Access to legal documents: Exact match, best match, and combinations

Arampatzis, A.; Kamps, J.; Koolen, M.; Nussbaum, N.

Publication date
2008

Published in
The Sixteenth Text REtrieval Conference Proceedings (TREC 2007)

Access to Legal Documents: Exact Match, Best Match, and Combinations

Avi Arampatzis[1] **Jaap Kamps**[1,2] **Marijn Koolen**[1] **Nir Nussbaum**[2]

[1] Archives and Information Studies, Faculty of Humanities, University of Amsterdam
[2] ISLA, Informatics Institute, University of Amsterdam

Abstract: In this paper, we document our efforts in participating to the TREC 2007 Legal track. We had multiple aims: First, to experiment with using different query formulations, trying to exploit the verbose topic statements. Second, to analyse how ranked retrieval methods can be fruitfully combined with traditional Boolean queries. Our main findings can be summarized as follows: First, we got mixed results trying to combine the original search request with terms extracted from the verbose topic statement. Second, by combining the Boolean reference run wit our ranked retrieval run allows us to get the high recall of the Boolean retrieval, whilst precision scores show an improvement over both the Boolean *and* the ranked retrieval runs. Third, we found out that if we treat the Boolean query as free text with varying degrees of interpretation of the original operator, we get competitive results. Moreover, both types of queries seem to capture different relevant documents, and the combination between the request text and the Boolean query leads to substantial gain in precision and recall.

1 Introduction

In our first participation in the Legal track, we experimented with different query formulations and run combinations. Since the focus in the Legal track ad hoc evaluation is on recall oriented measures, we investigated methods to increase recall by combining result lists based on different query representations. We also analyzed differences between our ranked retrieval runs and the Boolean reference run, and investigated ways to fruitfully combine the strengths of both approaches. Finally, we also checked different methods of exploiting the Boolean topic statements and combinations thereof with the request text.

The rest of this paper is organized as follows. In Section 2, we describe the experimental set-up. In Section 4, we discuss our official submission, results, and additional experiments. Finally, we summarize our findings in Section 5.

2 Experimental Set-up

Our retrieval system is based on the LUCENE engine version 1.9 [4].

2.1 Index

the Legal track uses a test collection, containing 6,910,192 documents (∼58Gb uncompressed). The documents are from the legal domain, on issues of the tobacco industry. The documents are all in XML format, containing limited meta-data.

We created two separate indexes as follows.

Full-text: the full textual content of the documents, including the meta-data tags, as is (∼37GB).

Text-only: the text inside the tags, not including the tags (∼33GB).

During indexing, we captured the document ID and stored it in the index. In tokenization, we removed the common stop-words and stemmed using the Snowball stemming algorithm [5].

The original corpus contains almost 7 million documents in 650 files and is over 58Gb uncompressed data. The file-name convention is `iitcdip.x.y.xml`, where `x` is a letter a-z (excluding s) and `y` is a letter a-z. As a first step we decided to create 25 partial indexes for each of the two indexing choices mentioned above, for example: index `a` indexed the files `iitcdip.a.a.xml` to `iitcdip.a.z.xml`. We chose to index in chunks because a single indexing process would have taken very long time. Indexing in chunks enabled us to use seven computers concurrently, each indexing different chunks. Since accessing multiple indexes is substantially slower, we eventually merged the 25 indexes into one index, an action that LUCENE is handling straightforwardly.

2.2 Retrieval Model

For ranking, we used a vector-space retrieval model. Our vector space model is the default similarity measure in

LUCENE, i.e., for a collection D, document d, and query q:

$$sim(q,d) = \\ = \sum_{t \in q} \frac{tf_{t,q} \cdot idf_t}{norm_q} \cdot \frac{tf_{t,d} \cdot idf_t}{norm_d} \cdot coord_{q,d} \cdot weight_t \ ,$$

where

$$\begin{aligned} tf_{t,X} &= \sqrt{\text{freq}(t,X)} \ , \\ idf_t &= 1 + \log \frac{|D|}{\text{freq}(t,D)} \ , \\ norm_q &= \sqrt{\sum_{t \in q} tf_{t,q} \cdot idf_t^{\,2}} \ , \\ norm_d &= \sqrt{|d|} \ , \\ coord_{q,d} &= \frac{|q \cap d|}{|q|} \ . \end{aligned}$$

3 Experiments

3.1 Runs

This was our first participation in the TREC Legal Track. Some of the runs described in this paper are post-submission experiments and have not been included in the pooling process.

We made runs using the search request as stated in the *RequestedText* tag:

Full-text: runs on the *full-text* index. In this run, we used the Snowball stemming algorithm on the *RequestedText* meta-data in the query.

Text-only: similar to *full-text*, but using the *text-only* index.

The Legal Track topics have very lengthy topic descriptions providing a range of background information on the topic of request. Hence, we tried to extract potentially useful terms from them. Specifically, we decided to select only those terms that are most characteristic for a single topic, with reference to the whole topic set. That is, the terms that best distinguish the topic at hand from the other topics in the topic set. For this we used a variant of the parsimonious language modeling techniques [3], and created a query by selecting the 25 terms that are most characteristic for the topic. For example, topic 52 reads:

```
Please produce any and all documents
that discuss the use or introduction of
high-phosphate fertilizers (HPF) for the
specific purpose of boosting crop yield
in commercial agriculture.
```

and the 25 selected terms are:

```
hpf sugar mh vsf gcc valhalla candy
phosphate plaintiffs its fertilizers
crop high gladsheim beet yield community
groundwater health death use fertiliz
contamination phosphat cause
```

We used the selected terms in the following ways:

SelectedTerms: the query string fed into LUCENE is the original query (the *RequestedText* tag) appended by the most significant 25 terms in the background information supplied in the file $fullL07\_v1.xml$. Duplicate terms were removed.

CombiTextSelectedTerms: a combination of **Text-only** and **SelectedTerms**. We used the standard combination method CombSUM [2] and combined full length runs without normalizing the scores.

One of the main differences between our ranked-retrieval approach and the Boolean reference run is the query: we used the *RequestedText* field whereas for the reference run the *FinalQuery* was used. To study the effect of the different topic statements, we performed additional runs, trying to exploit the terms and operators stated in the topic's Boolean query fields. Specifically, the tags *FinalQuery*, *ProposalByDefendant* and *RejoinderByPlaintiff*, the former one representing the query agreed by the sides and the two latter ones representing the negotiation history between the defendant and the plaintiff. Our experiments showed that using *FinalQuery* gives better results than *ProposalByDefendant* and *RejoinderByPlaintiff*, and we will only discuss experiments using *FinalQuery* in this paper.

The Boolean queries, including *FinalQuery* that we use, consist of terms and operators which represent a query syntax, explained in [1]. Since this syntax does not correspond to Lucene's query syntax and is actually more expressive than Lucene's syntax, we did some translation of the queries to a Lucene-readable syntax. Unfortunately, Lucene's ability to deal with multiple wildcards is very limited in such a big index (using too many wildcards would lead to a crash with a '*too many clauses*' error message) and we had to translate them as well.

The runs are as following:

BoolTermsOnly: The terms alone are used; all the other characters, such as parentheses or proximity operators are removed.

BoolLuceneTrans: The original syntax is translated into Lucene's query language. Proximity operator (`w/k`) is translated into `AND`, wildcards symbol (`!`) are removed and `BUT NOT` is replaced by `AND NOT`.

BoolTermsWildcardExp: For this run we used the terms alone and removed all the other characters except the wildcard symbol (`!`). Wildcards are expanded to all the possible variations that appear in the complete topic text, with the OR operator between them. In this

Table 1: Statistics over judged and relevant documents per topic.

	# of	per topic				
	topics	min	max	median	mean	st.dev
judged	43	488	1,000	499	567.53	164.84
relevant	43	10	391	72	101.02	97.76
B	43	103	22,518	2,665	5,004.02	6,156.75

way, the keyword `produc!` in topic 52 was replaced by `produc OR production OR product OR products`.

BoolWildcardExpLuceneTrans: Here we expanded the wildcards in the original Boolean query as we did in the **BoolTermsWildcardExp** run and then translated the result as we did in the **BoolLuceneTrans** run.

We were especially interested in combinations of both types: the request text and the Boolean query text, in various interpretations. We combined the **TextOnly** run with each of the four Boolean runs, using the CombSUM method, creating **CombiTextTerms**, **CombiTextLuceneTrans**, **CombiTextTermsWildcard** and **CombiTextWildcardLuceneTrans** runs.

4 Results

4.1 Topics and Judgments

The results are based on the qrels over 43 topics. Statistics of the assessments are shown in Table 1. We include the number of results of the negotiated Boolean query ("B" for short), since it plays an important role in the recall oriented measures used. It is striking that B is orders of magnitude larger than the number of known relevant documents. Moreover, there is no significant correlation between B and the number of relevant documents (Pearson r = 0.059). There is a significant correlation (0.55) between the number of judged and number of found relevant documents, which is not unexpected.

4.2 Runs

Table 2 shows the results (all scores based on `l07_eval` v1.0). First, we look at **Full-text**, and compare it to the similar **Text-only** run. The **Full-text** run scores marginally better on bpref, but the **Text-Only** run scores marginally better on all other measures including the main measure, i.e. estimated recall at B.

Second, we look at the **SelectedTerms** run. We see a drop in performance for all measures. This is not unexpected: the selected terms from the complete topic statement are less focused on the topic. Nevertheless, the run may have picked up documents that are missed by the original query or improve the ranking of retrieved relevant documents. Can we use these results to improve recall in our original run? We combine the results from the two runs **Text-only** and **SelectedTerms** using standard CombSUM, i.e. we just add the scores of the individual runs per document and re-rank. The results are mixed. For the **CombiTextSelectedTerms** run, we see a drop in MAP and Precision@10 when compared to the best individual run, but an increase in bpref. We see a minimal gain in recallB, but a loss of estimated recall at B.

4.3 Combining Ranked & Boolean Retrieval

We conducted further experiments trying to shed light on the relative strength and weaknesses of our ranked retrieval methods versus the Boolean reference run.

First, we looked at the reference Boolean run `refL07B` which has unranked set results (all selected documents have a RSV of 1). As Table 2 shows, this results indeed in very poor MAP and P10 scores. The bpref score is comparable to the scores of ranked retrieval (this is also clearly signalling that bpref should not be treated as an approximation of traditional MAP). In terms of recall, the reference run is retrieving fewer relevant documents overall (but has only B results per topic whereas our runs have up to 25,000). It has slightly better recall at B, and much better estimated recall at B. Summarizing, the reference run has unimpressive precision but very good recall.

Is there a way to combine the strength of ranked and Boolean approaches? What we did is the following. Recall that `refL07B` run assigns a score of 1 to every document. Our runs score in the range $[0, 1]$. Now, consider what happens if we combine the reference run with one of our runs: it will first have all documents of the reference run, but ranked by our retrieval score, and then have the remaining documents from our run, again ranked by our retrieval score. The run **CombiTextRef** in Table 2 combines the **Text-only** run with the `refL07B` run. What we see is that, indeed, the recall at B and estimated recall at B of the Boolean run are preserved. However, the MAP, bpref, and P10 scores even improve over the scores from the ranked retrieval run alone. Summarizing, combining Boolean and ranked retrieval allows us to get the best of both worlds: the high recall scores of the Boolean run are maintained, while the MAP and precision scores show an improvement over both the Boolean *and* the ranked retrieval run.

4.4 Boolean Queries Runs

We now look at the relative strength and weaknesses of the different query statements, by making various runs based on the Boolean *FinalQuery* field.

We first look at the four variants to derive a query from the *FinalQuery* field (as discussed above). As Table 2 shows, extracting just the keywords from the Boolean query,

Table 2: Results for the Legal Track 2007 (using `l07_eval` v1.0). Best scores are in bold-face.

Run	MAP	bpref	P10	num_rel_ret	recallB	est_RB
Full-text	0.0878	0.3266	0.2837	3,338	0.4792	0.1448
Text-only	0.0880	0.3255	0.2860	3,339	0.4835	0.1548
SelectedTerms	0.0355	0.2619	0.1070	2,522	0.3173	0.0772
CombiTextSelectedTerms	0.0846	0.3302	0.2698	3,306	0.4841	0.1447
`refL07B`	0.0167	0.2902	0.0209	2,145	0.4864	**0.2158**
CombiTextRef	0.1181	**0.3842**	0.3209	3,553	0.4864	**0.2158**
BoolTermsOnly	0.0878	0.3274	0.2535	3,016	0.4846	0.1417
BoolLuceneTrans	0.0880	0.3039	0.2535	2,200	0.4172	0.1526
BoolTermsWildcardExp	0.1021	0.3321	0.3140	3,352	0.5122	0.1335
BoolWildcardExpLuceneTrans	0.0915	0.3250	0.2488	2,758	0.4369	0.1555
CombiTextTerms	0.1220	0.3615	0.3326	3,473	**0.5673**	0.1843
CombiTextLuceneTrans	0.1191	0.3683	0.3047	3,426	0.5520	0.1908
CombiTextTermsWildcard	**0.1264**	0.3592	**0.3465**	**3,490**	0.5627	0.1644
CombiTextWildcardLuceneTrans	0.1191	0.3665	0.3256	3,431	0.5479	0.1976

BoolTermsOnly performed slightly worse than the **Text-Only** run. Adding a basic translation of the operators into Lucene syntax as in **BoolLuceneTrans** gives mixed results in terms of performance: we see a small improvement in estimated Recall at B, but bpref and known recall at B drop.

We added the wildcard expansion in order to compensate somewhat for Lucene's inability to cope with a too long query expression. In this way we could look into the variations of the terms as they appear in the entire topic statements. The whole index contains too many variations for Lucene's built-in equivalent, especially because of the often poor graphic quality of the original scanned documents, which results in many mistakes during the OCR process, creating an astounding number of unique (and often wrong) terms. A Boolean query that has a few wildcards could be expanded internally into thousands of clauses or even more. The runs that incorporate the wildcard expansion (**BoolTermsWildcardExp** and **BoolWildcardExpLuceneTrans**) did perform better in general. The general conclusion of the runs based on the Boolean query is that their performance meets and exceeds that of the keyword query based on the *RequestText* field.

Finally, we look at how complementary the runs based on the different topic fields are, by looking at their combination. As Table 2 shows, this leads to improvement throughout. The stricter interpretation of the *FinalQuery* leads to better results (since it is more different from the original *RequestText* run) and we get close to the reference run's score on estimated recall at B. The best scoring run is **CombiTextWildcardLuceneTrans** with a score of 0.1976 against 0.2158 for the reference run. Our general conclusion is that *RequestText* and *FinalQuery* complete each other nicely: combining them leads to substantial gains in precision and recall.

5 Conclusions

Our first participation in the Legal Track was driven by two main aims. First, we experimented with using different query formulations trying to exploit the verbose topic statements. Second, we analysed how ranked retrieval methods can be fruitfully combined with traditional Boolean queries. Our main findings can be summarized as follows: First, we got mixed results trying to combine the original search request with terms extracted from the verbose topic statement. Second, by combining the Boolean reference run with our ranked retrieval run allows us to get the high recall of the Boolean retrieval, whilst precision scores show an improvement over both the Boolean *and* the ranked retrieval runs. Third, we found that if we treat the Boolean query as free text with varying degrees of interpretation of the original operator, we get competitive results. Moreover, both types of queries seem to capture different relevant documents, and the combination between the request text and the Boolean query leads to substantial gain in precision and recall.

Acknowledgments This research was supported by the Netherlands Organization for Scientific Research (NWO, grant # 612.066.513, 639.072.601, and 640.001.501), and by the E.U.'s 6th FP for RTD (project MultiMATCH contract IST-033104).

References

[1] Trec legal README, 2007. http://trec-legal.umiacs.umd.edu/trec07topics/readmeL07_v1.txt.

[2] E. Fox and J. Shaw. Combination of multiple searches. In D. Harman, editor, *The Second Text REtrieval Conference (TREC-2)*, pages 243–252. National Institute for

Standards and Technology. NIST Special Publication 500-215, 1994.

[3] D. Hiemstra, S. Robertson, and H. Zaragoza. Parsimonious language models for information retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 178–185. ACM Press, New York NY, 2004.

[4] Lucene. The Lucene search engine, 2007. http://lucene.apache.org/.

[5] Snowball. Stemming algorithms for use in information retrieval, 2007. http://www.snowball.tartarus.org/.