Webis at TREC 2015: Tasks and Total Recall Tracks

Matthias Hagen Steve Göring Magdalena Keil Olaoluwa Anifowose
Amir Othman Benno Stein

Bauhaus-Universität Weimar
99421 Weimar, Germany
<first name>.<last name>@uni-weimar.de

ABSTRACT

In this paper we give a brief overview of the Webis group's participation in the TREC 2015 Tasks and Total Recall tracks.

In the task understanding subtask of the Tasks track, we use different data sources (AOL query log, Freebase, etc.) and APIs (Google, Bing, etc.) to retrieve topics related to a given query. All sources are combined in our SQuare system. The task completion subtask is based on combining the results of our ChatNoir 2 for the different topics identified in the task understanding subtask. Finally, for the ad-hoc subtask (similar to the previous years' Web tracks), we use an axiomatic re-ranking approach of the search results obtained from ChatNoir 2.

In the Total Recall track, we employ a simple SVM baseline with variable batch sizes equipped with a keyqueries step to identify potentially relevant documents.

1. TASKS TRACK

The Tasks track has three subtasks: task understanding, task completion, and the ad-hoc subtask that is similar to previous years' TREC Web tracks. For each of the subtasks, we briefly describe our approach.

1.1 Task Understanding

The goal of the task understanding subtask is to automatically identify related queries for all possible aspects or topics a given user query may consist of. For each of 50 given user queries up to 100 related queries should be generated. Our proposed system is implemented as a web service called SQuare[1] (**s**earch for **qu**eries that **a**re **re**lated). The general approach is divided in two steps: acquisition of related queries and scoring of each query.

1.1.1 *Acquisition of Related Queries*

In the first step, we generate related queries using different APIs and datasets: query suggestion APIs, Graph/RDF-based, search result-based, AOL query log analysis, Netspeak completion and ChatNoir phrase extraction.

Suggestion APIs.

We use the query suggestion APIs from Google and Bing for each given query. Since both APIs typically cover similar related queries, on average we obtained 6–8 related queries from these APIs.

[1]`http://webis16.medien.uni-weimar.de/square/`

AOL query log analysis.

We have divided the AOL query log [8] into search sessions and missions [5]. For a given query q, we then identify all search sessions that contain a query with a *tf*-weighted cosine similarity to q of at least 0.8. All queries of such sessions are then used as potentially related queries.

Graph, RDF.

Since most queries of the Tasks track come with annotated Freebase entities (ID and name), we submit requests for similar entities/topics to the Interestgraph [11], WikiData [13], and Freebase [3]. All retrieved topics are used as potentially related queries.

Search Results.

We submit the given user query to the general Google web search [4] and to the Wikipedia search [14] to retrieve the top-10 search results. For each search result we get the title and use the terms with at least two characters as potential query phrases.

Netspeak Frequent Phrases.

For a query q, we use Netspeak [10, 12] to find related queries as follows. Let w_1 and w_n be the first and last word in q, respectively. We then submit the request $* \, w_1 * w_n *$ to Netspeak. The query results are the most frequent phrases containing w_1 and w_n, where the $*$-operator matches zero or more words. The top-10 Netspeak results are used as potentially related queries.

ChatNoir Keyphrase Extraction.

Our last data source for related queries is ChatNoir [9]. We retrieve the top-10 results and extract the top-10 keyphrases from the main content [7] using a head noun phrase extractor [1].

1.1.2 *Query Scoring*

Based on the above different techniques of the acquisition step, we obtain a set Q of potentially related queries for a given query q. For the different approaches, our pilot studies showed different qualities of the found potentially related queries such that we assign different weights to queries found by either technique reflecting the quality: Google suggest 100, Bing suggest 90, AOL query log sessions 80, Interestgraph 70, Wikipedia search result titles 60, Google search result titles 50, Freebase 40, Wikidata 30, Netspeak frequent phrases 20, ChatNoir search result keyphrases 10.

A query found by different approaches gets the summed weights as its score and the queries are ranked by descending scores. Queries achieving the same score are ordered by the number of ClueWeb12 results found with ChatNoir 2 preferring queries with more results.

1.1.3 Run webis1

We only had one run based on the top-100 related queries we found. However, having known before that the pooling depth for evaluation would be 20, we would have submitted two other runs having ranks 21–40 and 41–60 also be judged. On average, we extracted about 250 different related queries for a given query. From these, the highest scoring 100 queries are used.

1.2 Task Completion

Our runs for the task completion and ad-hoc subtask are on the full ClueWeb12 corpus (category A) using the ChatNoir 2 search engine to retrieve an initial baseline result set, that is then potentially re-ranked. ChatNoir 2 is the successor of ChatNoir [9]; it is built on ElasticSearch using BM25F as the retrieval model.

As for the task completion subtask, based on a given user query, all documents should be given as output that are relevant to any task a user may be trying to fulfill. In our general approach for this subtask, we use the related queries from the task understanding subtask (cf. Section 1.1) as the task set. For each related query individually, we retrieve documents using ChatNoir 2. Afterwards, we combine the top-10 retrieved documents of the different related queries in different ways. Note that we do not include the original query here although it might still be the best description of the user need. The reason is that our runs for the ad-hoc task will be based on the original query and we wanted to avoid getting similar judgments for two different tasks. The final result list containing results for the original query and the related ones might of course achieve better evaluation results than the combined list of the related queries or the results of the original query individually.

1.2.1 Runs

For this subtask, we assume a low pooling depth having the third run use documents not in the top-20 of the other two runs. The underlying idea is to obtain judgments for more documents.

webisC1.

In our first run, we use a simple interleaving approach. The first result is the first rank from the highest scoring related query, the second result is the highest rank from the second highest scoring related query, etc. Results that are already contained in the interleaved ranking are not used again.

webisC2.

Our second run weights the top-10 retrieved documents for each related query based on the number of related queries that have the result in their top-10. The most frequent results form the combined result set. In case of same frequencies, the task understanding score of the query that has the result in the highest result rank determines; if even this is equal, the order is chosen at random.

webisC3.

In our last run, we again use the interleaving approach of our first run, but skip documents that are contained in the top-20 of our first two runs. The idea is to obtain some more judgments for other documents.

1.3 Ad-hoc Task

The ad-hoc subtask is similar to the past years' TREC Web tracks. For a given query, a ranked list of documents should be returned. In the last year, we participated in the Web track with an early prototype of an axiom-based re-ranking framework [6]. As for this year's ad-hoc subtask, we improved many parts of the framework; especially we added more axioms.

The basic idea remains the same but we use the new ChatNoir 2 setup and pre-trained static axiom weights obtained in pilot experiments for BM25F or $tf \cdot idf$ as baseline retrieval models.

1.3.1 Runs

Again, we have three runs that assume a low pooling depth for judgment and have the lower ranks of our first runs as the top ranks in the third run to obtain judgments for more documents.

webisA1.

We axiomatically re-rank the ChatNoir 2 baseline results using the BM25F axiom weights.

webisA2.

The results of webisA1 from which the top-20 were removed. Again, the idea is to simply get more judgments in case that the pooling depth will be set to 20.

webisA3.

We axiomatically re-rank the results of a $tf \cdot idf$ baseline retrieval model. From the final ranking, the top-20 results of our first two runs are excluded to obtain judgments for more documents. The original $tf \cdot idf$ ranking can be evaluated by re-inserting the excluded documents.

2. TOTAL RECALL TRACK

The objective of the Total Recall track is to retrieve all relevant documents for a given topic with as little effort as possible. The components involved in our system include Apache Lucene 5.3 (used for indexing and searching employing the BM25 retrieval model), MongoDB 3.0 (database for storage), and LIBSVM (for training an SVM classifier). Our two runs (baseline and keyphrase) are separated into four separate steps: (1) initialization, (2) first iteration, (3) subsequent iterations, (4) finalization. Basically, the only difference of our two runs is in the subsequent iterations step.

2.1 Baseline Run

Initialization.

As a preprocessing step, the $tf \cdot idf$ scores for the vocabulary in the corpus are derived and stored in a key-value store. The $tf \cdot idf$ weighted document vectors will serve as the feature input of an SVM classifier used in later steps. However, when the number of documents in the corpus increases, there is a possibility that our system substantially slows down due to the SVM classifier. Our pilot experiments show that this effect occurs around corpus sizes of 300,000 documents. We perform sampling to remedy this situation. If the corpus has more than 300,000 documents, a subset of 300,000 documents would be chosen randomly.

An index for the corpus is built using Apache Lucene's BM25 retrieval model with default parameter settings.

First Iteration.

In each iteration, we send a number of documents to the "user" (i.e., the organizer's API) for relevance judgment. We call this number the batch size. During the first iteration, the batch size is set to 32. The size of 32 is chosen since our pilot experiments showed it to represent a good compromise between sending too many or too few documents for judgment. And since it is a multiple of 2, it is easy to later halve or double the batch size.

For a given topic, we start by removing the stop words; however, when more than 50% of a topic's terms are stop words, we keep them. The processed topic is then run as an ad-hoc query against the BM25 index. We consider the top-128 results (again a multiple of 2) as relevant and label them as positive examples for the SVM classifier. Another 128 random documents that do not appear in the ad-hoc query's result are labeled as negative examples for the SVM. The SVM is then trained on the labeled documents and used to classify all other corpus documents (remember that it could only be a sample for large corpora). From the resulting ranking, the 16 top-ranking documents are chosen that are not among the top-16 results of the ad-hoc query. These 16 SVM-classified documents are combined with the top-16 results of the ad-hoc query and submitted to the user for judgment. The response is a list of true labels for the 32 documents (relevant or not) that is used in the subsequent iterations.

Subsequent Iterations.

After each iteration, the batch size may change by being doubled or halved (easy due to the initial multiple of 2). There will be a point in a run where the next batch size becomes zero due to rounding. At that point, our approach decides to stop. In case that the batch size is not zero, the following steps will be performed.

First, the relevance feedback from the previous iteration is used to create a new training set for the SVM classifier. This is done by labeling the user-defined relevant documents as positive and the non-relevant documents as negative. Since the number of relevant and non-relevant documents might differ, we try to balance the sets to reduce bias. If the number of non-relevant documents is less than the number of relevant documents, some random documents from the corpus that have not been judged as relevant are added to the negative examples. If the number of non-relevant documents is larger than the number of relevant documents, non-relevant documents are removed at random until the number of relevant document is reached.

The trained SVM classifier is then used to classify the corpus documents not judged before. The top-n documents the SVM classifier judges as relevant are selected to send for judgment retrieval. Here, n is the batch size that depends on the previous iteration: It is doubled from the previous batch size if the ratio of relevant documents to non-relevant documents was greater than 0.5 in the previous iteration. It is halved if the ratio of relevant documents to non-relevant documents was less than 0.4. Otherwise, the previous batch size is not changed. To avoid sending too many documents, we set the maximum batch size to 2048 for athome1 and athome2 and to 4096 for all other corpora. If the maximum batch size is reached, the above rules would be ignored and the following would be applied: The batch size is divided by 16 if the ratio of relevant documents to non-relevant documents is less than 0.4 and halved otherwise.

Finalization.

Eventually, the batch size will become smaller since the number of relevant documents not shown to the user is monotonically decreasing. When the next batch size becomes zero, a run on a topic is finalized.

2.2 Run with Keyphrase Extraction

In this run, a keyphrase extraction component is added to enrich the set of potentially positive examples in each iteration instead of shrinking the set of judged non-relevant documents. The only difference to the baseline is the treatment of subsequent iterations (i.e., iterations following the first iteration). The other parts are identical to the baseline run.

Subsequent Iterations.

In case that more than 128 documents were judged as relevant in previous iterations, we choose 128 at random that were not used for keyphrase extraction in previous iterations. For these 128 documents, we compute the pairwise $tf \cdot idf$-weighted cosine similarities and remove all similarities below 0.2. From the remaining similarities, we select the four documents with the highest pairwise cosine similarities with the assumption that these deal with similar topics and were judged as relevant by the user. If no such documents can be found (e.g., when all similarities are below 0.2 or no relevant documents exist), the keyphrase extraction is skipped. In case that four documents could be identified, we extract the top-10 head noun keyphrases [1] from their concatenated main contents [7]. These keyphrases are used to form a keyquery for as many of the documents as possible. A keyquery for a document set is a query that retrieves the documents in its top-k results [2]. In our case, we employ the Lucene BM25 index as the reference search engine and set $k = 32$. From a keyquery the top-128 results not previously judged as relevant are used as additional positive examples for training the SVM classifier (in addition to the documents already judged as relevant). In this case, not that many non-relevant documents have to be removed for balancing than was the case in the baseline run. The negative examples again are the documents judged as non-relevant.

All other Steps.

The other three steps and the batch size adapting strategy of the subsequent iterations step are identical to the baseline run.

3. REFERENCES

[1] K. Barker and N. Cornacchia. Using noun phrase heads to extract document keyphrases. In *Proceedings of AI 2000*, pages 40–52.

[2] T. Gollub, M. Hagen, M. Michel, and B. Stein. From keywords to keyqueries: Content descriptors for the web. In *Proceedings of SIGIR 13*, pages 981–984.

[3] Google. Freebase data dumps. `https://developers.google.com/freebase/data`, 2015.

[4] Google. Web search API, 2015.

[5] M. Hagen, J. Gomoll, A. Beyer, and B. Stein. From search session detection to search mission detection. In *Proceedings of OAIR 2013)*, pages 85–92.

[6] M. Hagen, S. Göring, M. Michel, G. Müller, and B. Stein. Webis at TREC 2014: Web, Session, and Contextual Suggestion tracks. In *Proceedings of TREC 2014*.

[7] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *Proceedings of WSDM 2010*, pages 441–450.

[8] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Proceedings of Infoscale 2006*, paper 1.

[9] M. Potthast, M. Hagen, B. Stein, J. Graßegger, M. Michel, M. Tippmann, and C. Welsch. ChatNoir: A search engine for the ClueWeb09 corpus. In *Proceedings of SIGIR 2012*, page 1004.

[10] M. Potthast, M. Trenkmann, and B. Stein. Netspeak: Assisting writers in choosing words. In *Proceedings of ECIR 2010*, page 672.

[11] Prismatic. InterestGraph API. `http://interest-graph.getprismatic.com/`, 2015.

[12] Webis. Netspeak API. `http://netspeak.org`, 2015.

[13] Wikidata. Knowledge base API. `https://www.wikidata.org/`, 2015.

[14] Wikipedia. Web API, 2015.