Webis at TREC 2016: Tasks, Total Recall, and Open Search Tracks

Matthias Hagen Johannes Kiesel Payam Adineh Masoud Alahyari
Ehsan Fatehifar Arefeh Bahrami Pia Fichtl Benno Stein

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

ABSTRACT

We give a brief overview of the Webis group's participation in the TREC 2016 Tasks, Total Recall, and Open Search tracks.

Our submissions to the Tasks track are similar to our last year's system. In the task understanding subtask of the Tasks track, we use different data sources (ClueWeb12 anchor texts, AOL query log, Wikidata, etc.) and APIs (Google, Bing, etc.) to retrieve suggestions related to a given query. For the task completion and ad-hoc subtask, we combine the results of the Indri search engine for the different related queries identified in the task understanding subtask.

Our system for the the Total Recall track also is similar to our last year's idea with some slight changes in the details; we employ a simple SVM baseline with variable batch sizes equipped with a keyqueries step to identify potentially relevant documents.

In the Open Search track, we axiomatically re-rank a BM25-ordered result list to come up with a final document ranking.

1. TASKS TRACK

The Tasks track has three subtasks: task understanding, task completion, and the ad-hoc for each of which 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 cover.

Like last year [5], we implement a two-step approach: (1) collecting for each user query a number of related queries suggested by the different modules listed below, and (2) ranking them to select queries for the final output. The suggested related queries are ranked summing up simple scores that depend on the importance of the different suggestion modules. We derived these scores in a manual pilot experiment in which we assessed the output of the different modules for non-test-set queries (scores are shown in parenthesis below).

Google Suggestions (100). Like in the last year, we submit the queries to Google and collect the query suggestions.
Bing Suggestions (90). Analogous to the Google suggestions, but using Bing search.
Anchor Text Graph (80). We adopted last year's idea of Bennett and White [2] and used the ClueWeb12 Anchor Text Graph to find similar queries. The assumption is that the texts in HTML anchors (with `href` attribute) are short descriptions of the documents they link to. Therefore, such *anchor texts* can be seen as alternative descriptions of the document they link to. Our approach works as follows. In a pre-processing step, we use Mirex [8] to extract anchor texts for the URLs in the ClueWeb12 and store them with their frequency of being anchor texts in an Apache Lucene 2.4.1 index. Given a query, we retrieve from the index the anchor texts similar to the query, ordered by their frequency. For post-processing, we remove all anchor texts that contain dates, fewer words than the original query (since we want to find more specific queries), or that have a *tf*-weighted cosine similarity of less than 0.3 to the original query (to remove vastly different suggestions).
Google Autocomplete (75). We use the Google autocompletion API to get additional suggestions for related queries. Different to Google suggestions, this API only returns queries that have the original query as a prefix.
AOL query log sessions (70). First, we split the AOL query log [11] into search sessions [4] (i.e., sets of queries submitted for the same information need by some user). For a given query q, we then retrieve all search sessions that contain a query having a *tf*-weighted cosine similarity of at least 0.8 compared to q. All queries of such sessions then form potentially related queries with the idea that other users submitted them for some information need related to q.
Netspeak Frequent Phrases (60). For a query q, we use Netspeak [13, 14] to find related queries as follows. Let w_1 be the first and w_n be the last word of q. We then sent the request $* \, w_1 * w_n *$ to Netspeak. The query results are the most frequent phrases containing w_1 and w_n, with $*$ matching zero or more words. The top-10 results (ordered by frequency) are used as potentially related queries.
Wikidata (50). Since most queries of the Tasks track come with annotated Freebase entities (ID and name), we submit requests for similar entities/topics to Wikidata [15]. All retrieved topics are used as potentially related queries.
Freebase (30). The same as Wikidata, but using the Wikidata API that allows access to the old Freebase database.
ChatNoir Keyphrase Extraction (5). Our last data source for related queries is ChatNoir [12]. We retrieve the top-10 results and extract the top-10 keyphrases from their main content [9] using a head noun phrase extractor [1].

Runs

We submitted the query suggestions with the highest scores (sum of the importance values of the modules suggesting a

query) in the run webis1. In order to evaluate each module, we also submitted the queries that score highest in the individual modules but were not part of webis1 as runs webis2 and webis3. Since we did not know the pooling depth for the evaluation, we used a round-robin approach to distribute the individually highest scoring suggestions across these two runs.

1.2 Task Completion & Ad-hoc

The setting of the task completion subtask is as follows: given a user query, return all documents that are relevant/useful to any task a user may be trying to fulfill with the query. For the ad-hoc subtask, a ranked list of documents fitting the user query should be returned.

Runs

Our runs are on the full ClueWeb12 corpus (category A) and contain the top-3 results returned by the Indri search engine [10] for the queries we found in the task understanding subtask. We use the same documents and ranking for the task completion and the ad-hoc subtask. Runs webisC1 and webisA1 use the related queries from run webis1; webisC2 and webisA2 the ones from webis2; and webisC3 and webisA3 the ones from webis3. We create the list of documents by taking the top-3 documents that the Indri search engine returns for the highest ranking query from the task understanding subtask, then adding the top-3 from the second highest and so on. We then filter out all duplicates in the list, and sent the list to the server for evaluation.

2. TOTAL RECALL TRACK

The objective of the Total Recall track is to return all(!) relevant documents for a given topic without too many irrelevant results. Like last year [5], we submit documents in several iterations until a stopping criterion is met. Our two runs are equal in the first iteration and the stopping criterion, but differ in the other iterations.

In a pre-processing, we index the respective document set using Apache Lucene's BM25 retrieval model with default parameter settings. We also pre-compute the *tf·idf* scores for each document to use them for training a LibSVM classifier. For each topic (i.e., query) we proceed as follows.

First Iteration. We obtain the first 512 results for the user query from Lucene and submit them for evaluation.

Subsequent Iterations, Baseline Approach. We use the judgements obtained from the previous iteration(s) to train an SVM-classifier using the LibSVM library. We take the *tf·idf* vectors of all relevant documents as positive examples and the ones from all irrelevant documents as negative examples, but as most as many as we have positive samples since we expect a majority of the results being irrelevant. The trained SVM is then used to classify all documents that were not yet submitted for evaluation. We rank these documents by the classifier's confidence for the positive class and submit the top-n documents in the current iteration. The value of n may change from iteration to iteration depending on how well the classifier did in the previous iteration:

- If the ratio of relevant to irrelevant documents in the last submission was greater or equal to 2, we double n if it is not larger than 1024 and do not change it otherwise (i.e., maximum "batch size" is 2048).
- If the ratio of relevant to irrelevant documents in our last submission was less than 0.4, we halve n if it is larger than 128 and do not change it otherwise.
- If the ratio of relevant to irrelevant documents in our last submission was even less than 0.1, we halve n if it is larger than 64.
- Otherwise, we don't change n

In the rare case that the SVM cannot find relevant documents not submitted before, we submit 128 random documents that were not submitted before.

Subsequent Iterations, Keyqueries Approach. In our second approach, we enhance the baseline with a diversification algorithm that uses keyqueries.

A keyquery for a document set is a query that retrieves these documents in its top-k results [3]. We use keyqueries when at least 128 documents were judged as relevant in the previous iterations (otherwise we proceed as the baseline does). From the at least 128 relevant documents, we randomly choose 128 documents and compute the pairwise $tf \cdot idf$-weighted cosine similarities for all document pairs. We select the four documents with the highest sum of their six pairwise cosine similarities, ignoring very low similarities below 0.2. Such four documents are assumed to "represent" a specific topic covered in the previous result lists (the previous random selection should ensure that changing the topic in foucs in some later iteration should be possible). In case that four documents could be identified, we extract the top-10 head noun phrases [1] from their concatenated main contents [9] (if no four documents are found due to the 0.2 lower bound on the similarity we proceed with a baseline iteration). The keyphrases are used to form a keyquery for as many of the relevant documents as possible against the Lucene BM25 index as the reference search engine with k set to 32. The top-128 results from the keyquery 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).

Stopping Criterion. We stop submitting results when the following empirically determined inequality is satisfied:

$$1.5 \cdot |D_{\text{relevant}}| + 1500 < |D_{\text{irrelevant}}| \quad .$$

Where $|D_{\text{X}}|$ is the number of documents that have been submitted so far and were judged as relevant/irrelevant.

Runs

We submitted two runs, the first using the baseline iterations and the second using the iterations with keyqueries.

3. OPEN SEARCH TRACK

The objective of the Open Search track is to rank a small set of candidate documents (e.g., papers) in return to a scholarly search query.

Our approach works as follows. We first rank the candidates with Lucene's BM25 implementation. This ranking is then run through our axiomatic re-ranking pipeline [7] with the axiom weights trained for BM25. We omitted the axioms from our pipeline that are not suited for the scholarly search setup due to missing information (e.g., we omitted the PageRank axiom due to the non-availability of a full citation graph at the time of submission).

The re-ranking obtained from the combined axioms then forms the submitted ranking similar to our runs for the Web track 2014 and the Session tracks 2014–2015 [6, 5].

4. REFERENCES

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

[2] P. N. Bennett and R. W. White. Mining tasks from the web anchor text graph: MSR notebook paper for the TREC 2015 tasks track. In *Proceedings of TREC 2015*.

[3] 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.

[4] 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.

[5] M. Hagen, S. Göring, M. Keil, O. Anifowose, A. Othman, and B. Stein. Webis at TREC 2015: Tasks and Total Recall tracks. In *Proceedings of TREC 2015*.

[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] M. Hagen, M. Völske, S. Göring, and B. Stein. Axiomatic Result Re-Ranking. In *Proceedings of CIKM 2016*, pages 721–730.

[8] D. Hiemstra and C. Hauff. MapReduce for information retrieval evaluation: "Let's quickly test this on 12 TB of data". In *Proceedings of CLEF 2010*, pages 64–69.

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

[10] Lemur Project. Indri. `http://www.lemurproject.org/indri.php`, 2016.

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

[12] 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.

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

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

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