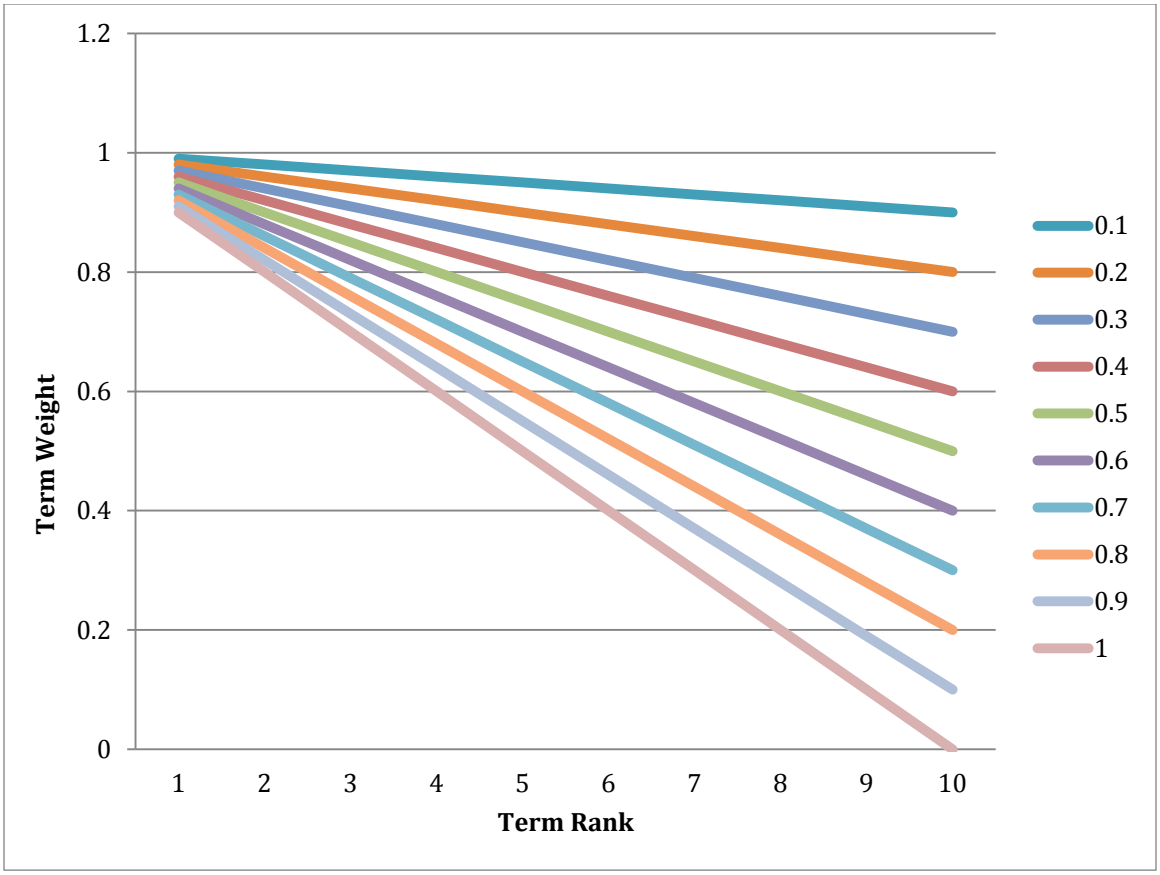

Figure 2 Term-weighting decay variations with different step sizes

As is displayed in Figure 2, the term weight decreases more rapidly when the step size is larger. Meanwhile, the difference of term weight between the feedback terms and the original query terms is larger when the step size is larger. Therefore, the original query terms will put much effect on a search when the step size is large, whereas feedback terms, especially low-ranked feedback terms, will get more chances to influence a search when the step size is small.

The study introduced vector space model (VSM) as retrieve model and term frequency-inverse document frequency (TF-IDF) as scoring model. Term selection took advantage

of TF-IDF scores to rank and select terms from feedback tweets for query expansion. All the hashtags within feedback tweets were harvested for query expansion with hashtags or selected terms coupled with hashtags. No language model was used to recognize and filter out non-relevant hashtags.

A corpus and an experimental IR system were developed for experiments. All the experiments were running on the researcher's laptop with 2 cores and 4 gigabytes RAM[1].

Experimental Corpus

In this study, experiments were carried out in a manner of TREC ad hoc tasks. The experiments were running on an English corpus of tweets derived from TREC Microblog Track 2011. The original corpus was developed by collecting tweets over a 2-week period from January 24th 2011 to February 8th using Twitter Streaming API, containing approximately 16 million tweets posted in a variety of languages. Each tweet instance is stored in XML format with fields including document identifier, timestamp, author and text. As the study only focused on English tweets, the researcher extracted an English corpus from the raw corpus using Language Detection Library for Java, which can achieve the best accuracy (99.7%) for English among the most famous language detection libraries (Mccandless, 2012). The extracted English corpus contains 3,334,449 tweets in total.

[1] RAM: random access memory

TREC Microblog Track 2011 also provided a test collection associated with the experimental corpus. The test collection consists of a topic file that contains 50 real user queries and a *qrels* file that stores human relevance judgments of each tweet within the corpus to each query within the topic file. Each user query is identified by a topic number, a title that contains query text, a query timestamp and query tweet timestamp defined as the timestamp of the tweet within the corpus that is chronologically nearest to the query. As this study required only manipulation of text but barely investigated some features of tweet such as recency and authority, only content in the title field within each topic was used for formulating queries.

Experimental System

To perform experiments, an experimental IR system was developed in Java using an open source search engine library Apache Lucene, which is one of the most widespread search engine libraries. Lucene has efficient implementations of indexing and searching techniques, and it provides simple APIs to the developer to build search engine quickly. Besides, Lucene makes most search engine components pluggable and has implemented a number of standard components, so that developers can either plug their own algorithm implementations into Lucene or make use of existing implementations to build search engines. A latest stable version of Apache Lucene 4.6.1 was used in this study.

Lucene includes a variety of scoring models, among which the default one, TF-IDF, was used in this study. Lucene has revised traditional TF-IDF scoring function by introducing a set of factors into the function. The scoring function is shown in Figure 3.

$$score(q,d) = coord(q,d) * queryNorm(q) * \sum_{t} (tf(t) * idf(t^2) * boost(t) * norm(t,d))$$

Figure 3 Lucene's TF-IDF scoring function (Hatcher et al. pp. 86)

In this function, *score(q,d)* represents the score of document *d* for query *q*. *coord(q,d)* is a score factor based on how many of terms in the query occur in the document *d*. This factor favors documents with more query terms. *queryNorm(q)* is a normalizing factor that makes scores between different queries comparable. The factor does not influence document ranking for a specific query as it only correlates to query. *boost(t)* is a score factor based on the user-defined relevancy of a specific term. It is 1 for all terms by default and can be set by programmer in queries. Documents containing terms with higher boost value tend to get a higher score, and *vice versa.* In this study, *boost(t)* is used to implement term weighting scheme as will be described below. *norm(t, d)* correlates to document length and field boost. Documents with shorter length tend to be scored higher. Regarding field boost, no fields are boosted in this study so it will not influence the score.

The system is made of five main components: the indexing component, the search component, the evaluation component, the tracking component and the analysis component. The system's workflow is shown in Figure 4.

The indexing component is responsible for extracting content from the experimental corpus in XML format, detecting tweets in a specific tweet, extracting hashtags, tokenizing these tweets using a specific tokenizer and indexing them. In this particular study, the indexing component processed English tweets using an English analyzer

included with Apache Lucene. The English analyzer extracts words from the text and processes each word with normalization, stemming, stop words filtering and possessives removal. The component adopts the default stop word collection provided by Apache Lucene 4.6.1 and keeps term vectors along with each term in the index to favor term selection algorithm during query expansion. Since a part of this study would investigate the influence of use of user-generated metadata, particularly hashtags, on the effectiveness of pseudo-relevance feedback, the indexing component applies a regular expression script towards the textual part of each tweet to identify, extract and store hashtags at indexing. According to the assumption of TREC ad hoc tasks of Microblog Track, indexes should be created for each query, containing tweets posted earlier than current query, since it is assumed that users can only query for posted tweets. However, only one index was created for experiments in this study due to time and resource constraint. The index was stored in the researcher's laptop with a size of 1,005 megabytes in total.

The search component is a composite of four major subcomponents: the topic reader, the query maker, the term collector and the experiment. The topic reader simply extracts topic number and query content from the topic file in XML format and feeds the queries to the query maker.

The query maker acquires terms for query expansion from the term collector and reads experiment setup from the experiment component, so as to formulate and launch desired queries against the index. Particularly, the query maker takes advantage of an important feature of Apache Lucene known as *boost* to implement the term weighting scheme.

Typically, a *boost* is a floating point number assigned to each term in a query. Terms with higher *boost* values are supposed more relevant than those with lower *boost* values. In this study, the query maker would compute term weight using the formula shown in Figure 1 and assigned the weight as a boost to each weighted term directly.

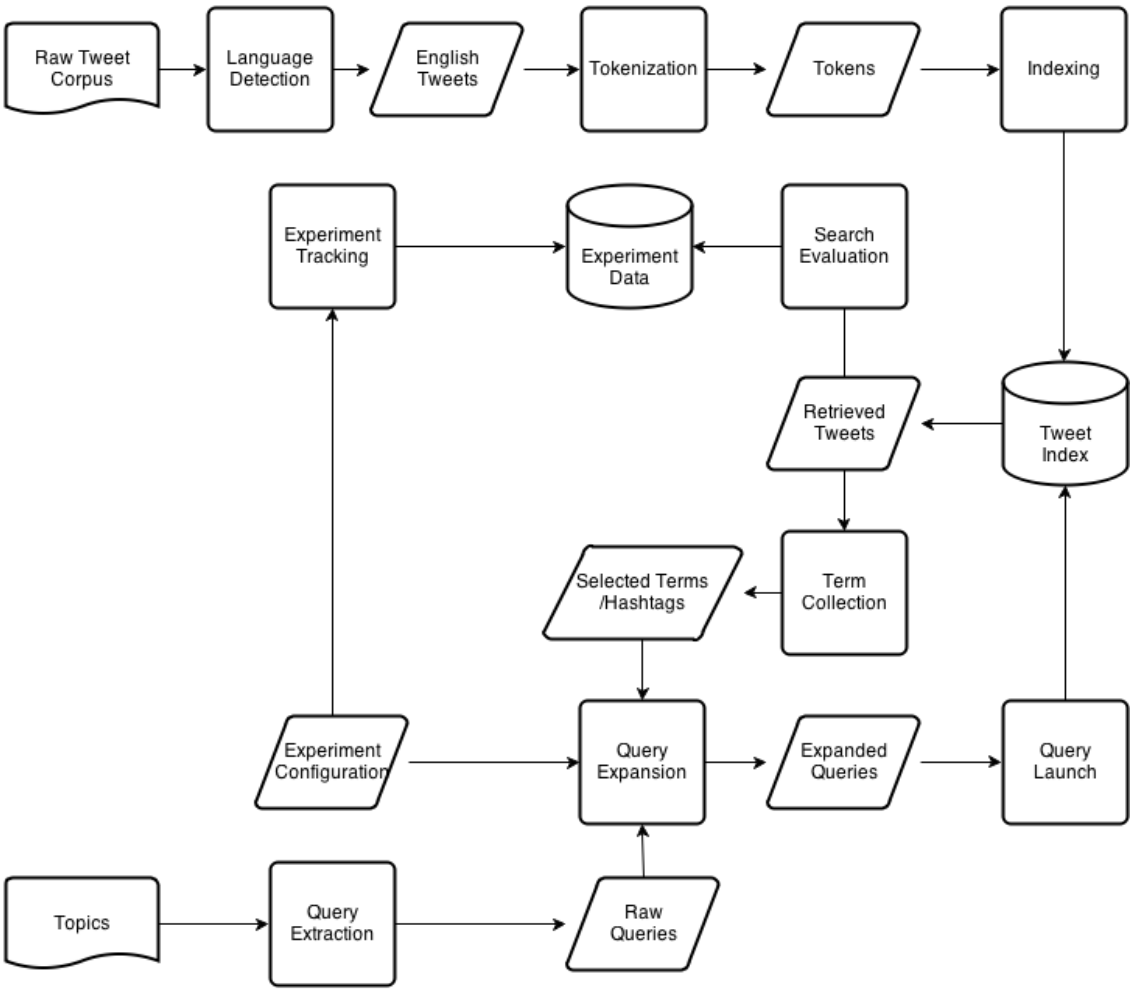

Figure 4 Experimental IR system workflow

The term collector is an implementation of PRF and term selection algorithm. It selects top *k* (*k* is a number specified in the query configuration) retrieved tweets, ranks terms

within these tweets by their TF-IDF scores and returns a specific number of terms at the top to the query maker for query expansion. A term's TF is calculated by adding up its total occurrences in the feedback tweets. A term's IDF is calculated by dividing the total number of feedback tweets by the number of tweets with the term occurring and then computing logarithm of the quotient. It can also return hashtags in the top *k* retrieved tweets to the query maker. The code of the implementation is shown in Appendix I.

The experiment subcomponent integrates all the subcomponents to make the search component work as a whole. It requires the researcher setting all the studied parameters as a part of experiment setup and passes them to proper subcomponents, making experiments easy to repeat.

The core of the evaluation component is a tool specially developed for evaluating TREC ad hoc tasks, called *trec_eval*. The tool requires a *qrels* file and a *result* file to calculate scores of all the metrics overall or for an individual query. Current version of *trec_eval* employed in this system is 9.0.The *result* file will be generated by the evaluation component as related data is collected from the search component at completion of each search.

The tracking component is in charge of keeping track of experiment statistics. It monitors each experiment and collects relevant data that describes the experiment. Instead of persisting experiment data into a relational database (RDB), the tracking component also stores and indexes the data using Apache Lucene. The workaround not only spares efforts to design an RDB schema, but also lends certain flexibility to the researcher to

manipulate the data, as Apache Lucene works as a NoSQL database, which is not compliant with ACID model[2].

The analysis component is used to analyze experiment data and visualize the analysis results. The data visualization functionality was developed using Java AWT library and an open-source graphing library known as GRAL. The latest version of GRAL 0.1 was included in this component. Although this component is not a typical part of an IR system, it is indispensable in this study as it contributes much to the completion of data analysis in this study. The analysis component reads experiment data, calculates improvement of a specific metric in every single experiment and plots a line chart to show variation of metric improvement as the value of a particular parameter changes. There are always three lines, which stand for query expansion with only selected terms in top retrieved tweets, with only hashtags and with both selected terms and hashtags respectively, in each line chart for comparison. The main code of the analysis component is listed in Appendix II.

Data Collection

The researcher conducted 2056 experiments and collected the data in a size of 440 kilobytes in total. Details about the data collection are listed in Table 1.

[2] ACID model refers to "Atomicity, Consistency, Isolation and Durability", which are four properties that guarantee a reliable database transaction.

Experiment Group No.	Number of PRF Iterations	Number of Feedback Terms	Number of Feedback Tweets	Term-Weighting Decay Step Size	Genre of Feedback Terms
1	0	0	0	0	N/A
2	1 ~ 6 (by 1)	5	5	0.1	selected terms
3	1	5 ~ 25 (by 5)	5 ~ 25 (by 5)	0.0 ~ 1.0 (by 0.1)	all
4	1	30 ~ 100 (by 5)	30 ~ 100 (by 5)	0.1	all
5	1	5 ~ 25 (by 5)	100 ~ 500 (by 100)	0.1	all
6	1	5 ~ 25 (by 5)	1000	0.1	all

Table 1 Data collection details

The experiment in Group 1 is the baseline run for comparison. Experiments in Group 2 focused on the influence of PRF iterations on the search performance. Experiments in Group 3 mainly contributed to the study on the impact of term weight distribution. Experiments in Group 3 ~ 6 concentrated on the effect of variation of feedback term number or feedback tweet number on effectiveness of the studied approach. Except experiments in Group 2, all other experiments performed only one-round PRF iteration because one-round PRF iteration yielded best performance, which will be explained in the paragraphs below.

Performance Evaluation

Each search was evaluated using all metrics supported in *trec_eval*. However, this study only examined scores of P@30, MAP and NDCG@30 since they reflected search preferences of most microblog searchers. P@30 measures the fraction of relevant documents in the top 30 retrieved documents. MAP measures average precision averaged across all testing queries. MAP score is potentially higher if ranks of relevant documents are higher. NDCG is a more fine-grain and sensitive measurement. Instead of binary relevance, NDCG assumes there are multiple levels of relevances. NDCG score will decrease rapidly as the rank of relevant documents drop. In this study, the level of relevances ranged from -2 (totally non-relevant) to 2 (highly relevant) and NDCG@30 was investigated.

The study compared the metric scores of the baseline run and each run applied pseudo-relevance feedback. The score differences were calculated to check whether pseudo-relevance feedback improved the performance with specific parameters

Results

The best performance improvement evaluated with each selected metric was observed and recorded in this study. As a result, pseudo-relevance feedback brought about 27.6% improvement in P@30 score, 46.7% in MAP score and 34.5% in NDCG@30 score at best for microblog search. The result indicated that pseudo-relevance feedback could improve performance of microblog search significantly in certain settings.

The researcher analyzed the variation of improvement as a specific parameter changed. The results were displayed in line charts generated by the analysis component of the experimental system.

The researcher ran experiments performing searches with increasing PRF iterations and varying sets of other parameters as is shown in Figure 5. The analysis revealed that one-round PRF iteration resulted in the maximum performance improvement but more PRF iterations led to decreasing performance improvement or even declining performance. Such variation remained when any other parameter changed.

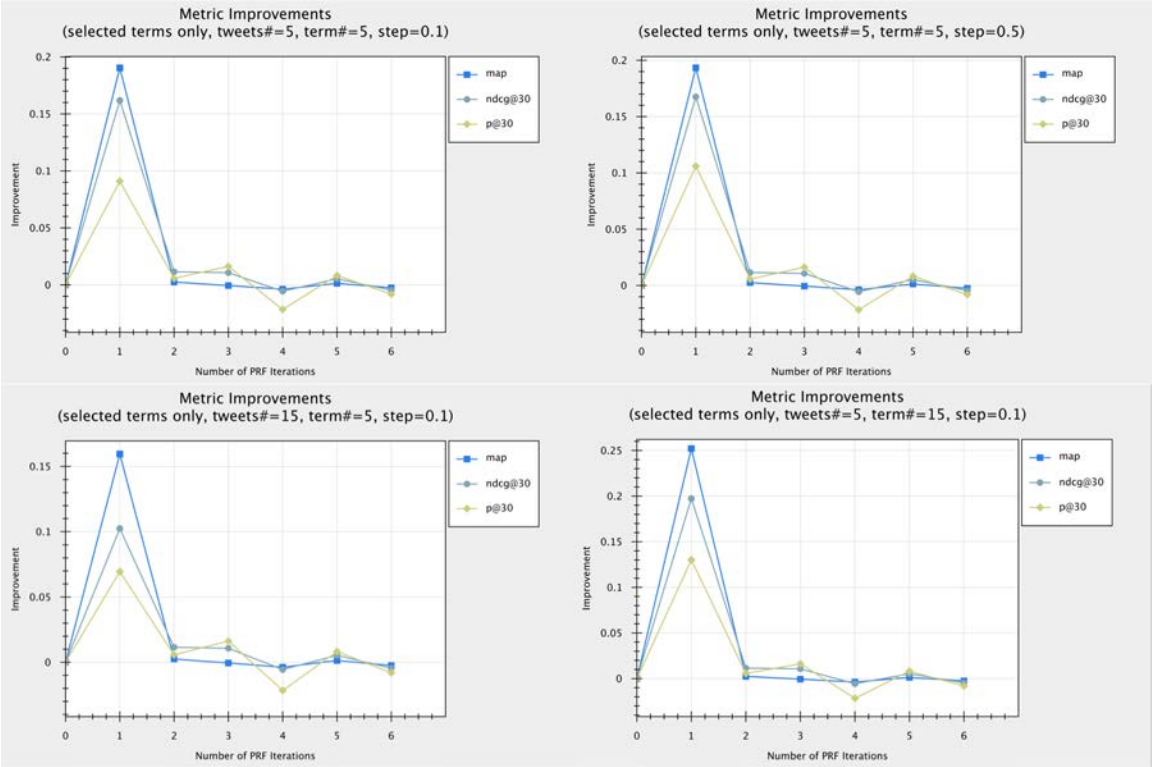

Figure 5 Performance improvement variation on increasing PRF iterations

Then the researcher looked into effect of increasing feedback terms on the performance. Experiments that harvested 5 to 100 feedback terms from 20, 50, 80 and 100 feedback tweets as were shown in Figure 6. The term weight distribution was fixed at 0.1.

The results indicated that query expansion with feedback terms contributed to the performance improvement, although the improvement fluctuated as feedback terms increased. It was observed that the least improvement was achieved when the number of feedback terms was less than 10, and a number of feedback terms ranging from 10 to 35 terms would yield the maximum performance improvement in most cases.

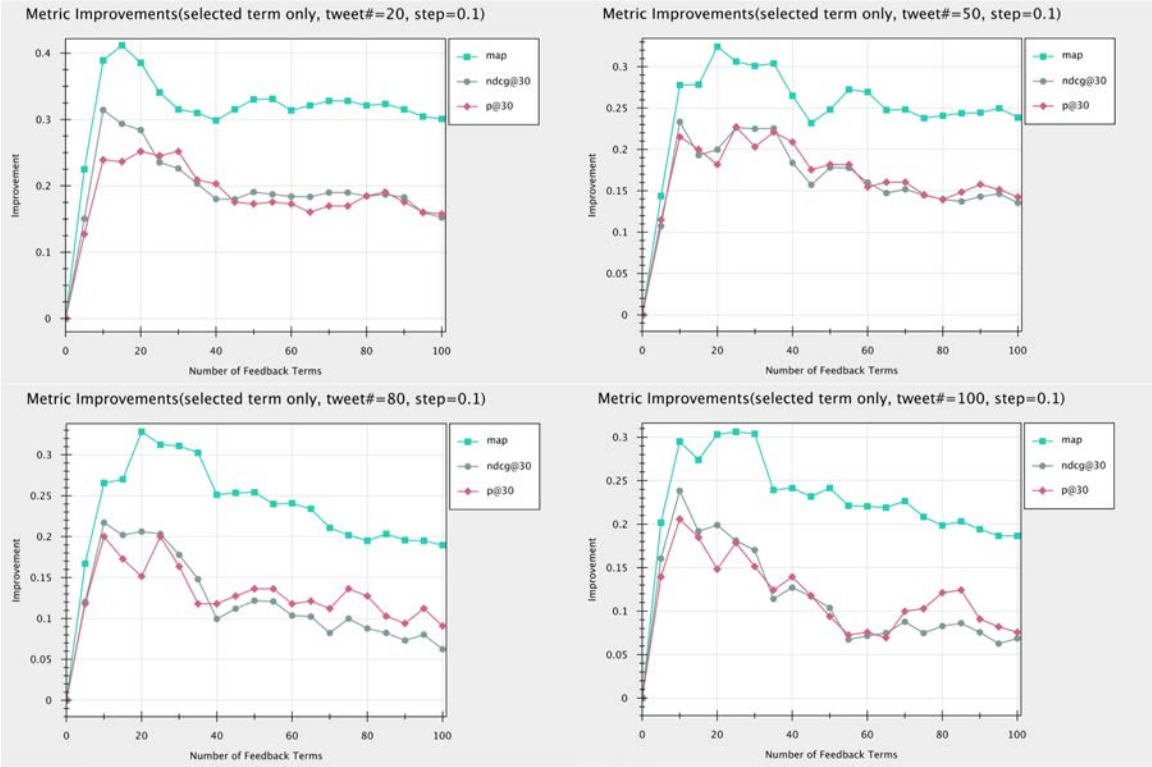

Figure 6 Metrics variation on increasing feedback terms

Term weight distribution was also an influential factor of pseudo-relevance feedback. The experiment focused on influence of the term-weighting decay on the variation of performance improvement. Term weight decays at a specific step size as the term rank decreases. Step sizes of term-weighting decay ranging from 0 to 1 were tested against different sets of feedback terms and tweets. The results were shown in Figure 7.

As was shown in the graph, influence of term-weighting decay on the performance improvement of microblog search varied as the number of feedback terms and tweets changed. When collecting many terms from a few tweets, MAP score improvement significantly increased, whereas improvements of P@30 and NDCG@30 scores fluctuated as the step size became larger. When the number of feedback terms and the

number of feedback tweets were equal, improvements of all the metric scores increased as the step size increased. However, when collecting a few feedback terms from many feedback tweets, improvements of all the metric scores fluctuated as the step size increased. The improvement of MAP score even declined when the weight was distributed sparsely.

The result of experiments related to influence of feedback tweets on search performance is shown in Figure 8. Apparently, pseudo-relevance feedback with only hashtags significantly crippled search performance. Although search performance was improved when expanding queries with both selected terms and hashtags, it was outperformed by query expansion with only feedback terms.

As for number of feedback tweets, search performance was strengthened when the number ranged between 5 and 30. But overall search performance declined as the number increased. Particularly, the performance slumped when the number of feedback tweets was huge.

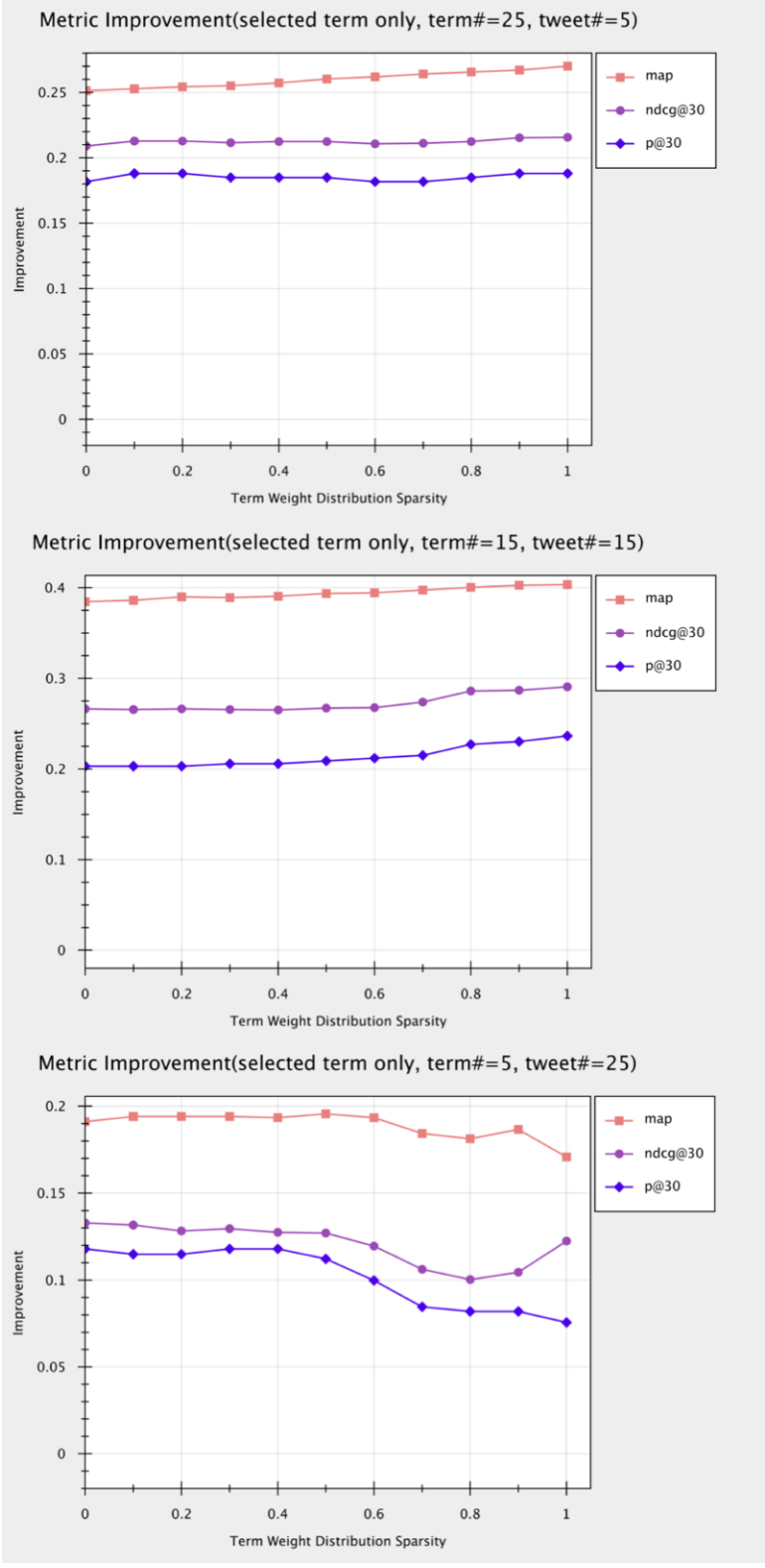

Figure 7 Metrics variation on increasing step size of term-weighting decay

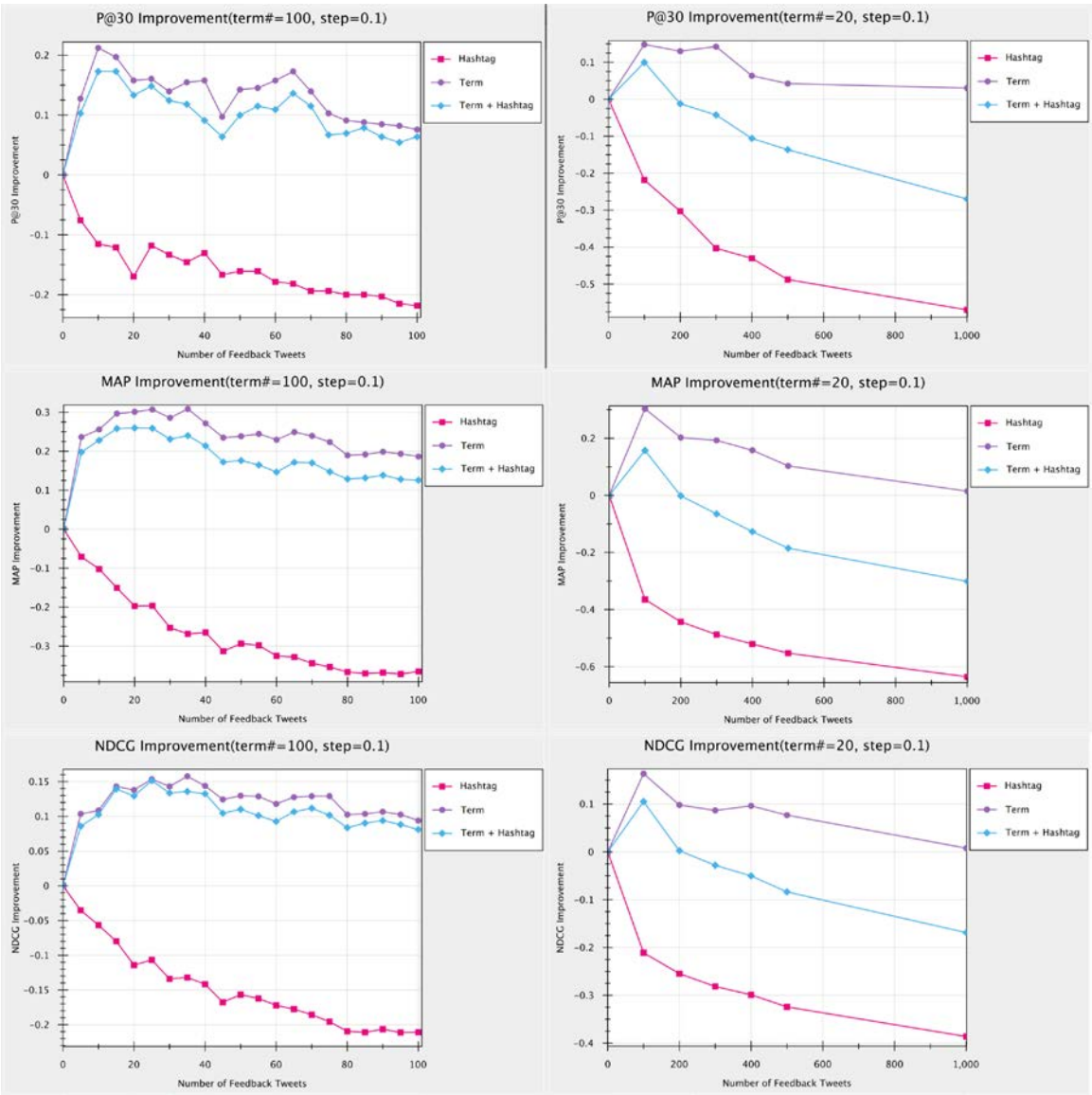

Figure 8 Metrics variation on increasing feedback tweets number and use of different genres of feedback terms

Discussion

A considerable amount of data has been derived from the experiments. After analyzing the resulting data, a number of observations of interest will be discussed in this section.

Regarding number of PRF iterations, it was expected that the effectiveness of PRF would be weakened as the iterations increased. It has been proved in relevance feedback (RF) that more than one-round RF iterations have marginally effectiveness on improving search performance (Manning et al., pp. 186). According to the variation observed from the experiments, the statement is also valid for pseudo-relevance feedback in the environment of microblog search.

In regard of number of feedback terms, it is surprising that the performance improvement is not significantly affected by too many feedback terms in the expanded queries, especially when the number of feedback tweets is large. The term weight distribution scheme applied in this study may favor it. The scheme assigns decreasing weights to feedback terms at lower ranks, so that influence of these less relevant terms on the search will be trivial. Another reason is that the number of meaningful feedback terms can be large when there are numerous feedback tweets. In this case, the search engine can acquire more information about the relevant tweets as the feedback terms increases. Thus, it explains the reason why the improvement even increases when many feedback terms are collected from a large number of feedback terms.

It was also observed that weighting query terms contributed to the performance of microblog search, but improvement it brought about varied as the number of feedback terms and tweets changed. The improvement increased as the step size increased when the number of feedback terms was larger than or equal to the number of feedback tweets. It was reasonable since the term weighting scheme aimed at differentiating terms at higher ranks and lower ranks. Large step size would let higher-ranked terms, which were suppose more relevant than lower-ranked terms, exert more influence on the search while weaken the impact of lower-ranked terms. However, the improvement declined as the step size increased when the number of feedback terms was significantly smaller than the number of feedback tweets. It was partly blamed on the term selection strategy, as a term's TF-IDF score could sometimes fail to reflect a term's relevancy. In this particular case, the five selected feedback terms at the top, which would probably be equally relevant, were nonetheless biased due to different TF-IDF scores. Thus, the system would mistakenly give higher ranks to documents that contained terms at rank one or two than those that contained term at rank four or five.

As expected, the improvement of search performance decreased as the number of feedback tweets increased. It was probably caused by weakness of term selection algorithm with TF-IDF scores, because relevant feedback terms will be "gamed" by increasing number of non-relevant ones as the size of feedback term pool scales. Tweets at low ranks may contain terms with high TF-IDF scores. These terms may replace relevant terms with relatively lower TF-IDF scores when the lower-ranked tweets are included into the feedback term pool. In this case, the search performance is affected.

Another interesting observation is that pseudo-relevance feedback with hashtags is harmful to performance of microblog search. The reason of this phenomenon is two-fold. On one hand, users are inclined to add meaningless or non-relevant hashtags into tweets, as average users have limited capability to recognize if a hashtag is really related to the topic of their tweets. Also, few users expect that hashtags created by them will used to facilitate microblog search, thus they are not obligated to define hashtags highly related to the topic of their tweets. For instance, when querying against the corpus using MB004 topic “Mexico drug war”, the researcher acquired a tweet at rank 11 saying “@sarah_tay2 I dont know wat would kill us 1st, the ice on the roads or the drug war going on in the border, between mexico&the US #imgood :)”. The tweet itself was related to the topic to certain extent, but the hashtag “#imgood” seemed used to inform the author’s situation at that point of time rather than relate this tweet to other tweets reporting this news. The researcher then searched for “imgood” to see if any tweets related to “Mexico drug war” could be returned. As expected, no relevant tweets were returned for this search. Therefore, expanding queries using hashtags like “imgood” would definitely introduce non-relevant tweets into the search result. On the other hand, since a tweet may be related to multiple topics and sometimes people tend to create multiple tweets to identify multiple topics covered in a single tweet, query expansion with hashtags may unnecessarily return tweets related to other topics in this case. For instance, the researcher identified tweet at rank 23 in the search result for the query “Mexico Drug War”. The tweet said “Another Drug Catapult Seized in Mexico http://goo.gl/fb/f5k0o #warnews #military #drugs #stateside”. The tweet was also related to the searching topic but contained hashtags that might be related to other topics. Particularly, by expanding

the query using hashtag “#warnews” and “#military”, a tweet saying “War In Afghanistan News 5 Feb 2011 http://goo.gl/fb/oCzsJ #warnews #military #afghanistannews” was returned at rank 10 in the search result, although the tweet was actually related to the war in Afghanistan rather than the Mexico drug war. In this study, this phenomenon was exacerbated without using any language model to filter out garbage hashtags such as “#a”, which made little sense.

Future Work

It is discovered in this study that refining queries directly using hashtags harvested from the pseudo-relevance feedback process harms the performance of microblog search, whereas hashtags can be identical symbols to identify sparse tweets from a large corpus. A potential extension of this study might be development of a reliable language model to select appropriate hashtags related to the query and weight the hashtags in the expanded query properly. Particularly, the language model should focus on topicality of each hashtag. It should examine if a harvested hashtag is relevant to the topic. An intuitive approach is to conduct a new search using each every harvested hashtag and check if the search results are relevant (probably using evaluation metrics used in this study). A hashtag will be used for query expansion only when its relevancy reaches a specific threshold set by the researcher. In this way, many non-relevant hashtags are likely to be excluded and search performance can be improved. Also, ranking and selecting hashtags by their TF-IDF scores may probably be a heuristic approach to filter out useless hashtags.

Term weighting scheme is another aspect of this study to be improved in future work. The scheme integrated in this study has obvious defect that affects the results of the study. The defect will be fixed in the future. In addition, the basic idea underlying the term weighting scheme is fairly simple. More complex schemes should be identified and may potentially contribute to performance of microblog search.

Finally, the study only focuses on ad hoc microblog retrieval, regardless of a variety of other searches in the context of microblog search, such as people search, entity search, etc. Further work is expected done on these areas.

Conclusion

It is found in the study that pseudo-relevance feedback can effectively improve performance of ad hoc microblog search in most cases. The largest improvement of 45.5% in MAP score has been achieved in this study. Several factors exert influence in the effectiveness of the studied approach, including number of PRF iterations, number of feedback terms, number of feedback tweets, genre of feedback terms and term weight distribution.

It is the finding that more than one-round PRF iteration will adversely influence the performance of microblog search. The influence of number of feedback terms fluctuates as other parameters vary. The search performance tends to decrease as the number of feedback tweets increases. Among the three genres of feedback terms investigated in this study, refining queries with only feedback terms brings about the best performance improvement, whereas use of hashtags even harms the search performance. Term weight distribution has minor influence on the search performance, but it may be partly due to the design defects existing in the term weighting scheme used in this study.

References

Cook, M. (2013). The Numbers Behind Twitter. Retrieved from http://www.masters-in-finance.org/twitter/.

McFedries, P. (2007). Technically speaking: All a-twitter. *Spectrum, IEEE,44*(10), 84-84.

Efron, M. (2011). Information search and retrieval in microblogs. *Journal of the American Society for Information Science and Technology*, *62*(6), 996-1008.

Smith, Craig (2014). By the numbers: 138 Amazing Twitter Statistics. Retrieved from: http://expandedramblings.com/index.php/march-2013-by-the-numbers-a-few-amazing-twitter-stats/#.Uz7SRR__5k8

Java, A., Song, X., Finin, T., & Tseng, B. (2007, August). Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis* (pp. 56-65). ACM.

Teevan, J., Ramage, D., & Morris, M. R. (2011, February). # TwitterSearch: a comparison of microblog search and web search. In *Proceedings of the fourt*

ACM international conference on Web search and data mining (pp. 35-44). ACM.

Efron, M. (2010, July). Hashtag retrieval in a microblogging environment. In*Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (pp. 787-788). ACM.

Hunter H. Janes. #Precision: An Exploration of the Utility of User-Generated Metadata for the Creation of Precise Microblog Query-Expansion Systems. A Master's Paper for the M.S. in I.S. degree. April, 2013.

Borlund, P. (2003). The IIR evaluation model: a framework for evaluation of interactive information retrieval systems. *Information research*, *8*(3), 8-3.

Voorhees, E. M. (2007). TREC: Continuing information retrieval's tradition of experimentation. Commun. ACM, 50, 51–54.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge: Cambridge university press.

Yu, S., Cai, D., Wen, J. R., & Ma, W. Y. (2003, May). Improving pseudo-relevance feedback in web information retrieval using web page segmentation.

In *Proceedings of the 12th international conference on World Wide Web* (pp. 11-18). ACM.

Yan, R., Hauptmann, A., & Jin, R. (2003). Multimedia search with pseudo-relevance feedback. In *Image and Video Retrieval* (pp. 238-247). Springer Berlin Heidelberg.

Lau, C. H., Li, Y., & Tjondronegoro, D. (2011). Microblog Retrieval Using Topical Features and Query Expansion. In *TREC*.

Bandyopadhyay, A., Ghosh, K., Majumder, P., & Mitra, M. (2012). Query expansion for microblog retrieval. *International Journal of Web Science*, *1*(4), 368-380.

Massoudi, K., Tsagkias, M., de Rijke, M., & Weerkamp, W. (2011). Incorporating query expansion and quality indicators in searching microblog posts. In *Advances in Information Retrieval* (pp. 362-367). Springer Berlin Heidelberg.

Mccandless, Michael (2012). Accuracy and Performance of Google's Compact Language Detector. Retrieved from: http://java.dzone.com/articles/accuracy-and-performance

Hatcher, E., Gospodnetic, O., & McCandless, M. (2004). Lucene in action.

Appendix I – Implementation of Term Collector

```
public class TermCollector {

	private final static String TEXT_FN = "text";
	private final static String HTAG_FN = "hashtag";
	private ScoreDoc[] scoreDocs;
	private IndexReader indexReader;
	private Map<String, Float> termMap;
	private Map<BytesRef, Float> idfMap;
	private Set<String> queryTerms;
	private DefaultSimilarity sim;
	private int numRetDocs;
	private int numTerms;

	public TermCollector(int numDocs, int numTerms){
		this(null, null, null, numDocs, numTerms);
	}

	public TermCollector(Query query, ScoreDoc[] scoreDocs, IndexReader indexReader,
int numDocs, int numTerms){
		this.scoreDocs = scoreDocs;
		this.indexReader = indexReader;
		termMap = new HashMap<String, Float>();
		idfMap = new HashMap<BytesRef, Float>();
		queryTerms = extractTerms(query);
		sim = new DefaultSimilarity();
		this.numRetDocs = numDocs;
		this.numTerms = numTerms;
	}
	/**
	 * Get top terms of current search
	 *
	 * @return
	 * @throws IOException
	 * @throws Exception
	 */
	public Map<String, Float> getTerms()
			throws ResetException, IOException{
		if(queryTerms == null || scoreDocs == null || indexReader == null)
			throw new ResetException("TermCollector must be reset before
reusing");
		rankTerms();
		Map<String, Float> tmpMap = new HashMap<String, Float>();
		int cnt = 0;
		for(String term : termMap.keySet()){
			if(cnt == numTerms) break;
			cnt ++;
			tmpMap.put(term, termMap.get(term));
		}

		Map<String, Float> sortedMap = new TreeMap<String, Float>(
				new ValueComparator(tmpMap));
		sortedMap.putAll(tmpMap);

		return sortedMap;
	}

	/**
	 *
	 * @return
	 * @throws IOException
```

```
	*/
	public Set<String> getHashtags()
			throws IOException{
		Set<String> htags = new HashSet<String>();
		for(int i = 0; i < ((numRetDocs < scoreDocs.length) ? numRetDocs :
scoreDocs.length); i ++){
			Terms v = indexReader.getTermVector(scoreDocs[i].doc, HTAG_FN);
			if(v == null) continue;
			TermsEnum te = v.iterator(null);
			BytesRef br;
			while((br = te.next()) != null){
				if(te.seekExact(br)){
					htags.add(br.utf8ToString());
				}
			}
		}
		return htags;
	}

	/**
	 * Get query terms of current search
	 *
	 * @return
	 */
	public Set<String> getQueryTerms(){
		return new HashSet<String>(queryTerms);
	}

	void reset(Query q, ScoreDoc[] scoreDocs, IndexReader indexReader){
		queryTerms = extractTerms(q);
		this.scoreDocs = scoreDocs;
		this.indexReader = indexReader;
	}

	void clean(){
		reset(null,null,null);
	}

	private void rankTerms()
			throws IOException{
		termMap = new HashMap<String, Float>();
		Directory tmpDir = new RAMDirectory();
		IndexWriter writer = new IndexWriter(tmpDir,
			new IndexWriterConfig(Version.LUCENE_46,
				new EnglishAnalyzer(Version.LUCENE_46)));

		for(int i = 0; i < ((numRetDocs < scoreDocs.length) ? numRetDocs :
scoreDocs.length); i ++)
			writer.addDocument(indexReader.document(scoreDocs[i].doc));
		writer.close();

		IndexReader ireader = DirectoryReader.open(tmpDir);
		int numDocs = ireader.numDocs();
		IndexSearcher searcher = new IndexSearcher(ireader);
		ScoreDoc[] docs = searcher.search(new MatchAllDocsQuery(),
numRetDocs).scoreDocs;
		for(ScoreDoc sd : docs){
			TermsEnum te = ireader.getTermVector(sd.doc,
TEXT_FN).iterator(null);
			BytesRef br;
			while((br = te.next()) != null){
				if(te.seekExact(br)){
					float idf;
					if(idfMap.containsKey(br)){
						idf = idfMap.get(br);
					}else{
						int docFreq = ireader.docFreq(new
Term(TEXT_FN, br));
						idf = sim.idf(docFreq, numDocs);
						idfMap.put(br, idf);
					}
```

```
                                        String term = br.utf8ToString();
                                        if(termMap.containsKey(term)){
                                                termMap.put(term, termMap.get(term) + idf);
                                        }else{
                                                termMap.put(term, idf);
                                        }
                                }
                        }
                }

                Map<String, Float> sortedMap = new TreeMap<String, Float>(
                                new ValueComparator(termMap));
                sortedMap.putAll(termMap);
                termMap = sortedMap;

                tmpDir.close();
        }

        private Set<String> extractTerms(Query q){
                if(q == null) return null;
                Set<Term> tset = new HashSet<Term>();
                Set<String> res = new HashSet<String>();
                q.extractTerms(tset);
                for(Term t : tset)
                        res.add(t.text());
                return res;
        }
}
```

Appendix II – Implementation of Analysis Component

```
public class Analysis {

	private static class plotConfig{
		String title;
		double maxX;
		double minX;
		String labelX;
		double maxY;
		double minY;
		String labelY;
		boolean isWithLegend;

		plotConfig(String title, double maxX, double minX, String labelX, double
maxY, double minY, String labelY, boolean isWithLegend){
			this.title = title;
			this.maxX = maxX;
			this.minX = minX;
			this.labelX = labelX;
			this.maxY = maxY;
			this.minY = minY;
			this.labelY = labelY;
			this.isWithLegend = isWithLegend;
		}
	}

	private final static int WIDTH = 700;
	private final static int HEIGHT = 500;
	private final static double TOP = 15.0;
	private final static double LEFT = 80.0;
	private final static double BOTTOM = 70.0;
	private final static double RIGHT = 160.0;
	private final static double RIGHT_NO_LEGEND = 80.0;
	private final static double Y_LABEL_DIST = 2.0;
	private final static String REC_BASE = System.getProperty("user.home") +
"/Documents/run";
	private final static String WITH_HTAG = "Hashtag";
	private final static String WITH_TERM = "Term";
	private final static String WITH_BOTH = "Term + Hashtag";
	private final static ArrayList<Shape> shapes = new ArrayList<Shape>();
	static{
		shapes.add(new Rectangle2D.Float(-4.0f, -4.0f, 8.0f, 8.0f));
		shapes.add(new Ellipse2D.Float(-4.0f, -4.0f, 8.0f, 8.0f));
		final GeneralPath d = new GeneralPath();
		d.moveTo(0.0f, -5.0f);
		d.lineTo(5.0f, 0.0f);
		d.lineTo(0.0f, 5.0f);
		d.lineTo(-5.0f, 0.0);
		shapes.add(d);
		final GeneralPath t = new GeneralPath();
		t.moveTo(0.0f, -5.0f);
		t.lineTo(5.0f, 5.0f);
		t.lineTo(-5.0f, 5.0f);
		t.closePath();
		shapes.add(t);
	}

	public static void main(String[] args)
			throws IOException {

		String[] metrics = {"p@30", "map", "ndcg@30"};
```

```
                plotIterations(5, 5, 0.1, metrics);
                plotIterations(5, 5, 0.5, metrics);
                plotIterations(5, 15, 0.1, metrics);
                plotIterations(15, 5, 0.1, metrics);


                plotTermNum(0, 100, 20, 0.1, metrics);
                plotTermNum(0, 100, 50, 0.1, metrics);
                plotTermNum(0, 100, 80, 0.1, metrics);
                plotTermNum(0, 100, 100, 0.1, metrics);

                plotStep(0, 1.0, 15, 5, metrics);
                plotStep(0, 1.0, 5, 15, metrics);
                plotStep(0, 1.0, 15, 15, metrics);

                plotDocNum(0, 100, 20, 0.1, "map");
                plotDocNum(0, 100, 20, 0.1, "ndcg");
                plotDocNum(0, 100, 20, 0.1, "p@30");
                plotDocNum(100, 1000, 20, 0.1, "p@30");
                plotDocNum(100, 1000, 20, 0.1, "map");
                plotDocNum(100, 1000, 20, 0.1, "ndcg");

        }

        //only selected terms, 5 terms, 5 docs, 0.1 step by default
        @SuppressWarnings("unchecked")
        public static void plotIterations(int termNum, int docNum, double step, String...
metrics)
                        throws IOException{
                double maxIterNum = 0,
                           maxImpr = 0,
                           minImpr = 0;
                Directory recDir = FSDirectory.open(new File(REC_BASE));
                IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(recDir));
                Query stepQuery = NumericRangeQuery.newDoubleRange("step", step, step,
true, true);
                Query docQuery = NumericRangeQuery.newIntRange("document number", docNum,
docNum, true, true);
                Query termQuery = NumericRangeQuery.newIntRange("term number", termNum,
termNum, true, true);
                Query htagOccurQuery = new TermQuery(new Term("with hashtags", "false"));
                Query termOccurQuery = new TermQuery(new Term("with selected terms",
"true"));

                BooleanQuery single = new BooleanQuery();
                single.add(stepQuery, BooleanClause.Occur.MUST);
                single.add(docQuery, BooleanClause.Occur.MUST);
                single.add(termQuery, BooleanClause.Occur.MUST);
                single.add(htagOccurQuery, BooleanClause.Occur.MUST);
                single.add(termOccurQuery, BooleanClause.Occur.MUST);

                Query multiple = new TermQuery(new Term("run type", "multiple
iterations"));
                BooleanQuery query = new BooleanQuery();
                query.add(single, BooleanClause.Occur.SHOULD);
                query.add(multiple, BooleanClause.Occur.SHOULD);

                TopDocs hits = searcher.search(query, Integer.MAX_VALUE);
                Map<String, DataTable> dsMap = new TreeMap<String, DataTable>();
                for(ScoreDoc sd : hits.scoreDocs){
                        Document d = searcher.doc(sd.doc);
                        int iterNum = d.getValues("metrics").length - 1;
                        for(String m : metrics){
                                double impr =
getValue(d.getField("improvement").stringValue(), m);
                                if(dsMap.containsKey(m)){
                                        dsMap.get(m).add(iterNum, impr);
                                }else{
                                        DataTable table = new DataTable(Integer.class,
Double.class);
                                        table.add(0, 0.0);
```

```
                                        table.add(iterNum, impr);
                                        dsMap.put(m, table);
                                }
                                maxIterNum = Math.max(iterNum, maxIterNum);
                                maxImpr = Math.max(impr, maxImpr);
                                minImpr = Math.min(impr, minImpr);
                        }
                }

                visualize(dsMap, new plotConfig(
                                "Metric Improvements\n(selected terms only, tweets#=" + docNum + ",
term#=" + termNum + ", step=" + step + ")",
                                maxIterNum + 1.0,
                                0,
                                "Number of PRF Iterations",
                                maxImpr + 0.01,
                                minImpr - 0.02,
                                "Improvement",
                                true));


                recDir.close();
                searcher.getIndexReader().close();
        }

        //only with terms and single iteration by default
        @SuppressWarnings("unchecked")
        public static void plotDocNum(int start, int end, int termNum, double step, String
metric)
                        throws IOException{
                int maxDocNum = 0;
                double maxImpr = 0;
                double minImpr = 0;

                Directory recDir = FSDirectory.open(new File(REC_BASE));
                IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(recDir));
                Query stepQuery = NumericRangeQuery.newDoubleRange("step", step, step,
true, true);
                Query docQuery = NumericRangeQuery.newIntRange("document number", start,
end, true, true);
                Query termQuery = NumericRangeQuery.newIntRange("term number", termNum,
termNum, true, true);
                Query runTypeQuery = new TermQuery(new Term("run type", "single
iteration"));

                BooleanQuery query = new BooleanQuery();
                query.add(stepQuery, BooleanClause.Occur.MUST);
                query.add(docQuery, BooleanClause.Occur.MUST);
                query.add(termQuery, BooleanClause.Occur.MUST);
                query.add(runTypeQuery,BooleanClause.Occur.MUST);

                TopDocs hits = searcher.search(query, Integer.MAX_VALUE);
                Map<String, DataTable> dsMap = new TreeMap<String, DataTable>();
                for(ScoreDoc sd : hits.scoreDocs){
                        Document d = searcher.doc(sd.doc);
                        int docNum = d.getField("document
number").numericValue().intValue();
                        boolean isWithHtag = Boolean.parseBoolean(d.getField("with
hashtags").stringValue());
                        boolean isWithTerm = Boolean.parseBoolean(d.getField("with selected
terms").stringValue());
                        String category;
                        if(isWithTerm && isWithHtag)
                                category = WITH_BOTH;
                        else if(isWithTerm)
                                category = WITH_TERM;
                        else
                                category = WITH_HTAG;
                        double impr = getValue(d.getField("improvement").stringValue(),
metric);
                        if(maxDocNum < docNum) maxDocNum = docNum;
```

```
                        if(maxImpr < impr) maxImpr = impr;
                        if(minImpr > impr) minImpr = impr;

                        if(dsMap.containsKey(category)){
                                dsMap.get(category).add(docNum, impr);
                        }else{
                                DataTable table = new DataTable(Integer.class,
Double.class);
                                //insert a dummy tuple to show improvement between baseline
run and following iterations
                                table.add(0, 0.0);
                                table.add(docNum, impr);
                                dsMap.put(category, table);
                        }
                }

            visualize(dsMap, new plotConfig(
                        metric.toUpperCase() + " Improvement(term#=" + termNum + ", step="
+ step + ")",
                        maxDocNum + 1.0,
                        0,
                        "Number of Feedback Tweets",
                        maxImpr + 0.01,
                        minImpr - 0.02,
                        metric.toUpperCase() + " Improvement",
                        true));

            recDir.close();
            searcher.getIndexReader().close();
        }

        //only with terms and single iteration by default
        @SuppressWarnings("unchecked")
        public static void plotTermNum(int start, int end, int docNum, double step,
String... metrics)
                        throws IOException{
                int maxTermNum = 0;
                double maxImpr = 0;
                double minImpr = 0;

                Directory recDir = FSDirectory.open(new File(REC_BASE));
                IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(recDir));
                Query stepQuery = NumericRangeQuery.newDoubleRange("step", step, step,
true, true);
                Query docQuery = NumericRangeQuery.newIntRange("document number", docNum,
docNum, true, true);
                Query termQuery = NumericRangeQuery.newIntRange("term number", start, end,
true, true);
                Query withTerm = new TermQuery(new Term("with hashtags", "false"));
                Query withHtags = new TermQuery(new Term("with selected terms", "true"));
                Query runTypeQuery = new TermQuery(new Term("run type", "single
iteration"));

                BooleanQuery query = new BooleanQuery();
                query.add(stepQuery, BooleanClause.Occur.MUST);
                query.add(docQuery, BooleanClause.Occur.MUST);
                query.add(termQuery, BooleanClause.Occur.MUST);
                query.add(runTypeQuery,BooleanClause.Occur.MUST);
                query.add(withTerm, BooleanClause.Occur.MUST);
                query.add(withHtags, BooleanClause.Occur.MUST);

                TopDocs hits = searcher.search(query, Integer.MAX_VALUE);
                Map<String, DataTable> dsMap = new TreeMap<String, DataTable>();
                for(ScoreDoc sd : hits.scoreDocs){
                        Document d = searcher.doc(sd.doc);
                        int termNum = d.getField("term number").numericValue().intValue();
                        for(String m : metrics){
                                double impr =
getValue(d.getField("improvement").stringValue(), m);
                                if(dsMap.containsKey(m)){
                                        dsMap.get(m).add(termNum, impr);
```

```
                                }else{
                                        DataTable table = new DataTable(Integer.class,
Double.class);
                                        table.add(0, 0.0);
                                        table.add(termNum, impr);
                                        dsMap.put(m, table);
                                }
                                if(maxTermNum < termNum) maxTermNum = termNum;
                                if(maxImpr < impr) maxImpr = impr;
                                if(minImpr > impr) minImpr = impr;
                        }
                }

            visualize(dsMap, new plotConfig(
                                "Metric Improvements(selected term only, tweet#=" + docNum
+ ", step=" + step + ")",
                                maxTermNum + 1.0,
                                0,
                                "Number of Feedback Terms",
                                maxImpr + 0.01,
                                minImpr - 0.02,
                                "Improvement",
                                true));

            recDir.close();
            searcher.getIndexReader().close();
        }

        //only with terms and single iteration by default
        @SuppressWarnings("unchecked")
        public static void plotStep(double start, double end, int docNum, int termNum,
String... metrics)
                        throws IOException{
                double maxStep = 0,
                           maxImpr = 0,
                           minImpr = 0;

                Directory recDir = FSDirectory.open(new File(REC_BASE));
                IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(recDir));
                Query stepQuery = NumericRangeQuery.newDoubleRange("step", start, end,
true, true);
                Query docQuery = NumericRangeQuery.newIntRange("document number", docNum,
docNum, true, true);
                Query termQuery = NumericRangeQuery.newIntRange("term number", termNum,
termNum, true, true);
                Query withTerm = new TermQuery(new Term("with hashtags", "false"));
                Query withHtags = new TermQuery(new Term("with selected terms", "true"));
                Query runTypeQuery = new TermQuery(new Term("run type", "single
iteration"));

                BooleanQuery query = new BooleanQuery();
                query.add(stepQuery, BooleanClause.Occur.MUST);
                query.add(docQuery, BooleanClause.Occur.MUST);
                query.add(termQuery, BooleanClause.Occur.MUST);
                query.add(runTypeQuery,BooleanClause.Occur.MUST);
                query.add(withTerm, BooleanClause.Occur.MUST);
                query.add(withHtags, BooleanClause.Occur.MUST);

                TopDocs hits = searcher.search(query, Integer.MAX_VALUE);
                Map<String, DataTable> dsMap = new TreeMap<String, DataTable>();
                for(ScoreDoc sd : hits.scoreDocs){
                        Document d = searcher.doc(sd.doc);
                        double step = d.getField("step").numericValue().doubleValue();
                        for(String m : metrics){
                                double impr =
getValue(d.getField("improvement").stringValue(), m);
                                if(dsMap.containsKey(m)){
                                        dsMap.get(m).add(step, impr);
                                }else{
                                        DataTable table = new DataTable(Double.class,
Double.class);
```

```
                                        table.add(step, impr);
                                        dsMap.put(m, table);
                                }
                                if(maxStep < step) maxStep = step;
                                if(maxImpr < impr) maxImpr = impr;
                                if(minImpr > impr) minImpr = impr;
                        }
                }
                visualize(dsMap, new plotConfig(
                        "Metric Improvement(selected term only, term#=" + termNum + ",
tweet#=" + docNum +")",
                        maxStep + 0.05,
                        0,
                        "Term Weight Distribution Density",
                        maxImpr + 0.01,
                        minImpr - 0.02,
                        "Improvement",
                        true));

            recDir.close();
            searcher.getIndexReader().close();
        }


        public static void visualize(Map<String, DataTable> data, plotConfig config){
                DataSeries[] dsList = new DataSeries[data.keySet().size()];
                int i = 0;
                for(String cat : data.keySet()){
                        DataTable table = data.get(cat);
                        table.sort(new Ascending(0));
                        DataSeries ds = new DataSeries(cat, table, 0, 1);
                        dsList[i ++] = ds;
                }

                XYPlot plot = new XYPlot(dsList);
                if(config.isWithLegend){
                        plot.setLegendVisible(config.isWithLegend);
                        plot.setLegendLocation(Location.EAST);
                        plot.setLegendDistance(0.4);
                        plot.setInsets(new Insets2D.Double(TOP, LEFT, BOTTOM, RIGHT));
                }else{
                        plot.setInsets(new Insets2D.Double(TOP, LEFT, BOTTOM,
RIGHT_NO_LEGEND));
                }

            plot.getTitle().setText(config.title);
            plot.getAxis(XYPlot.AXIS_X).setMin(config.minX);
            plot.getAxis(XYPlot.AXIS_X).setMax(config.maxX);
            plot.getAxisRenderer(XYPlot.AXIS_X).setLabel(config.labelX);
            if(config.minX >= 0)
                plot.getAxisRenderer(XYPlot.AXIS_X).setIntersection(-Double.MAX_VALUE);
            plot.getAxis(XYPlot.AXIS_Y).setMin(config.minY);
            plot.getAxis(XYPlot.AXIS_Y).setMax(config.maxY);
            if(config.minY >= 0)
                plot.getAxisRenderer(XYPlot.AXIS_Y).setIntersection(-Double.MAX_VALUE);
            plot.getAxisRenderer(XYPlot.AXIS_Y).setLabel(config.labelY);
            plot.getAxisRenderer(XYPlot.AXIS_Y).setLabelDistance(Y_LABEL_DIST);
            for(i = 0; i < dsList.length; i ++){
                DataSeries ds = dsList[i];
                plot.setLineRenderer(ds, new DefaultLineRenderer2D());

                float a = (float)Math.random(),
                          b = (float)Math.random(),
                          c = (float)Math.random();

                Color color = new Color(a, b, c);
                plot.getPointRenderer(ds).setColor(color);
                plot.getPointRenderer(ds).setShape(shapes.get(i));
                plot.getLineRenderer(ds).setColor(color);
            }
```

```
                JDialog dialog = new JDialog();
                dialog.setTitle(config.title);
                dialog.setSize(WIDTH, HEIGHT);
                dialog.setResizable(true);

                JPanel contentPane = new JPanel();
                contentPane.setLayout(new BorderLayout());
                dialog.setContentPane(contentPane);
                dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

                contentPane.add(new InteractivePanel(plot), BorderLayout.CENTER);
                dialog.setVisible(true);
        }

        public static double getValue(String mapstr, String metric){
                Map<String, Double> imprs = new HashMap<String, Double>();
                String[] pairs = mapstr.toLowerCase().replaceAll("[{} ]", "").split(",");
                for(String p : pairs){
                        if(p.startsWith("p_"))
                                p = p.replaceAll("_", "@");
                        else if(p.startsWith("ndcg_"))
                                p = p.replaceAll("_cut_", "@");
                        String[] kv = p.split("=");
                        imprs.put(kv[0], Double.parseDouble(kv[1]));
                }
                if(!imprs.containsKey(metric)){
                        System.err.println(metric + " not found.");
                        System.exit(1);
                }
                return imprs.get(metric);
        }
}
```