htw saar @ TREC 2018 News Track

Agra Bimantara, Michelle Blau, Kevin Engelhardt, Johannes Gerwert, Tobias Gottschalk, Philipp Lukosz, Shenna Piri, Nima Saken Shaft, Klaus Berberich

htw saar, Goebenstrasse 40, 66117 Saarbruecken, Germany
{klaus.berberich}@htwsaar

Abstract. This paper describes our participation in the background linking task of the TREC 2018 News Track. We explored four different methods to address the task. All of our methods largely rely on off-the-shelf open-source components (e.g., Apache Lucene for indexing the documents). The methods differ in how they analyze the given input document to obtain a query (e.g., by keyword extraction or named entity recognition) and to what extent the returned results are re-ranked taking meta data of the documents (e.g., publication dates) into account.

1 Introduction

The News Track is a new track first run in TREC 2018. It considers two tasks, namely background linking and entity retrieval. While the latter focuses on ranking named entities mentioned in a given document, the former starts from a document and seeks to retrieve other documents published earlier that can provide background information to the user to help her understanding. The two tasks are tested on a collection consisting of more than $590,000$ news articles published by The Washington Post.

Saarbrücken University of Applied Sciences (htw saar) participated in the background linking task of the TREC 2018 News Track. In this paper, we describe our approach to address this task. The proposed methods were designed and implemented by two teams of undergraduate students, who had little prior knowledge about Information Retrieval, as parts of a semester project and thesis work, respectively. We tested and submitted runs for four different methods, which were implemented largely relying on off-the-shelf open-source components such as Apache Lucene for indexing the document collection.

In the rest of the paper, we provide details on how the provided document collection was preprocessed in Section 2. Details about our methods are described in Section 3. Following that, in Section 4 we discuss the results as reported back by TREC. Finally, we draw conclusions in Section 5.

2 Dataset and Preprocessing

In this section, we describe the document collection provided by TREC and preprocessing steps that we applied to it prior to indexing.

The document collection provided by TREC consists of more than 590,000 news articles published by The Washington Post either in print or as blog posts. After its initial release, the document collection was revised to remove documents having the same identifier. Even after this revision, we observed duplicate documents, with different identifiers but identical content, in our results. As discussed below, our methods eliminate such duplicates either in a preprocessing step, when indexing the document collection, or in a result filtering step, i.e. post retrieval.

3 Methods

We next describe the four methods that produced our submitted runs (`htwsaar1-4`).

3.1 Method 1 (`htwsaar1`)

Methods 1 and 2 rely on Elasticsearch [1] as an indexing platforms. The revised document collection provided by TREC was indexed, treating all fields from the JSON documents separately. Elasticsearch builds on-top of Lucene and we used the default settings for tokenization and normalization, i.e., words were lowercased but not stemmed. Also for the retrieval model, we rely on the default setting, which is a variation of the vector space model [3].

Our first method analyzes the input document by performing a simple keyword extraction. More precisely, terms contained in the input document are ranked according to their value

$$tf(v,d) \cdot \log\left(1 + \frac{|D|}{df(v)}\right)$$

with $tf(v,d)$ as the frequency of term v in the input document d, $|D|$ as the size of the document collection, and $df(v)$ as the document frequency of term v. The twenty terms achieving the highest value are selected and used as a query. In addition, a date filter is applied to allow only documents published on or before the publication date of the input document.

The result returned by Elasticsearch are re-ranked taking their publication dates into account. The score attached to every result document is modified by a factor of

$$e^{-\lambda(|t_q - t|)}$$

with t_q as the publication date of the input document and t as the publication date of the considered result document. Here, publication dates are represented as UNIX timestamps in milliseconds and $\lambda = 10^{-12}$ was employed as a parameter setting. As an additional post-filtering step, duplicate documents having the same title, the same authors, and published on the same date are removed. Moreover, based on the task instructions, all documents published in the sections "Opinion", "Letters to the Editor", and "The Post's View" are also filtered out.

3.2 Method 2 (`htwsaar2`)

Our second method operating on the index built using Elasticsearch relies on named entity recognition to construct a query for the given input document. Stanford CoreNLP [5] is used to analyze the title of the document and spot mentions of named entities. Terms belonging to such a mentioned of a named entity are concatenated to yield the query that is issued. If the title of the document does not contain any mention of a named entity, the method falls back to using the entire title as a query. The returned results undergo the same temporal re-ranking and post-filtering as described above for Method 1.

3.3 Method 3 (`htwsaar3`)

Methods 3 and 4 were implemented in a separate system, relying on Apache Lucene [2] as an indexing platform. As a pre-processing step, based on our observation that duplicate documents were still present in the collection, we applied a simple heuristic to remove suspected duplicates. More precisely, when processing input files in lexicographic order of their file name, we check for every document whether a document with the same title, written by the same author, and published on the same date has already been seen. If this is the case for a document, it is ignored and not added to our index. Moreover, based on task instructions, all documents published in the sections "Opinion", "Letters to the Editor", and "The Post's View" are also filtered out. This leaves us with a total of 480,627 documents. From the JSON documents provided by TREC we extracted the *author*, *publication date*, *URL*, *title*, and *contents* and indexed them as separate fields. For the title and content fields we relied on Lucene's default tokenization and normalization, i.e., words were lowercased but not stemmed. The other fields were indexed as-is, in the case of publication date as a numerical value to allow filtering based on date ranges.

Our first method operating on the thus created index relies on a two-step approach. In a first step, the given input document is analyzed to construct a query that can be issued against Lucene. The returned results, in a second step, are then re-ranked taking into account meta data of the input documents. In the following, we describe these two steps in more detail.

Query Construction by Document Analysis Not all words contained in an input document are equally useful as query terms – at least that was our initial intuition. To identify salient words contained in the input document, more specifically its *title* and *content*, our third method relies on key-phrase extraction and named entity recognition. We make use of TextRank [4] to identify key phrases contained in the title or content of the input document, relying on the implementation[1] provided by one of the original authors. Likewise, to spot mentions of named entities (specifically persons, organization, and locations), we again use Stanford CoreNLP [5] with its default models for English.

[1] https://github.com/boudinfl/pke

Having extracted key phrases and named entities, we next determine how often each of them is mentioned in the input document and rank them accordingly. Thus we obtain four ranked lists of key phrases, persons, organizations, and locations in descending order of their frequency in the document. Let $\langle m_1, \ldots, m_n \rangle$ denote one such ranked list, we boost the terms contained in the mention m_i at rank i as

$$\max(1.1, C_T - 0.1 \cdot i)$$

with C_T as a constant that is chosen for each of the four types of mentions that we consider. For key phrases $C_K = 3.0$, for persons $C_P = 3.0$, for organizations $C_O = 2.1$, and locations $C_L = 1.8$. These choices reflect our initial expectation regarding the importance of the different types of mentions. To illustrate this with an example, assume that michael jordan was found to be the second most frequent named entity mentioned in the input document. In this case, the terms michael and jordan will be assigned a boost of 2.8. If a term is found in different types of mentions in the document (e.g., as a person and in a key phrase), the largest boost determined is used. Boosting query terms in Lucene effectively inflates their term frequency in any document by a factor corresponding to the specified boost. As a result, these terms end up having a stronger influence on the result, which is determined using a variation of the vector space model [3] in Lucene. The query that is issued against Lucene then consists of all terms that we found in any key phrase or named entity boosted as described above. In addition, a date filter is applied, allowing only such documents into the result that were published on or before the publication date of the input document.

Re-Ranking based on Meta Data Once result documents have been obtained for the query constructed in the first step, we re-rank them taking into account their author and publication date. For the publication date, we reckon that documents published at a time closer to the publication date of the input document are preferable. Thus, we determine how many weeks w a document has been published before the input document and boost its score as returned by Lucene by a factor of

$$1 + \frac{0.7}{\sqrt{1+w}} \,.$$

A document published in the same week as the input article thus obtains a boost of 1.7, whereas a document published three weeks before the input article only obtains a boost of 1.35. As a second criterion for re-ranking we check whether a document was written by the same author as the input document. If this is the case, its score is boosted by a factor of 1.2 and remains unaltered otherwise. The idea here is that authors tend to specialize on certain topics or events, so that documents by the same author should be more likely to be related.

3.4 Method 4 (`htwsaar4`)

Our last method relies on the same approach for indexing documents and removing duplicates as Method 2. In contrast, it uses a much simpler approach

to derive a query from the input document and does not apply any re-ranking based on content or meta data. It simply forms a verbose query by concatenating the title and the body of the input document and issues it against Lucene. Again, a date filter is applied to allow only documents published on or before the publication date of the input document. This was the first method developed and served as a baseline for internal testing. Quite surprisingly, it turned out to be the best performing among our methods.

4 Results

We now discuss our findings regarding the performance of our four methods based on the results provided by TREC. Since, at the time of writing, runs for other participants are not yet available, we compare our methods against a hypothetical competitor (`median`) that achieves median performance for all topics. For our discussion, we focus on nDCG@5 as a primary effectiveness measure, in line with the task description.

Table 1. nDCG@5 for a hypothetical competitor that achieves median performance for all topics (`median`) and our four runs (`htwsaar1-4`)

	`median`	`htwsaar1`	`htwsaar2`	`htwsaar3`	`htwsaar4`
nDCG@5	0.3448	0.4150	0.1957	0.4609	0.4619

Overall Performance Table 1 lists nDCG@5 across all topics. As can be seen from the figures, our second method `htwsaar2`, which relies on extracting named entities from the title of the input document, can not outperform the hypothetical competitor. This can be explained by the fact that, if the title contains only few named entities, the method ends up generating a very short query, arguably loosing the gist of the input document. All of our other methods (i.e., `htwsaar1`, `htwsaar3`, and `htwsaar4`) do outperform the hypothetical competitor by a reasonable margin. Among them, quite surprisingly, the simplest method `htwsaar4`, which uses the entire input document as a query, achieves the best performance. The two methods `htwsaar1` and `htwsaar3` relying on an analysis of the input document and re-ranking based on meta data follow closely. Our take home from these results is that verbose queries, retaining many or all of the terms from the input document, seem favorable for the task. Whether our simple baseline (`htwsaar4`) with its very verbose queries would be viable in a production setting is questionable. The two methods (`htwsaar1` and `htwsaar3`) conducting an analysis of the entire document, yielding shorter queries, may be more appropriate, even though they sacrifice a bit of retrieval effectiveness.

Table 2. Topics for which our methods achieved the highest gain (+) or loss (-) relative to the hypothetical competitor.

`htwsaar1`

Gain/Loss	Topic	Title of Input Document
+	811	'Car hacking' just got real: In experiment, hackers disable SUV on busy highway
-	433	Defining cool, from Walt Whitman and James Dean to Steve Jobs and Tony Hawk

`htwsaar2`

Gain/Loss	Topic	Title of Input Document
+	378	Germans would rather send humanitarian aid than forgive Greece's debts
-	805	Apparently, the once-feared snakehead is just another fish in the Potomac

`htwsaar3`

Gain/Loss	Topic	Title of Input Document
+	646	The astonishing state-by-state rise in food stamp reliance
-	341	How major U.S. cities and transit systems are reacting to the Brussels attacks

`htwsaar4`

Gain/Loss	Topic	Title of Input Document
+	811	'Car hacking' just got real: In experiment, hackers disable SUV on busy highway
-	818	Eggs are okay again

Gains and Losses To shed some light on in which cases our methods work and in which cases they don't, we conduct a gains and losses analysis. For each of our four methods, we identify the topic for which it achieves the highest absolute gain or loss in terms of nDCG@5 relative to the hypothetical competitor. Table 2 lists the identified topics.

Interestingly, two of our methods (`htwsaar1` and `htwsaar4`) achieve their biggest gains for the same topic 811. Inspecting the document, we observe that it mentions a fair amount of named entities and technical terms, which can be picked up by our methods when constructing the query. Our method `htwsaar2` achieves its biggest gain for the topic 378, for which it manages to recognize the two key named entities Germans and Greece from the title. Topic 646, for which our method `htwsaar3` achieves its biggest gain, mentions a fair amount of locations and our method is able to pick up key phrases such as food stamps and SNAP benefits.

Looking at biggest losses, we observe that our methods suffer them for different topics. Topic 455, which our method `htwsaar1` looses on, corresponds to a relatively short document, which makes it harder for the method to extract meaningful key words. Our method `htwsaar2` looses on topic 805, a document for which only Potomac is recognized as a named entity in the title, resulting in a very unspecific query. Topic 341 is the one for which our method `htwsaar3` suffers its biggest loss. Inspecting the document, we observe that many of the named entities therein are related to the U.S., shifting the focus of the generated query away from the terror attacks in Brussels. Finally, our method `htwsaar4` looses on topic 818. The document about dietary recommendations related to cholesterol contains many general terms, leading to a diluted query.

5 Conclusion

In this paper, we described our methods that participated in the TREC 2018 News Track. Based on the results available at this time, we conducted an initial analysis. Interestingly, the simplest method that we considered achieved the best results in terms of nDCG@5 among our methods. More sophisticated methods, relying on an analysis of the input document, got close in terms of nDCG@5 and may be more viable in practice than using the entire input document as a query. We also observed that relying only on the title of the input document is insufficient, our method based on this strategy performed worst among our methods.

References

1. **Elasticsearch**,
   https://www.elastic.co/products/elasticsearch (last accessed: 2018/10/26)
2. **Apache Lucene**,
   http://lucene.apache.org (last accessed: 2018/10/26)
3. **Apache Lucene - Scoring**,
   https://lucene.apache.org/core/3_5_0/scoring.html (last accessed: 2018/10/26)
4. Bougouin, A., Boudin, F., Daille, B.: Topicrank: Graph-based topic ranking for keyphrase extraction. In: Sixth International Joint Conference on Natural Language Processing, IJCNLP 2013, Nagoya, Japan, October 14-18, 2013. pp. 543–551. Asian Federation of Natural Language Processing / ACL (2013), http://aclweb.org/anthology/I/I13/I13-1062.pdf
5. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: Association for Computational Linguistics (ACL) System Demonstrations. pp. 55–60 (2014), http://www.aclweb.org/anthology/P/P14/P14-5010