Exploring the Legal Discovery and Enterprise Tracks at the University of Iowa

Brian Almquist,[1] Viet Ha-Thuc,[2] Aditya K. Sehgal,[2] Robert Arens[2] and Padmini Srinivasan[1,2,3]

[1]Department of Management Sciences

[2]Computer Science Department

[3]School of Library and Information Science
The University of Iowa
{brian-almquist, hathuc-viet, aditya-sehgal, robert-arens, padmini-srinivasan}@uiowa.edu

The University of Iowa Team, under coordinating professor Padmini Srinivasan, participated in the legal discovery and enterprise tracks of TREC-2007.

1. Legal Discovery Track

1.1 The System

1.1.1 Lucene

In designing our own toolset for the TREC Legal Track, we opted to use the Lucene library of indexing and search tools. Lucene, developed in Java, is highly scalable and extendable. Indexing and searching the TREC-Legal collection proved well within Lucene's capabilities.

We indexed the entire TREC collection, opting to merge the document content and the title into a single field, using the Lucene StandardAnalyzer, which strips punctuation, but recognizes and retains elements such as e-mail addresses. The StandardAnalyzer stoplist was used for indexing. For our explorations, we converted topic fields into term vectors for querying the collection. For each topic, our system returned a ranked set of results with enough documents to match in quantity either those retrieved by a reference Boolean query executed on behalf of the TREC 2006 evaluators, or enough to reach a set cap on the number of documents returned, whichever was greater.

1.1.2 Okapi reranking

The Lucene default scoring system was used to retrieve and rank an initial collection of documents. If a given topic required *n* returned documents, the 1.5*n* Lucene-collected documents were re-scored using Okapi-BM25 scores, and reranked. Our implementation of Okapi-BM25 used parameters $k_1 = 1.2$, and $B = 0.75$. Each repeated instance of a term in our query vector was treated as a unique term, which is consistent with setting the k_3 parameter to ∞.

1.1.3 Metadata

The TREC-Legal document collection includes voluminous, but inconsistently applied, document-level metadata, much of it relating to authorship, organizational source, internal distribution of the document, and physical storage of the document. We conducted runs applying the Boolean query to each of the metadata fields individually. Besides the body text ("ocr-text"), the greatest success was found querying against "title" metadata. Our reported experimental runs are executed against a single indexed field containing both the "ocr-text" and "title" metadata.

1.2 Query Expansion

1.2.1 Wildcard Expansion

Lucene handles wildcards by finding all eligible candidate terms in the queried index field, and adding them to the term vector. One of the consequences of OCR-error is that many error-distorted words become eligible wildcard substitutes. Without a mechanism in place to reweight the terms, certain wildcard expansions would overwhelm all other terms in the query vector. This also adds to the processing time for the query.

Before submitting the query vector to Lucene, our system will detect a wildcard term, then replace it with a set of candidate replacements. Candidates are selected on the basis of their document frequency. The size of the set is a parameterized value that we tested. Table 1 provides some examples of these candidate substitutions. We found that accepting two substitute terms would not limit us if the first candidate was not related to the topic. On the other hand, two replacement terms was not so large a pool that it would drown the query, or force it into accepting apparent OCR error.

Table 1. R-prec, MAP, and some sample candidate replacements for different numbers (*n*) of replacements.

Wildcard Replacements (*n*)	**0**	**1**	**2**	**3**
R-prec	0.0785	0.0893	***0.0968***	0.0916
% improvement from *n* = 0		13.76	23.31	16.69
MAP	0.0430	0.0462	***0.0554***	0.0522
% improvement from *n* = 0		7.44	28.84	21.40
Wildcard Term (examples)	**Candidate Terms**			
calif!	caliph	california	caliph	califano
ad!	ad	additional	advertising	addition
cigar!	cigar	cigarette	cigarettes	cigar

1.2.2 Blind Feedback

Blind feedback, or pseudo-relevance feedback, has been established as an effective technique for this task. As implemented in our system, we conducted a preliminary run to retrieve an initial set of top-ranked documents. From this batch, we extract terms for query expansion from a subset of the top ranked documents. The queries are re-executed, using the exact configuration as the preliminary run, except that candidate query expansion terms have been added to the query term vector.

The number of top-ranked documents analyzed, and the number of terms extracted are both parameters that can adjusted. To qualify for inclusion in the query term vector, a term has to consist entirely of letter characters, it cannot already exist in the query vector, it has to meet a minimum document frequency threshold, and it cannot consist entirely of consonants. Terms are also passed over for inclusion if they are included in the SMART list of stopwords, a much more extensive list than the one included with Lucene, or if WordNet fails to recognize it as a word.

Once a set of qualifying expansion terms have been selected, they are added to the original term vector, and the query is executed once again, the results reranked as noted above, and then evaluated.

1.3 Query Reduction

TREC Legal Topics include extensive descriptive fields, many of which are designed to simulate the language that would accompany the request for documents in the setting of a civil proceeding. The RequestText field for each topic appears to distill the request for information into one or two brief natural language sentences. For an example, we will look at TREC-Legal Topic 32, where the RequestText field is occupied by the string (Table 2).

Even though the RequestText clarifies the information need, it still contains language that may dilute a query, or cause it to "drift" from relevant documents. A brief examination suggested that much language of this type was repeated between topics, and that it would usually be concentrated at the beginning of the RequestText field. After initially experimenting with a manual procedure that essentially excised perceived non-relevant language from the beginning of the RequestText field, our automated version would remove a term from the RequestText vector if it was present in the RequestText of some amount of other documents in the collection. An example of these different variations of the RequestText field is presented in Table 2. This procedure would be executed as a preprocessing step before any of the queries would be executed.

Table 2. A Sample RequestText field and the modified versions used in our training

Full RequestText	`All documents discussing, referencing, or relating to the doctrine of "market share liability" which also relate to one or more events taking place in the State of California.`
Manually edited RequestText	`the doctrine of market share liability which also relate to one more events taking place in the State of California.`
Automatically reduced RequestText	`doctrine market share liability which also relate one more events taking place state california`

1.4 Experiments

Our initial experiments, conducted using a subset of the topics, attempted to establish boundaries for wildcard expansion. We then evaluated the top three-performing configurations, along with a run with no expansions, against the full set of queries. Table 1 lists the results, including R-Prec and Mean Average Precision (MAP) across the full set of queries. These results provided us with the wildcard expansion model that we used for all further experiments. Each term from the topic containing a wildcard operator is replaced with the two eligible candidate terms with the greatest document frequency. The only field from the topic used for these queries was the "FinalQuery," ostensibly the query that results as a negotiation between attorneys for both sides of the case in question.

After establishing the wildcard expansion parameter for our experiments, we found queries could be improved by adding the RequestText field, improved further by manually editing down this field as described in §1.3. We then further improved our results by adding the Okapi-BM25 reranking postprocessing step, as traced in Table 3**.**

Table 3. NoRT - query vector is derived only from FinalQuery. AddRT - RequestText terms added to query. ModRT - RequestText is manually edited before it is added to the query. AddOB - Rescoring and reranking added as postprocessing step. Percentages denote improvement over "NoRT"

Run	**NoRT**	**AddRT**	**ModRT**	**AddOB**
R-prec	0.0968	0.1242 / 28.31%	0.1379 / 42.46%	0.1387 / 43.29%
MAP	0.0554	0.0771 / 39.17%	0.0913 / 64.80%	0.1027 / 85.38%

Our query expansion experiments began with conservative parameter values, using three documents for term extraction, while adding five terms to the query vector. All query expansion experiments evaluated the candidate terms for sufficient discriminatory value with regards to document frequency and the remainder of the documents in the collection. Initial experiments conducted failed to reveal significant improvement by adding the various filters described in § 1.2.2. All three variations of expansion demonstrated an improvement over the initial run generating the documents. As shown in Table 4, experiments increasing the number of documents evaluated for terms, or increasing the number of terms to be added to the query demonstrated that, for these tasks, while query expansion can improve performance, it must be used in a conservative fashion.

Table 4. Query expansion results from first increasing the number of documents used and then the number of terms extracted. The baseline results are generated from a run using no query expansion terms. Percentages denote change from the baseline.

Run	Baseline	3 docs 5 terms	15 docs 5 terms	6 docs 5 terms	3 docs 25 terms	3 docs 10 terms
R-prec	**0.1387**	***0.1551*** / **11.82%**	0.1416 / 2.09%	0.1471 / 6.06%	0.1111 / −19.90 %	0.1422 / 2.52%
MAP	**0.1027**	***0.1078*** / **4.97%**	0.1041 / 1.36%	0.1051 / 2.34%	0.0811 / −21.03 %	0.1007 / −1.95%

Our last experiment involved adjusting the RequestText modification procedure. We replaced the manually edited RequestText field with its original contents, which were then subjected to our automatic procedure for filtering generic language. This generated an improvement in our mean average precision (0.1078 to 0.1106, 2.79 percent improvement), although the benefit to R-prec was minimal (0.1551 to 0.1557, 0.39 percent improvement). If the level of improvement over the manual editing was not conclusive, it would still serve as a reasonable proxy for manual editing in an automated system.

Table 5 presents the results from TREC-Legal 2007, along with the comparable runs from our training using the TREC-Legal 2006 topics. The requested reference run was used as our baseline. For all runs, query term vectors are constructed by processing source strings — for example, the RequestText for the reference run — through Lucene's StandardAnalyzer. Results from the query are rescored using Okapi-BM25, and then reranked. For each run, we have described the source for the query term vector below.

- **7REF (IowaSL07REF)**: Term vector is built from the contents of the RequestText field after it is processed by the Lucene StandardAnalyzer.
- **702 (IowaSL0702)**: The same as 7REF, except that terms from all three Boolean queries are added to the term vector. Wildcards remain unexpanded.
- **703 (IowaSL0703)**: The same as 702, except that wildcards from the Boolean queries are replaced with two eligible candidate terms, as described in § 1.2.1.
- **704 (IowaSL0704)**: The same as 703, except that we have now added blind feedback query expansion terms to the term vector as described in § 1.2.2.
- **705 (IowaSL0705)**: The same as 704, except that as a preprocessing step, the terms from the RequestText Field have been automatically reduced, as described in § 1.3.
- **706 (IowaSL0706)**: The same as 705, except that all Boolean queries, save the one representing a final query negotiated between both sides of the legal dispute, have been dropped from the query term vector source. .
- **707 (IowaSL0707)**: The same as 706, except that the RequestText field has been edited manually, as described in § 1.2.2.
- **604, 605, 606, 607**: These identify runs executed against the TREC-Legal 2006 topics using identical procedures as their equivalent 2007 runs. Runs similar to 702 and 703 were also completed, but are not presented as they only used the final negotiated Boolean query for each topic.

Table 5. Collected results for submitted runs for TREC-Legal 2007, with results from equivalent training runs.

RUN	7REF	702	703	704	705	706	707	604	605	606	607
Est_RB	0.1078	0.1341	0.1639	***0.1669***	0.1613	0.1531	0.1419	—	—	—	—
R-prec	0.1300	0.1834	0.1806	0.1733	***0.1842***	0.1761	0.1731	0.1538	0.1656	0.1557	0.1551
MAP	0.0884	0.1286	0.1304	0.1325	***0.1394***	0.1288	0.1200	0.1086	0.1126	0.1106	0.1078

The difference between each run in Table 5 represents an incremental change in our procedures. Amongst our submissions, run 704, achieves the best results for the metric designated for TREC 2007, Estimated Recall at B, where B is the number of items returned by a reference run based on a Boolean query. In fact, this run garnered the best result for this measure amongst all submitted runs identified as completed without manual processing. Adding the query expansion procedure has improved our performance here, but our performance for this metric is only boosted if we use the RequestText field in its entirety. We further conducted Wilcoxon Signed Rank Tests for each incremental change in the procedure over all topics. The improvement in performance from run 703 to run 704 (adding blind feedback expansion terms to the query) is significant at the .05 level. A more significant decline in performance is detected between runs 706 and 707 (switching from automatic filtering of the RequestText field to a manual edit), at the .025 level.

In our training, run 605 achieved the best results for R-Prec and MAP, measures which were used to differing degrees for TREC-2006. This result was duplicated in the runs for TREC-2007. Again, run 705 also retains the highest score for R-Prec for all runs from the TREC Legal Track participants that indicated a lack of any manual processing. The *y*05 runs differ from the *y*04 runs in that the text of the topic RequestText field has been automatically filtered before it was added to the query term vector. The *y*05 runs differ from the *y*06 runs in that all three Boolean queries have been used to generate the query term vector. Our Wilcoxon Signed Rank Tests, found that, for R-prec, improvement by adding an automatically filtered version of the RequestText field to the query term vector (704 to 705) was significant at the .05 level. For MAP, we found that using solely the FinalQuery from the collection of Boolean queries (705 to 706) hurt performance at the .025 significance level. Switching from the automatic filtering of the request text to the manual edit (706 to 707) caused a significant decline in performance at the .0005 level.

1.5 Discussion

Our early experiments with this task are variants of well-established techniques for information retrieval. The particular challenges of this document collection are reflected in the questions we have attempted to resolve as we seek to best utilize these procedures. Our decision to modify processing of wildcard operators is driven by the potential for OCR error to distort retrieval based on all but the most frequently valid expansions.

Likewise, terms frequently appearing in the natural language fields in the information requests might be too generic to effectively add value to the retrieval process, but such terms may also be specific enough to induce drift. Consequently, analysis, manual or automatic, would prove beneficial with regards to retrieval results derived from these fields. In particular, terms referring to the type of record requested (i.e., “document”, “memo”) are non-informative. The sources for the document collection, six tobacco companies and a not-for-profit tobacco industry “research” institute, meant that information requests for “tobacco” were also less than discriminatory.

Query expansion is an oft-explored method for improving retrieval methods. Though our initial “conservative” parameterization was designed to leave considerable room for exploration, we developed the use of “filters” to improve word quality after an initial assessment of typical query expansion terms revealed many words that appeared to refer to specific acronyms, or other arcane terms. Nevertheless, such terms, possibly referring to relevant legislation or organizations, may have been useful in improving search performance. The relative lack of improvement associated with the use of query expansion term filters was surprising.

1.6 Summary and Future Work

Query expansion still offers potential for improvements. External sources for expansion terms, i.e. WordNet synonyms, could provide useful improvements. But, other than the modifications to wildcard handling, opportunities that are specific to this task are still abundant. A technique that directly addresses the effect of OCR error in the collection on information retrieval would be useful. Attempts to mine the documents and associated meta-data for topical linkages may also bear fruit.

2 Enterprise Track

2.1 Document Preprocessing

Common document preprocessing was done for both the Document Search task and Expert Search task. Preprocessing focused on removing documents likely to be irrelevant to search tasks. We began by investigating the document headers, which contained a “Content-Type” tag. Non-text files (e.g. application, image, video) were removed from the collection. We then looked at files having disproportionately large or small file sizes. Very large files were found to contain no textual information, and seemed to be either database dumps or coordinate lists from GIS data. Similarly, very small files contained no useful information. These files were removed from the collection.

Documents were converted from web pages to text documents in order to facilitate later processing steps using **lynx**, a text-based web browser. This removed formatting and HTML tags from the documents. Email addresses were preserved for the expert search task. A copy of the corpus without the **lynx** conversion was retained for metadata indexing in the document search task.

2.2 Document Search

2.2.1 Lucene Indexing

Indexing and search was conducted using the Lucene search engine. The content of the documents were indexed using the StandardAnalyzer package, using the StandardAnalyzer list of stopwords. Another field containing the document number tag from the document header was added using the KeywordAnalyzer.

Further indexing was done on documents found to contain HTML metadata. Documents with HTML title tags had this information added as a separate index field, to be searched along with the document’s content. Furthermore, some documents contained Dublin Core (DC) metadata tags. These tags and their contents were also added as separate index fields.

2.2.2 Precision and recall search strategies

Our precision strategy uses phrase search on DC metadata fields and on content fields of documents. In (1), *T* is the topic, and the weight w_1 for metadata fields is set higher than the weight w_2 for content field to reflect the fact that information in metadata fields is more precise and condense than in document content.

$$\text{Precision\_Query} = (\text{DC metadata: “}T\text{”})\text{\^{} } w_1 \text{ OR } (\text{Content: “}T\text{”})\text{\^{} } w_2 \quad (1)$$

In recall strategy, each topic *T* is tokenized into a sequence of tokens $t_1, t_2 \ldots t_k$. then we apply Boolean search on DC metadata fields and on content fields of documents:

$$\text{Recall\_Query} = (\text{DC metadata: Q})\text{\^{} } w_1 \text{ OR } (\text{Content: Q})\text{\^{} } w_2$$

$$Q = (t_1 \text{ AND } t_2) \text{ OR } (t_1 \text{ AND } t_3) \text{ OR} \ldots \text{OR } (t_1 \text{ AND } t_k) \text{ OR} \ldots \text{OR } (t_{k-1} \text{ AND } t_k)$$

2.2.3 Relevance feedback

In our first strategy (*feeback_1*), the top three ranked documents retrieved by the recall search strategy are used as pseudo-relevant documents. In the second strategy (*feedback_2*), we use three documents given in the “*page*” field of each topic as relevant documents. Then, we remove stop words from a standard list, extract top *N* most frequent words $fb_1, fb_2 \ldots fb_N$ for each topic, and add the word to the query:

Feedback_Query = Original_Query OR

$$[\text{Content: } (fb_1 \text{ AND } fb_2 \text{ AND } fb_3) \text{ OR } (fb_1 \text{ AND } fb_2 \text{ AND } fb_4) \text{ OR} \ldots \text{OR } (fb_1 \text{ AND } fb_2 \text{ AND } fb_N) \text{ OR} \ldots \text{OR } (fb_{N-2} \text{ AND } fb_{N-1} \text{ AND } fb_N)]\text{\^{} } w_2$$

where *Original_Query* could be either *Precision_Query* or *Recall_Query* described in § 2.1.1.

2.2.4 Submitted runs

We use different combinations between search and relevance feedback strategies for submitted runs:

Run_1 = Recall search with feedback_1

Run_2= Recall search without feedback

Run_3 = Precision search with feedback_2

Run_4 = Recall search with feedback_2

2.3 Expert Search

2.3.1 Candidate identification

We developed regular expressions to identify email ids and use the Stanford NER system to identify 'PERSON' named-entities. Then we developed 14 patterns to map names to email addresses constraining the mapping to occur within the same document. The process results finally in 2,756 expert candidates, and each of them has a record with "*csiro.au*" emails along with variations of his or her names, other emails and the supporting documents in which he or she occurs.

For each candidate *c*, we use the supporting documents to build his or her profile which is a weighted vector over words *w*. The weight is estimated as follows:

$$\text{weight}(w|\, c) = \text{tf}(w)*(\log(\text{df}(w)+1)$$

where *tf(w)* is term frequency of word *w* in the supporting documents, and *df(w)* is frequency of supporting documents containing *w*.

2.3.2 Topic modeling

For each topic, the top 25 ranked documents retrieved by recall search strategy are considered as relevant documents. We exploit two strategies for topic modeling.

In the first one (*topic modeling 1*), we use a relevance-based topic model, expand each topic into a multinomial distribution over words *w*. Our relevance-based topic model is an extension of relevance-based language models in which each document in the relevance set C_t of topic *t* is assumed to be generated by 2 topics: *t* and t_{bo} where *t_bo* covers the combined notion of a "background" topic and any topic other than *t*.

The contributions of topic *t* and *t_bo* to each document *d* are automatically determined by inter-document statistics in C_t and the whole corpus *C* by an unsupervised machine learning technique. In general, the tokens frequently appearing in C_t but not frequently appearing in the *C* are likely generated by the topic *t*. On the other hand, the tokens often occurring in every document in corpus *C* (background token) and the tokens only occurring in a few documents in C_t (belonging to some other topic) are likely generated by the topic *t_bo*. Finally, distribution of topic *t* is computed by the contributions of *t* in relevant documents. That makes the model more appropriate and robust than relevance-based language models.

In the second one (*topic modeling 2*), we expand each topic into a weighted vector over words *w* as in the way we apply for building candidate profiles:

$$\text{weight}(w|\, t) = \text{tf}(w)*(\log(\text{df}(w)+1)$$

2.3.3 Submitted runs

Again, we try several different combinations for our submitted runs. In the first run, we apply the relevance-based topic model (*topic modeling 1)*, and compute the posterior probably for each candidate *c* given a topic *t* as follows:

$$p(c|\, t) = \sum_w p(c|w)*p(w|t)$$

$$= \sum_w [(p(w|c)*p(c)/p(w)]*p(w|t)$$

where *p(w|t)* is provided by the relevance-based topic model. The probability *p(w|c)* is estimated by normalizing the word weights from the weighted vector (profile) of candidate *c* (Section 2.1).The probability *p(w)* is approximated by frequency of word *w* in a collection of 10000 documents randomly selected in the whole corpus. And the prior probability *p(c)* is estimated by the number of times emails or names of candidates *c* are mentioned.

In the second run, we apply *topic modeling 2*. Given each topic, we rank the candidates by the cosine similarity between the weighted vector of the candidates and the weighted vector of the topic.