UMass at TREC 2004: Notebook

Nasreen Abdul-Jaleel, James Allan, W. Bruce Croft, Fernando Diaz, Leah Larkey, Xiaoyan Li, Donald Metzler, Mark D. Smucker, Trevor Strohman, Howard Turtle, Courtney Wade

Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts
Amherst, MA 01003

1 Terabyte

1.1 Model

The retrieval model implemented in the Indri search engine is an enhanced version of the model described in [30], which combines the language modeling [35] and inference network [38] approaches to information retrieval. The resulting model allows structured queries similar to those used in INQUERY [4] to be evaluated using language modeling estimates within the network, rather than *tf.idf* estimates. Figure 1.1 shows a graphical model representation of the network. As in the original inference network framework, documents are ranked according to $P(I|D, \alpha, \beta)$, the belief the information need I is met given document D and hyperparameters α and β as evidence.

Due to space limitations, a general understanding of the inference network framework is assumed. See [30] and [38] to fill in any missing details.

1.1.1 Document Representation

Typically, in the language modeling framework, a document is represented as a sequence of tokens (terms). Based on this sequence, a multinomial language model over the vocabulary is estimated. However, it is often the case that we wish to model more interesting text phenomenon, such as phrases, the *absence* of a term, etc. Here, we represent documents as multisets of binary feature vectors. The features can be nearly any interesting binary observation of the underlying text. The features used to represent documents in our model are discussed later.

We assume that there is a single feature vector for each position within a document, although in general this need not be the case. Such a model moves away from modeling text towards modeling features of text. Throughout the remainder of this paper we refer to such models as language models, although they really are better described as *language feature models*.

1.1.2 Language Models

Since our event space is now binary we can no longer estimate a single multinomial language model for each document. Instead, we estimate a multiple-Bernoulli model for each document, as in Model B of [31]. This overcomes the theoretical issues encountered in [30]. Note that the multiple-Bernoulli model imposes the assumption that the features (r_i's) are independent, which of course may be a poor assumption depending on the feature set.

We take a Bayesian approach and impose a multiple-Beta prior over the model (θ). Thus, $P(D|\theta) \sim MultiBernoulli(\theta)$ and $P(\theta|\alpha, \beta) \sim MultiBeta(\alpha, \beta)$. Our belief at node θ is then:

$$\begin{aligned} P(\theta_i|D,\alpha,\beta) &= \frac{P(D|\theta_i)P(\theta_i|\alpha_i,\beta_i)}{\int_{\theta_i} P(D|\theta_i)P(\theta_i|\alpha_i,\beta_i)} \\ &= Beta(\#(r_i,D)+\alpha_i, |D| - \#(r_i,D)+\beta_i) \end{aligned}$$

for each i where $\#(r_i, D)$ is the number of times feature r_i is set to 1 in document D's multiset of feature vectors.

We estimate such a model for the entire document. Additionally, we estimate field specific models for a number of HTML fields. To do so, we treat all of the text in a document that appears within a given field as a pseudo-document. For example, a model can be estimated for all of the text that appears within the `h1` tags of a document. More details of the specific fields we explored are given in Subsection 1.4.

1.1.3 Representation Nodes

The r_i nodes correspond to document features that can be represented in an Indri structured query. Indri implements exactly those operators available in INQUERY [4]. They are single terms, #N (ordered window N), and #uwN (unordered window N). See [30] for more details. The belief at a given representation node is computed as:

$$\begin{aligned} P(r_i|D,\alpha,\beta) &= \int_{\theta_i} P(r_i|\theta_i)P(\theta_i|D,\alpha_i,\beta_i) \\ &= E[\theta_i] \\ &= \frac{\#(r_i,D)+\alpha_i}{|D|+\alpha_i+\beta_i} \end{aligned}$$

Furthermore, selecting $\alpha_i = \mu P(r_i|C)$ and $\beta_i = \mu(1 - P(r_i|C))$ we get the multiple-Bernoulli model equivalent of the multinomial model's Dirichlet smoothing [42] estimate:

$$P(r_i|D,\alpha,\beta) = \frac{\#(r_i,D)+\mu P(r_i|C)}{|D|+\mu}$$

where μ acts as a tunable smoothing parameter.

1.1.4 Query Nodes

The query node operators are soft probabilistic operators. All of the query operators available in INQUERY are also available in Indri, with the addition of a weighted version of the #and operator named #wand. The operators are #combine (same as #and), #weight (same as #wand), #or, #not, #sum, #wsum, and #max. See [30] for the details of how beliefs are computed at the query nodes.

Since we are using language modeling probabilities within the network, the #wsum operator no longer makes sense and the the #combine (#and) and #weight (#wand) operators are more appropriate [30]. In fact, it can be shown that the Indri query #combine($q_1 \dots q_N$) using the estimates just described returns exactly the same ranked list as the query $q_1 \dots q_N$ using the traditional (multinomial with Dirichlet smoothing) query likelihood model.

1.2 Test Platform

We ran our index builds and our queries in parallel on a cluster of 6 identically configured machines (the machine configuration is shown in Figure 2).

For the run involving anchor text, we ran an application on a single machine that extracted all anchor text from the collection. We discarded all in-site links, that is, all machines that pointed to pages on the same machine as they originated from. The remaining link text was associated with the destination page of each link. This process took more time than indexing did, and it generated approximately 7GB of anchor text.

1.3 Indri Retrieval Engine

We indexed the GOV2 collection using Indri, a new language modeling retrieval engine developed at UMass based on the Lemur project. The Indri engine was written to handle question answering and web retrieval tasks against large corpora. The engine is written in C++ and runs on Linux, Windows, Solaris and Mac OS X. We used the Terabyte task as a proving ground for this engine.

The indexing algorithm is quite similar to the one described in [14], although it was developed before we had seen this paper. In the early phases of development we had attempted to store the vocabulary of the collection in a B-Tree data structure. With caching, this method seemed to do well on collections of up to 10GB, but after that point performance degraded dramatically. We eventually changed the system to flush the vocabulary during posting flushes, as in [14]. This technique dramatically improved our indexing times.

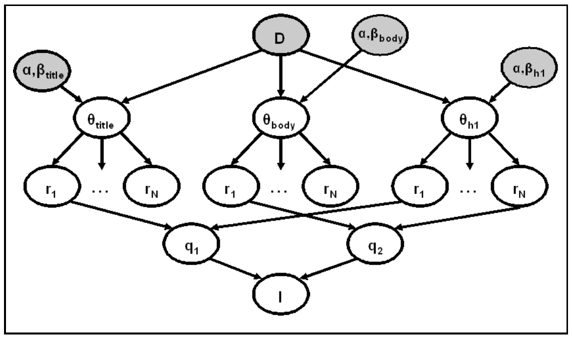

Figure 1: Indri's inference network retrieval model.

CPU	Intel Pentium 4 2.6GHz × 1
Bus speed	800MHz
OS	Linux 2.4.20 (Red Hat 9)
Memory	2GB
Boot volume	Western Digital 40G (WD400EB-75CPF0)
Work volume	Western Digital 250GB × 3 (WD2500JB-00EVA0) RAID 0 Average write seek: 10.9ms Average read seek: 8.9ms Rotational speed: 7200rpm
Network	1Gb/s Ethernet (Intel 82540EM)

Figure 2: Machine configuration for Terabyte task (6 of these machines were used) at a total cost of $9000 USD.

The Indri indexing process creates a variety of data structures:

- A compressed inverted file for the corpus, including term position information
- Compressed inverted extent lists for each field indexed in the corpus
- A vector representation of each document, including term position information and field position information
- A random-access compressed version of the corpus text

We found that flushing the vocabulary during posting flushes complicated the creation of document vectors. These vectors are compressed arrays of numbers, where each number corresponds to some term in the collection. In initial development, each term was assigned a fixed number when it was first seen in the corpus. With vocabulary flushing, we were no longer able to keep fixed term numbers throughout the indexing process. We therefore write the document vectors out using temporary term numbers. Once each term is assigned a final term number during the final inverted list merge, we rewrite the document vectors, exchanging the temporary term numbers for the final numbers.

1.3.1 Anchor Text

For the anchor text run, we followed the model presented by Ogilvie and Callan in [34]. In this model, different portions of the document are considered to be different representations of the same document. We used the heading fields (`h1`, `h2`, `h3` and `h4`), the `title`, the whole text, and the anchor text of each web page as these different representations. The Indri query language enabled us to weight these different representations during retrieval.

Let the set of pages in the corpus be C. For each document $d \in C$, there is a (possibly empty) set of documents in C that have links to d. Let this set of documents be L. We can partition L into two sets, L_I and L_E, where L_I consists of those documents on the same server as d, and L_E consists of those documents on other servers. Let $A(d)$ represent the text in links from L_E to d. We use $A(d)$ as the anchor text model for d.

In order to be able to use the anchor text model of a document during retrieval, we created an anchor text harvesting program. This anchor text harvester wrote out all the links in the collection into a separate anchor text only corpus. The program then associated the anchor text for each link to the destination document. Approximately 70% of the time for this process was taken in parsing the corpus to find the text; the remaining 30% was taken in associating the link text with the destination documents.

1.3.2 RETRIEVAL

Indri uses a document-distributed retrieval model when operating on a cluster. Each machine in the cluster runs a process called a query server, which can perform queries against its local collection. A process called the query director sends queries out to the query servers for processing, and then merges the results.

In order to generate scores on a cluster that are identical to those in a single collection, it is necessary for each cluster node to use the same corpus statistics for term frequency and corpus size. To do this, Indri starts every query by collecting statistics from the query servers. The query director combines these statistics to create a collection model for the query. These statistics are then sent back to the query servers with the query itself for final processing. This two-phase process allows Indri to handle statistics collection for phrases in the same way that it handles collection for terms.

On each query server, Indri scores documents in a document-at-a-time manner. The literature suggests that this is not the fastest way to perform retrieval, but we have found the document-at-a-time method to be more straightforward for field restricted query evaluation, and have therefore adopted it for all queries.

The Indri engine does employ a variety of optimizations at query time. Some of them were in place before we submitted our TREC results, but some were implemented afterward. We are including a second set of query times, run on the same hardware as the first, to show improvements achieved with newer optimizations.

In the results we reported at the deadline, our system used 1MB buffers on all term inverted lists in order to keep disk seek overhead to an acceptable minimum. We processed all queries into directed acyclic graphs before evaluation, which dramatically cut down on the time necessary to evaluate the more complicated adaptive window runs. We also incorporated a fast path for frequency-only terms; that is, terms that can be scored based on their frequency within the document, and without position information. For these frequency terms, we read postings from the inverted list in batches, and did not decompress the position information.

Since the deadline, we have added two more optimizations. The first is max_score, described in [39]. This optimization was in place before the deadline, but because of bugs in the implementation was not actually working. The max_score optimization allows Indri to skip inverted list postings that we know cannot be associated with a document in the top n positions of the ranked list. Note that this optimization is rank and score safe.

We have also implemented `#weight` operator folding. This optimization removes unnecessary `#weight` and `#combine` operators. For instance, a query such as `#weight( 0.5 #combine( Bruce Croft ) 0.5 #combine( James Allan ) )` is equivalent to `#weight( 0.25 Bruce 0.25 Croft 0.25 James 0.25 Allan )`. However, our implementation of max_score operates only on the top level `#weight` operator. As such, `#weight` folding, in concert with max_score, gave us a large speedup in the query expansion runs.

1.4 Runs

Five official runs were submitted for evaluation. We created runs that vary from very simple (indri04QL) to very complex (indri04FAW) with the aim of evaluating the efficiency and effectiveness of our system across a wide range of query types. In order to emulate reality as close as possible, all queries were automatically constructed using only the title field from the topic. The runs submitted were:

indri04QL – Query likelihood. For each topic we create an Indri query of the form `#combine(` $q_1 \ldots q_N$ `)`, where $Q = q_1, \ldots, q_N$ is the title portion of the topic.

indri04QLRM – Query likelihood + pseudo relevance feedback. For each topic, we construct a relevance model [25] from the top 10 documents retrieved using the indri04QL query. The original query is then augmented with the 15 terms with the highest likelihood from the relevance model. The final form of the Indri query is `#weight(`0.5 (`#combine(` $q_1 \ldots q_N$) 0.5 `#combine(` $e_1 \ldots e_{15}$)), where $e_1 \ldots e_{15}$ are the expansion terms.

indri04AW – Phrase expansion. This run explores how we can exploit Indri's proximity operators to improve effectiveness. We base our technique on the following assumption described in [5]: *query terms are likely to appear in close proximity to each other within relevant documents.*

For example, given the query "`Green party political views`" (topic 704), relevant documents will likely contain the phrases *Green party* and *political views* within relatively close proximity to one another. Most retrieval models ignore proximity constraints and allow query terms to appear anywhere within a document, even if the words are clearly unrelated. Let us treat a query as a set of terms Q and define $S_Q = \mathcal{P}(Q) \setminus \{\emptyset\}$ (i.e. the set of all non-empty subsets of Q). Then, our queries attempt to capture certain innate dependencies between query terms via the following assumptions on S_Q:

1. Every $s \in S_Q$ that consists of contiguous query terms is likely to appear as an exact phrase in a relevant document (i.e. #1)
2. Every $s \in S_Q$ such that $|s| > 1$ is likely to appear (ordered or unordered) within a reasonably sized window of text in a relevant document (i.e. #uw4$|s|$).

These assumptions state that (1) exact phrases that appear in a query are likely to appear as exact phrases within relevant documents and that (2) all query terms are likely to appear within close (ordered or unordered) proximity to each other in a relevant document. As a concrete example, given the query "`Prostate cancer treatments`" (topic 710) our system generates the following query:

```
#weight(
1.5 #combine( prostate cancer treatments )
0.1 #combine( #1( cancer treatments )
              #1( prostate cancer )
              #1( prostate cancer treatments ) )
0.3 #combine( #uw8( cancer treatments )
              #uw8( prostate treatments )
              #uw8( prostate cancer )
              #uw12( prostate cancer treatments ) )
)
```

Queries constructed in this way boost the score of documents that adhere to our assumptions. Experiments on the WT10g collection with queries of this form performed significantly better than traditional query likelihood queries.

indri04AWRM – Phrase expansion + pseudo relevance feedback. This run uses the query constructed from the indri04AW run for pseudo relevance feedback. Here, 5 documents were used to construct the relevance model and 10 expansion terms were added to the query. For this run, we weighted the original query 0.7 and the expansion terms 0.3 to yield a query of the form `#weight(` 0.7 Q_{orig} 0.3 `#combine(` $e_1 \ldots e_{10}$)), where Q_{orig} is the original query and $e_1 \ldots e_{10}$ are the expansion terms.

indri04FAW – Phrase expansion + document structure. Our final run is a largely untested and purely experimental attempt to make use of anchor text and document structure. As discussed earlier, the Indri search engine can index fields and can evaluated complex queries containing certain field constructs. Several past studies have found that anchor text and document structure yields inconsistent improvements in effectiveness for ad hoc web retrieval [12, 13, 17]. The results were obtained using the WT10g collection, which is roughly 2.5% the size of the GOV2 corpus. Therefore, we wish to explore whether these results hold for this larger collection.

The queries constructed for this run make use of main body text, anchor text, the `title` field, and header fields (`h1`, `h2`, `h3`, `h4`). The queries constructed are of the form `#weight(` 0.15 Q_{inlink} 0.25 Q_{title} 0.10 $Q_{heading}$ 0.50 $Q_{mainbody}$), where each Q_{field} is a phrase expansion query evaluated using the respective field language model. For example, Q_{inlink} is the phrase expansion query evaluated using a language model built from all of the anchor text associated with a page.

All smoothing parameters, weights, and window sizes were tuned using the WT10g collection and TREC topics 451-550 which were used for ad hoc web retrieval at TREC-9 and 10 [12, 13].

Table 1.4 gives a detailed summary of the runs. Each run used an index built from the entire collection of 25,205,179 documents. Documents are stemmed with the Porter stemmer and stopped using a standard list of 420 common terms. In the table, indexing time is the number of minutes required to build the index in parallel across the cluster. Therefore, this number is the *maximum* time required by any single machine to index its subcollection. Index size is the total size of the index on disk including both the inverted file and compressed collection. Average query time is the average number of seconds required to run a query (distributed across the cluster) and retrieve 20 documents. The last column denotes whether or not the run made any use of any document structure, such as titles, headers, etc.

run id	index time (mins)	index size (GB)	avg. query time (s)	struct?
indri04QL	355	224	1.36	no
indri04QLRM	355	224	26.0	no
indri04AW	355	224	6.5	no
indri04AWRM	355	224	39.4	no
indri04FAW	1300	226	52.2	yes

Table 1: Summary of runs.

2 Novelty

2.1 Overview of Our Approaches for the Four Tasks

There are four tasks in this year's novelty track and we participated in all of them. For the 50 topics in the 2004 track, each of them has 25 relevant documents, and zero or more non-relevant documents. Task 1 was to identify all relevant and novel sentences, given the full set of documents for the 50 topics. Task 2 was to identify all novel sentences, given the full set of relevant sentences in all documents. Task 3 was to find the relevant and novel sentences in the remaining documents, given the relevant and novel sentences in the first 5 documents only. Task 4 was to find the novel sentences in the remaining documents, given all relevant sentences from all documents and the novel sentences from the first 5 documents.

We compared the statistics of the 2004 track with both 2002 and 2003 tracks, and have found that the statistics of the 2004 track is closer to the 2003 track. The comparison of the statistics of the 2003 and 2004 novelty track data is shown in 2. Therefore we decided to train our system with the 2003 data when no training from this year's track was available for Task 1 and Task 2, and used the training data from this year's track as it was available for Task 3 and task4. We have already developed an answer-updating approach to novelty detection [1], which gave better performance in terms precision at low recall on both the 2002 and the 2003 novelty track data than the baseline approaches reported in that work. However, we could not use the answer-updating approach directly in the tasks of this year's novelty track because the evaluation measure used in novelty track was the F measure, which is the harmonic mean of precision and recall. Therefore, we used TFIDF techniques with selective feedback for finding relevant sentences and considered the maximum similarity of a sentence to its previous sentences and new named entities to identify novel sentences. The detail descriptions about our approaches are elaborated in the following subsections. Only the main approach for each task will be reported in this paper even though multiple runs for each topic were submitted to TREC from us.

Feature	Track 2003	Track 2004
Num. of Event Topics	28	25
Num. of Opinion Topics	22	25
Num. of Relevant Documents/Topic	25	25
Num. of Non-relevant Documents/Topic	0	11.16
Avg. Num. Sentences/Topic	797.4	1048.8

Table 2: Statistics comparison of 2003 and 2004 track data

2.2 Relevant Sentence Retrieval

For relevant sentence retrieval, our system treated sentences as documents and used the words in the title fields of the topics as queries. TFIDF techniques with pseudo feedback or selective pseudo feedback were used for finding relevant sentences for Task 1 and TFIDF techniques with relevance feedback or selective relevance feedback were used for Task3. Selective pseudo feedback means pseudo feedback was performed on some queries but not on other queries based on an automatic analysis on query words across different topics. Basically, a query with more focused query words that rarely appear in relevant documents related to other queries is likely to have a better performance without pseudo feedback. Selective relevance feedback means whether to performance relevance feedback on a query was determined by the comparison between the performance with and without relevance feedback in the top five documents for this query because the judgment of the top five documents was given for Task 3. Short sentences, non-informative sentences as well as non-normal sentences were removed in the final results. Non-informative sentences are the sentences that have less than n non-stopwords, where the best value of n is 3 (which was learned from the 2003 data). Sentences that have less than m terms are short sentences, where the best value of m is 7 from the 2003 data. Non-normal sentences refer to some special formats for some purposes other than offering the information about the story discussed in a news story. In addition to short sentences, non-informative sentences and non-normal sentences, sentences similar to given non-relevant sentences were also removed for Task 3 when partial judgment was available. Basically if the maximum similarity between a sentence and given non-relevant sentences is greater than a preset threshold (which was trained with the 2003 data), the sentence was treated as non-relevant sentence and thus removed from the result list.

The performance of finding relevance sentences using our approaches on the 2003 and 2004 data for Task1 and Task 3 are given in Table 3 and Table 4 respectively. There are three conclusions that can be drawn from the results. First, the F scores of the original full set of sentences show that how difficult the task is on different data set. It is clear to us that the task of finding relevant sentences on the 2004 data is more difficult than that on the 2003 data. Second, TFIDF techniques work well for relevant sentences retrieval on both the 2003 and 2004 data sets. Third, selective feedback gives better performance than applying feedback on all queries on the two data sets.

2.3 Identifying Novel Sentences

Similarities of a sentence to its previous sentences and the occurrence of new named entities in the sentence are two main factors considered in our approach to identifying novel sentences. New named entities have been used successfully in our answer-updating approach in novelty detection [27].

For Task 1 and Task3, our system started with the list of sentences returned from the relevant sentences retrieval, which unavoidably contains many non-relevant sentences in addition to relevant sentences. For Task 2 and Task 4, our system started with the set of given relevant sentences only. In either case, the cosine similarity between a sentence and each its previous sentence was calculated. The maximum similarity of a sentence to its previous sentences was used to eliminate redundant sentences. Sentences with a maximum similarity value greater than a preset threshold may be treated as redundant sentences. The value of the same threshold for all topics was tuned with the TREC 2003 track data when no training date from this year's was available. The value of the threshold for each topic was trained with the training data when the judgment of the top five documents was given for Task 3 and Task 4. In addition to the maximum similarity between a sentence and its previous sentences, new named entities were also considered in identifying novel sentences. A person's name or an organization in a sentence that did not appear in the previous sentences may give new information about who was related to an event or an opinion [27]. Therefore, a sentence with previously unseen named entities was treated as novel sentences. About 20 types of named entities were considered in our system, which included PERSON, LOCATION, ORGANIZATION, DATE and MONEY, etc. BBN's IdentiFinder [2] and our approach [27] were used for identifying named entities.

The performance of identifying novel sentences for Task 1, Task 2, Task 3 and Task 4 on the 2004 novelty track data are given in Table 5. The F-score on the starting set of sentences (as described above) for each task establishes a bottom line for performance of a novelty detection algorithm. The F-scores were evaluated when we simply assumed all the sentences were novel (without any novelty detection). Any successful novelty detection approach should beat the F-score bottom-line for each task. Table 4 shows that the F-scores of our approaches have significant increases from the bottom lines for all the four tasks.

3 HARD

UMass explored four different sub-tasks in the course of HARD 2004: fixed-length passage retrieval, variable-length passage retrieval, metadata, and clarification form feedback.

In order to allow these tasks to be studied, we established

Approaches	F-score (2003)	F-score (2004)
0. The Original full set of sentences	0.5398	0.303
1. TFIDF models with pseudo feedback	0.6429	0.393 (CIIRT1R1)
2. TFIDF models with selective pseudo feedback	0.6593	0.395 (CIIRT1R2)

Table 3: Performance of finding relevant sentences in Task 1 on 2003 and 2004 data

Approaches	F-score(2003)	F-score(2004)
0. The Original full set of sentences	0.5271	0.306
1. TFIDF models with relevance feedback	0.6229	0.405 (CIIRT3R2)
2. TFIDF models with selective relevance feedback	0.6554	0.406(CIIR31R1)

Table 4: Performance of finding relevant sentences in Task 3 on 2003 and 2004 data

responsibilities for each sub-task. First, we generate a clarification form and receive user feedback. Using the response, the first clarification form module constructs a new, possibly modified query representation. Depending on the retrieval element, the query representation is passed to either a passage retrieval module or a document retrieval module. Both of these modules return a ranked list of items (passages or documents). These items are then re-ranked based upon the satisfaction of topic metadata value. As a post-processing step, the ranked list is further altered by feedback elicited from the clarification form.

3.1 Methods and Materials

3.1.1 Collection Processing

We processed the HARD collection differently for retrieval and metadata classification. For both retrieval and classification, only text between the `<TITLE>` and `<TEXT>` tags were handled.

For retrieval, tokenization was based on non-alphanumeric characters. If a token was not in a list of Acrophile [24] acronyms, then it was down-cased. If a down-cased token was in the libbow stopword list [29], then it was ignored. The Krovetz stemmer [21] packaged with Lemur [1] was used to stem all remaining down-cased words. The topics and related text metadata where processed in the same manner with the additional processing step that `http://` URLs were automatically stripped from the related text.

For metadata classification, contiguous digits were replaced by a token representing a number. The paragraph tag, `<P>`, was retained as a token. Quotation marks, "``" and "''", were converted to the double quote mark, `"`. Contractions were pulled off and became their own tokens (n't, 's, 'd, 'm, 'll, 've, and 're). All punctuation was treated as separate tokens. All remaining text was down-cased and broken at whitespace boundaries.

3.1.2 Training Topics

The LDC supplied training data consists of 21 topics. For each topic, the LDC judged the top 100 documents returned by their search system. We augmented the training topics with additional judgments by obtaining in-house judgments on an additional 100 documents for each topic. This expanded set of judgments was used for parameter tuning.

3.1.3 Query Representation

A query model refers to a probability distribution over words representing the user's information need. In the simplest case, we have the maximum likelihood query model based on the the user's title and description fields. Here, the we would process the text according to section 3.1.1 and then form a maximum likelihood language model using remaining terms as evidence.

3.1.4 Retrieval Using Language Models

A description of retrieval using language models is beyond the scope of this document. We refer readers to the several papers on the subject [6]. We used a modified version of the Lemur language modeling toolkit to perform retrieval [1].

It has been shown that query likelihood and divergence ranking using a maximum likelihood query model are equivalent [23]. Therefore, without loss of generality, we confine our description to divergence-based retrieval. In this approach, we take a query model, $P(w|Q)$, and rank all documents in the collection according to the Kullback-Leibler divergence with $P(w|Q)$,

$$score(D, Q) \quad = \quad \sum_{w} P(w|Q) \log \frac{P(w|Q)}{P(w|D)} \qquad (1)$$

Here, the document language model, $P(w|D)$, may be estimated using a number of different techniques [42]; smoothing parameters used will be described whenever language model retrieval is used.

In addition to the maximum likelihood query model presented in section 3.1.3, we also used *relevance models* for query representation [25]. Relevance models are a form of massive query expansion through blind feedback. Constructing a relevance model entails first ranking the collection according to the maximum likelihood query model. Some set of documents at the top of this ranking become evidence for the relevance model, $P(w|\mathcal{R})$. If we call this set $\mathcal{R}$, then the relevance model is estimated according to,

$$P(w|\mathcal{R}) \quad = \quad \sum_{D \in \mathcal{R}} P(w|D) \frac{P(Q|D)}{\sum_{D' \in \mathcal{R}} P(Q|D')} \qquad (2)$$

where the query likelihood score, $P(Q|D)$, can be easily computed from the divergence measure [33]. The relevance model replaces the maximum likelihood query model in a second round of document ranking.

Approaches	Starting set of sentences	Identify novel sentences
F-scoreTask 1 (Ch%)	0.195	0.211(+8.2%)
F-scoreTask 2 (Ch%)	0.577	0.610(+5.7%)
F-scoreTask 3 (Ch%)	0.194	0.210(+8.2%)
F-scoreTask 4 (Ch%)	0.541	0.577(+6.7%)

Table 5: Performance of identifying novel sentences for Tasks 1-4

Ideally, we would include the entire collection in the set $\mathcal{R}$ and, therefore, $P(w|\mathcal{R})$ would have no terms with zero probability. However, computational limitations force us to let $|\mathcal{R}|$ be fixed; that is, we only consider the top N documents. Furthermore, we also *truncate* and *normalize* the relevance model to include only the M terms with highest probability. The first parameter, N, does not affect the estimation of the relevance model since we are normalizing the query likelihoods. The second parameter, M, requires a little explanation. First, we compute the relevance model as in Equation 2. Second, we order the terms in $P(w|R)$ in decreasing order of probability. Third, we select the top M terms from this ordering. Finally, we normalize these term weights.

A relevance model captures behavior of the returned documents but throws away the original query. In order to maintain information in the original query model, we linearly interpolate the relevance model with the original query model: $P'(w|\mathcal{R}) = \lambda P(w|\mathcal{R}) + (1-\lambda)P(w|Q)$. For our runs where we do this, we specify λ. In our experiments the relevance model is truncated prior to interpolation with the query. Depending on the module, a second truncation and normalization process is performed.

3.1.5 Retrieval Using Support Vector Machines

Of the runs that UMass submitted, several runs involved the use of support vector machines for passage or document retrieval [32]. This technique applies discriminative models to information retrieval. Previous work has demonstrated that the performance of support vector machines on the document retrieval task is on par with that of language models. Our document retrieval and passage retrieval experiments on HARD 2003 test queries and HARD 2004 training queries showed that using SVMs gave better results than traditional language models.

Support vector machines are a class of discriminative supervised learning models. SVMs used for classification create a hyperplane that maximizes the margin from the training examples. The discriminant function used to separate the two classes is given by: $g(R|D,Q) = \mathbf{w} \bullet \phi(\mathbf{f}(D,Q)) + b$, where R denotes the relevant class, D is a document, Q is a query, $\mathbf{f}(D,Q)$ is the vector of features, $\mathbf{w}$ is the weight vector in kernel space that is learned by the SVM from the training examples, $\bullet$ denotes inner product, b is a constant and ϕ is the mapping from input space to kernel space. The value of this discriminant function is proportional to the distance between the document D and the separating hyperplane in the kernel space.

The features are term-based statistics commonly used in information retrieval systems such as *tf*, *idf* and their combinations as shown in Table 6. Each of the six features is a sum over the query terms.

In order to provide a finer-grained weighting of query terms, we incorporated the query models described in Section 3.1.3 into our features. These hybrid features are presented in Table 6. In all cases, retrieval was performed using the hybrid features. However, unless otherwise noted, all models were built using the regular features.

The corpora, queries and relevance judgments for TREC 1 and TREC 2 provided training data. All the documents marked relevant for a query were used as positive training instances. An equal number of negative instances were obtained by random sampling of the remaining documents. These training instances are represented in terms of their transformed feature vectors in the kernel space. The support vector machine then learns the hyperplane that separates the positive and negative training instances with the highest margin. For our runs, we used a linear kernel. Hence the hyperplane is drawn in the original feature space. The equation of this hyperplane provides the discriminant function $g(R|D,Q)$ that is subsequently used for scoring documents (or fixed length passages).

The indexed elements (documents or passages) are treated as instances in the feature space. For a test topic, Q, an instance D is scored based on the value of the discriminant function $g(R|D,Q)$. The instances are then ranked based on this score.

3.1.6 Bootstrapping SVMs

Previous work has balanced classes by random sampling from the negative training instances [32]. We propose another technique for instance sampling, which we refer to as *bootstrapping*. This method differs from the random sampling technique in the selection of negative training instances. All positive instances are used for training as in the previously described sampling method. In bootstrapping, negative instances are selected in the following way. First, an initial SVM is created using the technique described in section 3.1.5. Then, negative training instances are selected from only the negative examples *misclassified* by the initial SVM created in step 1. As many negative instances are selected as there are positive instances. This training set is used to create an SVM boundary as described in section 3.1.5.

Sampling from the set of misclassified negative documents, as opposed to sampling from all the negatives, will produce a set of negative training instances that are closer to the positive instances in the feature space. The intuition is that this will result in a boundary that is still good for ranking but has fewer misclassified instances on the positive side.

3.2 Clarification Form Feedback

This year's HARD track again permitted sites to request one round of feedback from the topic creator. UMass studied four

	Features	Hybrid Features
1	$\sum_{q_i \in Q \cap D} \log(c(q_i, D))$	$\sum_{w \in V} P(w\|Q) \log(c(w, D))$
2	$\sum_{i=1}^{n} \log(1 + \frac{c(q_i,D)}{\|D\|})$	$\sum_{w \in V} P(w\|Q) \log(1 + \frac{c(w,D)}{\|D\|})$
3	$\sum_{q_i \in Q \cap D} \log(idf(q_i))$	$\sum_{w \in V} P(w\|Q) \log(idf(w))$
4	$\sum_{q_i \in Q \cap D} (\log(\frac{\|C\|}{c(q_i,C)}))$	$\sum_{w \in V} P(w\|Q)(\log(\frac{\|C\|}{c(w,C)}))$
5	$\sum_{i=1}^{n} \log(1 + \frac{c(q_i,D)}{\|D\|} idf(q_i))$	$\sum_{w \in V} P(w\|Q) \log(1 + \frac{c(w,D)}{\|D\|} idf(w))$
6	$\sum_{i=1}^{n} \log(1 + \frac{c(q_i,D)}{\|D\|} \frac{\|C\|}{c(q_i,C)})$	$\sum_{w \in V} P(w\|Q) \log(1 + \frac{c(w,D)}{\|D\|} \frac{\|C\|}{c(w,C)})$

Table 6: Features in the discriminative models: $c(w, D)$ represents the raw count of word w in document D, C represents the collection, n is the number of terms in the query, $|.|$ is the *size-of* function and $idf(.)$ is the inverse document frequency. In the case of the hybrid features, $P(w|Q)$ refers to a query model as described in Section 3.1.3. We define $\log(0) = 0$.

methods for eliciting user feedback. Different manifestations of these methods appeared on our submitted clarification forms.

3.2.1 Clarification Form Subsections

Passages Although the three minute time limit constrained our ability to request true document-level relevance judgments, we assumed that the presentation the most relevant *passages* retrieved would serve as an acceptable surrogate. Specifically, we performed SVM-based retrieval on a passage index comprised of 150-word overlapping passages. We used a linear model trained on TREC collections 1 and 2. We then split the top 15 document-unique passages into 25-word passages and selected the passage which the SVM scored the highest. These 15 25-word passages were then presented to the user with the document title and time stamp for feedback.

In addition to selecting the top 15 document-unique 150-word passages, we also experimented with using agglomerative clustering to remove redundancy from the passages presented. We used group-average, agglomerative clustering [28]. Term vectors were weighted according to a tf.idf scheme, $weight(x_i) = x_i/(\log((|C|+1)/(0.5+df_i)))$. Using these vectors, a cosine measure was used to compute the similarity matrix. We clustered clustered 200 150-word passages until a threshold similarity of 0.6 was reached. At that point, the largest 15 clusters were selected. The 15 150-word centroid passages of these clusters were then split into 25-word passages to be handled as above.

Query Reformulation Because the title and description subsections of the topic do not often serve as a good representation of a realistic user query, we allowed users to modify the stopped and stemmed version of their title and description query using a free entry text box.

Extracted Entities Previous work has shown that user feedback of term lists tends to have little (and sometimes negative) impact on retrieval performance [36]. We were interested in exploring the potential advantage of using different types of words as feedback candidates [22]. In particular, we were interested in the use of proper names rather than arbitrary terms as feedback sources. To accomplish this, we gathered the top 200 150-word passages after an initial retrieval and ran BBN's Identifinder across this set of passages [2]. We extracted the person, place, and organization names from this run and normalized the names by down-casing and removing punctuation and spaces. After removing names such as "New York Times", "AFP", and other source tags, we presented the user with the 15 most frequently occurring people, places, and organizations.

For each of these types of named entities, the user was also presented with a text box in which enter named entities not in the top 15 for that type.

Temporal Feedback Previous work has shown that some topics demonstrate strong temporal structure [9, 26]. In order to elicit temporal biases in the information need, we asked the user for relevant months in the year spanned by the collection.

3.2.2 Official Clarification Forms

CF1 Our first clarification form included a list of 15 25-word passages derived from the top 15 150-word passages, a query reformulation text box, a free-text named entity text box, and a temporal feedback interface.

CF2 Our second clarification form included a list of 15 25-word passages derived from clustering, a query reformulation text box, a free-text named entity text box, and a temporal feedback interface.

CF3 Our third clarification form included the list of 15 people, 15 places, and 15 organizations with free-entry for each entity type, a temporal feedback interface, and a query reformulation text box.

3.2.3 Incorporation of Responses

Passages Passage feedback was used in two ways. First, we performed query expansion based upon the relevant passages.

A query model was constructed by uniformly combining the language models of the relevant documents. We selected the top 200 terms from this distribution and renormalized the weights. This was our final query model for relevant passages. Secondly, passage feedback was used in order to re-rank documents at the end of the treatment. Specifically, we multiplied all final scores by 1 if they were from a document marked relevant, 0 if from a document marked non-relevant, and 0.5 otherwise.

Query Reformulation Whenever the user reformulated a query, we discarded the original query and constructed a query model from the new query strings.

Extracted Entities All relevant named entities or named entities entered in the free text box were combined to construct a named entity query model.

Temporal Feedback Temporal feedback was used in order to re-rank documents at the end of the treatment. Specifically, we multiplied all final scores by 1 if they were from a month marked relevant, 0 if from a month marked non-relevant, and 0.5 otherwise.

3.3 Fixed-Length Passage Retrieval

Passage retrieval was one of the issues that were studied as part of the HARD track. The central goal of the track was to perform high accuracy retrieval. Retrieving passages instead of whole documents could potentially return less non-relevant text at the top of the ranked list, thereby increasing the accuracy of the search.

Previous work [3] on passage retrieval has shown that there is no significant improvement in retrieval performance when "real" passage boundaries are detected, using sentence boundaries or paragraph boundaries. Fixed length passages have been shown to perform as well as passages extracted using heuristics on document retrieval tasks [3]. We, therefore, decided to avoid the overhead involved in creating a system that segments documents into passages "cleverly" and opted to use passages that have a fixed word length.

We explored various approaches to passage retrieval. We studied the performance of passage retrieval systems that used query likelihood, relevance models and support vector machines. Passage retrieval using SVMs, described in 3.1.5 performed better than the other systems. We also explored the comparative utility of retrieving the best passages from top ranked documents versus indexing overlapping passages and scoring each of these independent of the document that the passage came from. The latter method gave higher precision on our training data. Therefore, we scored pre-indexed passages for our final run.

One of the issues that we had to resolve was the size of passages to be retrieved. Experiments on HARD 2003 test queries indicated that retrieval using 100 word passages gave the best results. This was the passage size that was used for all the fixed-passage runs.

3.4 Variable-Length Passage Retrieval

One of the questions UMass explored through the passage retrieval portion of the HARD track was whether retrieving passages of different lengths could improve our ability to return only the relevant portions of documents. In order to keep our text index relatively small and maintain the theoretical possibility that any passage of any document could be retrieved by the system, we chose to extract passages from highly ranked documents at the time of retrieval, rather than indexing particular passages in advance.

Previous work has found no benefit to retrieving passages of different lengths, compared to overlapping fixed-length passages [19, 3]. However, past studies have only evaluated passage retrieval by its ability to retrieve relevant documents, due in part to the unavailability of passage-level relevance judgments. Now that the HARD track has provided these judgments and the evaluation is based on more fine-grained retrieval, we decided to revisit this question.

3.4.1 Extracting Relevant Passages

Our method of extracting relevant passages from documents is inspired by work by de Kretser and Moffat [7] that assigned a relevance score to every word in a document. They used term frequency within the query and inverse term frequency in the corpus to determine the score of each word, and used several different functions to determine how much query terms contributed to the scores of surrounding words.

Our approach to selecting relevant passages is similar, in that each term from an expanded query representation is assigned a score which affects the scores of proximal words. However, the scores we use are derived from language models, and the task is somewhat different.

This process of extracting passages for a topic starts with the top-ranked documents from some document run and a language model representing the topic. Of the different topic models we tried, the best-performing one was a mixture model between the maximum likelihood representation of the original query and the top 50 terms from the relevance model for the query, as described in section 3.1.4.

We refer to the range of word positions in a document that a particular query word affects as its *region of influence*. The *spread* of a query term is the number of words before it and after it that that query term influences. Thus, the size of the region of influence is equal to (2 $\times$ spread) + 1. This method takes as parameters the minimum spread and the maximum spread that any particular query term can have. The weights of the topic model are then linearly scaled to fall between these minimum and maximum values. For all of our submitted runs that used passages of varying lengths, the minimum spread was 1 and the maximum spread was 25.

We extract any group of words that falls within the region of influence of any query term as a passage, discarding passages with fewer than 400 characters. Next we score the remaining passages as described in the following section.

3.4.2 Scoring Passages

We experimented with several methods for scoring passages that fall into two basic classes. The first group used SVMs to score passages. The second assigned scores equal to the negative relative entropy between the topic and passage language models, but differed in how the passage was modeled.

Metadata	Pos.	Neg.	Total
Genre news-report	848	491	1339
Genre opinion-editorial	147	1192	1339
Genre other	344	995	1339
Geography US	590	758	1348

Table 7: Counts of human judgments collected for the genre and geography metadata broken down by positive and negative judgments.

Metadata	Pos.	Neg.	Total
Genre news-report	2603	2280	4883
Genre opinion-editorial	1633	3250	4883
Genre other	647	4236	4883
Geography US	1470	1451	2921

Table 8: Counts of judgments obtained by using the `<KEYWORD>` element of the documents to automatically guess a document's genre and geography.

Using the SVM models described in section 3.1.5 to score the passages did not perform as well as other methods on the training data, regardless of which topic representation we used. This was surprising because this technique works quite well with fixed-length passages. This could be a result of choosing the wrong topic representation, a bug in the implementation, or maybe the non-uniform passage lengths have an adverse effect.

For the class of relative-entropy-based measures, we tried three different topic models. The first used Dirichlet smoothing of the maximum-likelihood passage model with the collection model as the background model. The second used Dirichlet smoothing of the maximum-likelihood passage model with the document model as the background. Neither of these methods performed well on the training data.

The best-performing passage representation, used in UMassVPMM and UMassCVC, was a mixture of the collection, document, and passage models.

$$\begin{aligned} p(w|\Theta_{PSG}) &= \lambda_c p(w|\Theta_{ML_c}) + \lambda_d p(w|\Theta_{ML_d}) \\ &+ \lambda_p p(w|\Theta_{ML_p}) \end{aligned} \quad (3)$$

Θ_{ML_c}, Θ_{ML_d}, and Θ_{ML_p} are the maximum likelihood collection, document, and passage models respectively. The three lambdas sum to 1. In our submitted runs, λ_c was 0.8, and the other two parameters were 0.1.

Future work will investigate the possibility of using two different topic models for the passage extraction and passage scoring stages of this technique.

3.5 Metadata

For metadata our approach was to take a ranked list of documents and rerank the list based on the topic's metadata values. For the genre and geography metadata values we trained classifiers to determine to what degree a document satisfies the metadata value. Documents that better satisfy the metadata values are moved up in the ranked list compared to those that do not satisfy the metadata values.

3.5.1 Data Collection for Classifiers

We used several human annotators to obtain metadata judgments on documents from the collection. The majority of the judgments came from one of the authors. Table 7 shows the breakdown of judgments obtained by humans for each metadata category.

To boost performance, we automatically extracted training data from the corpus using the corpus' existing metadata. The AP wire, New York Times, and LA Times either contained explicit metadata in the `<KEYWORD>` element or was discernible in some other manner. The number of judgments collected in this mainly automatic fashion are shown in Table 8.

While we knew that this process would lead to mistakes, we did spot check the extracted documents, and we felt the gain from the additional training data exceeded the cost in misclassified examples. Also, we had counter balanced this automatically extracted data with over 1000 human judgments covering all subcollections.

3.5.2 Classifier Technology

We used linear support vector machines (SVMs) as our classifiers because of their success at text classification [41, 16, 10] and their ability to produce a ranking rather than merely a class prediction. The linear SVM learns a hyperplane in the feature space of the training examples that separates positive from negative examples. A document's distance from the hyperplane determines the degree to which the SVM predicts the document is a positive or negative example of the learned class. We used SVM^{light} with its default settings compiled for Windows to perform all classification [15].

3.5.3 Classifier Features

We used the same set of features for each of our classifiers. Our selection of features was guided by the choices others have used for the classification of text genre [18, 20, 37, 8, 11]. We used the 10K most frequently occurring tokens in the corpus. If a document contained one of these tokens, the corresponding feature value was 1 otherwise it was 0. We also used the out of vocabulary probability mass. The 10K most frequently occurring tokens constituted our vocabulary. We made eight binary features, one for each subcollection in the HARD collection: AFE, APE, CNE, LAT, NYT, SLN, UME, and XIE. Finally, we constructed a set of features focused on various length measures of a document: number of tokens, average token length, average sentence length, average paragraph length, variance in paragraph lengths, average corpus frequency of tokens, and four features that measured the number of words $<= X$ characters long where X was one of 6,7,8, and 9. We normalized each of these measures to vary between 0 and 1. We first took the log of the sentence, paragraph, and document length features before normalizing them.

3.5.4 Classifier Training

To deal with imbalances in the number of positive examples per class, we randomly oversampled from either the positive or negative examples, whichever was in the minority until 50% of the examples were positive [40]. No other special techniques were used.

Metadata	Avg. Prec.	Accuracy	F1
Genre news	0.99	0.96	0.96
Genre op-ed	0.97	0.95	0.91
Genre other	0.82	0.92	0.76
Geo. US	0.96	0.92	0.91

Table 9: This table describes the performance of SVM classifiers on the labeled data. All performance measures are averages from 3-fold cross validation. The class examples are oversampled so that positive examples comprise 50% of the training examples.

The performance of the classifiers on the final datasets is shown in Table 9. We aimed to improve average precision, which measures ranking ability, while keeping an eye on the other measures. One could obtain a high average precision while doing poorly on accuracy.

While these metrics are certainly indicative of the classifiers' power, some caveats must be stated. The HARD corpus contains many articles that are posted to the newswires multiple times in order to add more information or make small corrections. Our automatically judged articles may in fact contain several near copies of the same document. In addition, we included many examples from the same columnists. It is likely that a columnist's pieces are more similar to each other than a selection of opinion pieces written by different authors. These duplicates can thus straddle the train and test sets of the 3-fold cross validation and artificially inflate the performance metrics.

3.5.5 Metadata Reranking

We reranked the results based on a linear combination of the normalized outputs of both the retrieval and classifier outputs. We normalize each classifier's output across the whole corpus. For each topic, the document scores were normalized with the rank 1 document score set to 1 and rank 1000 document score set to 0. We rerank passages as though they were documents.

We tuned the linear combination with a simple parameter sweep using the LDC hard-relevance training data augmented with additional UMass judgments. The best coefficients found weighted the original IR results at 0.5, geography at 0.1, and genre at 0.4.

3.5.6 Use of Related Text

To utilize the related text metadata, we created a maximum likelihood model of the related text provided with the topic and linearly mixed this model with a model created for the title and description. This mixture model was used as the query. A parameter sweep was used to find the best mixture ratio on the training topics. The title and description model had a weight of 0.4 and the related text model had a weight of 0.6. We did not differentiate between on-topic and relevant related text and used both together.

3.6 HARD Runs

We submitted three baseline runs (UMassBaseQL, UMassBaseRM3, UMassBaseSVM) that did not use any of the metadata, clarification form, or passage techniques described earlier. Our other ten runs aimed to investigate the use of these techniques.

UMassBaseQL This run uses the maximum likelihood query model as described in section 3.1.4. It used both the title and the description. Smoothing was performed using the Dirichlet prior with its parameter set to 1000. **UMassBaseRM3** For this run, we used the title and description and the relevance modeling approach described in section 3.1.4. We used the first 50 documents retrieved to build the relevance model. The model was truncated to include only the 200 words of highest probability with a minimum probability of 0.001. The foreground model (the title and description) received a weight of 0.6 when mixed with the relevance model. Smoothing was performed using the Dirichlet prior with its parameter set to 1000. **UMassBaseSVM** This run used a support vector machine built from the normal features in Table 6 to retrieve documents using a hybrid representation. **UMassMerge** This run merged three different rankings. The first ranking used CF1 and all associated feedback. This run used the passage feedback and reformulation for building a query model. A hybrid SVM was used for an initial retrieval. This ranked list was reranked using temporal and document feedback. List two is UMassF. List three was identical to UMassRGG for the document topics. For the passage topics, passages were reranked using the genre and geography metadata as described in section 3.5.5. The source of the passages came from the fixed length SVM passage retrieval used by run UMassCFMC, which used a query model produced by CF1 and related text. These passages were reranked prior to removal of overlap as opposed to the passages in UMassCFMC which were reranked after overlap in the passages had been removed. The three lists were each normalized and merged by summing the scores of identical documents or passages and ranked according this sum. Overlap in passages were removed and the lists were trimmed to the top 1000 results. **UMassCFMC** This run was a pipeline of the CF1 clarification form, bootstrapped SVM retrieval, and genre and geography metadata reranking. The linear bootstrapped model used for UMassF was used with the query generated from the responses to CF1 as well the related text. Ranked lists were generated for document and passage topics in the same manner as for UMassF. The results were then normalized and reranked using the genre and geography metadata as per section 3.5.5. We performed temporal and document feedback to provide a final ranking. **UMassCFC** The linear bootstrapped model used for UMassF was used with the query generated from the responses to the clarification form, CF1. Ranked lists were generated for document and passage topics in the same manner as for UMassF. We performed temporal and document feedback to provide a final ranking. **UMassCMC** The initial retrieval was performed using a query model built from CF1. These results were then reranked using topic metadata values. We utilized the geography, and genre metadata to rerank the results from the clarification form. We performed temporal and document feedback to provide a final ranking.

UMassCVC UMassCVC used variable-length passage techniques described in section 3.4, starting from the baseline document run UMassBaseSVM and the top 50 terms from the query model generated from the response to clarification form CF1. After variable-length passage retrieval, we post-processed the results as described in 3.2.3. For the 25 topics where the retrieval element was documents, the results we submitted were identical to the results from our baseline run UMassBaseSVM.
UMassF For the 25 document topics, query models were generated using the top 10 results of a preliminary ranked list as described in section 3.1.3. This preliminary list was obtained by retrieving 100 word passages using query likelihood. The title and description was used as the query for each topic. A linear bootstrapped model was used for retrieval. The top 1000 documents were returned for each of the 25 document topics. The same process as above was repeated for the 25 passage topics, except that a passage index was used for retrieval. The top 1000 non-overlapping passages were returned for each of these topics.
UMassRGG This run utilized the related text, geography, and genre metadata. Documents were returned for all topics. The metadata was utilized as described in sections 3.5.6 and 3.5.5. Retrieval was via query likelihood with Dirichlet smoothing. The smoothing parameter was set to 1000.
UMassVPMM UMassVPMM was a baseline passage run of sorts; it does not use any metadata or clarification form feedback for retrieval. It used variable length passage retrieval as described in section 3.4. We used the interpolated relevance model query model described in 3.1.4. We used the baseline run UMassBaseSVM as the starting ranked document list. Because we found in training that boosting the scores of passages from the top 25 documents improved results, we added a constant to the score of each of these passages, large enough to ensure that they would be ranked above all other passages. For the 25 topics where the retrieval element was documents, the results we submitted were identical to the results from our baseline run UMassBaseSVM.
UMassC2 This run used the passage feedback and reformulation for building a query model. A hybrid SVM was used for an initial retrieval. This ranked list was reranked using temporal and document feedback.
UMassC3 This run used the named entity feedback and reformulation for building a query model. A hybrid SVM was used for an initial retrieval. This ranked list was reranked using temporal feedback.

3.7 Results and Discussion

In order to allow an initial analysis of our various techniques, we generated several new runs based on different combinations of feedback, metadata handling, and retrieval granularity. These runs were evaluated using relevance judgments for the HARD 2004 topics. Results are presented in Tables 10 and 11.

3.7.1 Clarification Forms

Our initial experiments allow us to investigate broad issues in ranking alternatives and named entity performance.

Passage Ranking Comparing the baseline, CF1, and CF2 rows in Tables 10, and 11, we observe that, in general passage feedback tends to improve performance. This result is not surprising given previous work in relevance feedback. What is a little more surprising is that clustering the results did not provide any advantage over the standard ranking. In fact, clustering often resulted in worse performance. One explanation for this behavior is the strictly positive nature of our feedback. Query models were built from positive documents. Negative information was essentially discarded. Therefore, to maximize the amount of information it receives, a system should get feedback from the documents which it is *most confident* about. By definition, these documents (or passages) will be the ones at the top of the ranked list. This intuition is confirmed by the number of passages marked relevant in the CF1 and CF2 clarification forms. On average, CF1 garnered more positive responses from users.

This result motivates two questions. First, how do we incorporate negative feedback into our existing framework? Research in retrieval by language models has ignored the question of negative feedback. If interaction and relevance feedback is to be considered an important aspect of HARD, it seems necessary to develop models for negative feedback. Second, how do we improve clustering so that removing redundancy does not result in detrimental loss of information in feedback? This question assumes both that the feedback in the likelihood ranking approach is redundant and that the feedback in the clustered approach is inferior. These assumptions need to be confirmed. Moreover, a similar question presents itself in novelty and subtopic retrieval and models from work in that field could improve future passage-based feedback forms.

Named Entities The results for runs using named entity information seem to confirm the difficulty of handling term-based feedback. The impact of named entity expansion is inconclusive. Training experiments demonstrate that, given the proper weighting of named entities, retrieval can be improved to the level of document feedback. That is, if we can detect that a person name is more important than a geographic name *for a particular query*, then we can match document feedback performance. However, the models we constructed used a uniform weight for all queries; person names always weighed the same as geographic and organizational names. Future experiments will attempt predict the relative import of entity types based on the query and corpus statistics.

3.7.2 Metadata

The results of using the related-text (RT), reranking results by genre and geography (GG), and the combination of RT and GG can be seen in Tables 10 and 11. For document retrieval, our use of related-text resulted in results as good as the use of the clarification form. In these tables, there is no evidence we were able to leverage genre and geography. We examined the use of genre applied to runs UMassBaseQL, UMassBaseSVM, UMassBaseRM3, QL+RT, and RM+RT on topics requesting a specific genre. We found an average 7% increase of precision at five documents for hard relevance and 8% for soft relevance. No similar increase was found for use of geography metadata.

We suspect the value of the genre and geography metadata

	QL_{doc}	RM_{doc}	SVM_{doc}^{trec12}	$VPMM$	$SVMH_{psg}^{trec12}$	SVM_{psg}^{boot}
baseline	0.222	0.218	0.223	0.211	0.174	0.185
GG	0.217	0.222	0.224	0.185	0.178	0.192
RT	0.272	0.262	0.214	0.196	0.231	0.214
GG+RT	0.257	0.255	0.207	0.196	0.231	0.206
CF1	0.335	0.331	0.307	0.308	0.298	0.294
CF2	0.263	0.262	0.252	0.306	0.253	0.275
CF3	0.228	0.230	0.246	0.191	0.192	0.207
GG+CF1	0.326	0.327	0.289	0.295	0.300	0.309

Table 10: Binary Preference at 12,000 characters for passage and document runs. QL refers to query-likelihood retrieval, RM to relevance model retrieval, SVM to retrieval using a support vector machine *trained* using normal features, and $SVMH$ to retrieval using a support vector machine *trained* using hybrid features. Both SVM and $SVMH$ used hybrid feature vectors for *retrieval*. Subscripts indicate whether documents or passages were presented in the ranked list. Superscripts indicate the data which the SVM was built from. GG runs used genre and geography metadata, RT used related text, and CF* used clarification form query models and re-ranking.

	QL_{doc}		RM_{doc}		SVM_{doc}^{trec12}		$SVMH_{doc}^{trec12}$		SVM_{doc}^{boot}	
	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft
baseline	0.327	0.320	0.343	0.339	0.322	0.314	0.286	0.300	0.287	0.318
GG	0.307	0.314	0.325	0.330	0.336	0.319	0.308	0.297	0.292	0.311
RT	0.359	0.373	0.348	0.355	0.365	0.361	0.337	0.353	0.342	0.354
GG+RT	0.363	0.363	0.348	0.353	0.347	0.355	0.358	0.351	0.317	0.342
CF1	0.341	0.361	0.339	0.362	0.357	0.374	0.335	0.358	0.347	0.363
CF2	0.316	0.330	0.315	0.329	0.323	0.345	0.320	0.338	0.336	0.372
CF3	0.333	0.298	0.333	0.303	0.332	0.329	0.319	0.323	0.297	0.326
GG+CF1	0.331	0.368	0.333	0.368	0.352	0.368	0.334	0.353	0.338	0.361

Table 11: Document R-Precision for hard and soft relevance. Labels are described in the caption to Table 10.

is limited partly because topics may in fact disambiguate themselves with respect to metadata such that the majority of on-topic documents already satisfy the metadata. We expected topics to be ambiguous with respect to their metadata, but many were not. Eleven of the 45 topics were completely unambiguous, i.e. all on-topic documents satisfied the metadata. Looking at the fraction of on-topic documents that were relevant, across topics the median fraction was 0.83. The training topics were similarly unambiguous with respect to metadata.

Another factor limiting the power of genre and geography metadata could be that searchers are unable to express their metadata needs correctly. On an initial exploratory analysis of the retrieval results, we discovered many documents judged relevant that clearly fall outside the requested metadata. Searchers know a relevant document when they see one, but a priori they don't fully know what metadata is required of a relevant document.

3.7.3 Passage Retrieval

Table 10 reveals two major findings in passage retrieval. First, document runs (shown in the first three columns) generally tend to do better than passage runs (columns 4-6) at passage retrieval, when a high-precision character-level metric such as binary preference at 12,000 characters is used for evaluation. Second, CF1 seems to provide big improvements over the baseline for every retrieval method.

As for the question of whether variable-length passages improve high-accuracy passage retrieval, the results in table 10 are somewhat misleading. Although VPMM did better than both the bootstrap SVM and the hybrid SVM as a baseline, experiments performed after the TREC submission deadline showed that the gain there comes from the difference in retrieval method, not in passage length. Preliminary experiments using the mixture model of VPMM on fixed-length passages provide a better baseline than any of the document or passage runs presented here.

The bootstrap SVM method provides a small gain over the hybrid SVM method for all combinations of clarification forms and metadata, except for those involving related text. Interestingly, it seems that within the group of passage runs, the lower the baseline score, the bigger the boost from related text. In fact, VPMM is even hurt by the use of related text.

The major question raised by our findings for passage retrieval is whether passage retrieval is worthwhile, given that document retrieval almost always does better than passage retrieval for this evaluation metric. Or are we simply using the wrong evaluation metric for what we are really trying to measure? The official TREC 2003 HARD track metric of passage R-precision got at the notion that systems should be rewarded for returning text from many different documents. The character level measures correct a flaw in passage-level R-precision that favored very short passages, but remove this notion that there is some inherent good in returning text from a variety of documents. The problem of how to evaluate passage retrieval has clearly not been solved yet.

4 Acknowledgments

This work was supported in part by the Center for Intelligent Information Retrieval, SPAWARSYSCEN-SD grant number N66001-02-1-8903, and Advanced Research and Development Activity and in part by NSF grant number CCF-0205575. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

References

[1] J. Allan, J. Callan, K. Collins-Thompson, B. Croft, F. Feng, D. Fisher, J. Lafferty, L. Larkey, T. N. Truong, P. Ogilvie, L. Si, T. Strohman, H. Turtle, and C. Zhai. The lemur toolkit for language modeling and information retrieval. `http://www.cs.cmu.edu/~lemur/`, 2003.

[2] D. M. Bikel, R. L. Schwartz, and R. M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231, 1999.

[3] J. P. Callan. Passage-level evidence in document retrieval. In *SIGIR 1994*, pp. 302–310.

[4] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *DEXA-92*, pp. 78–83.

[5] C. Clarke, G. Cormack, and F. Burkowski. Shortest substring ranking (multitext experiments for trec-4). In *TREC 4*, 1995.

[6] W. B. Croft and J. Lafferty. *Language Modeling for Information Retrieval*. Kluwer, 2003.

[7] O. de Kretser and A. Moffat. Effective document presentation with a locality-based similarity heuristic. In *SIGIR 1999*, pp. 113–120.

[8] N. Dewdney, C. VanEss-Dykema, and R. MacMillan. The form is the substance: Classification of genres in text. In *Workshop on Human Language Technology and Knowledge Management, ACL 2001*.

[9] F. Diaz and R. Jones. Using temporal profiles of queries for precision prediction. In *SIGIR 2004*, pp. 18–24.

[10] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *CIKM 98*, pp. 148–155.

[11] A. Finn and N. Kushmerick. Learning to classify documents according to genre. In *IJCAI-03 Workshop on Computational Approaches to Style Analysis and Synthesis*.

[12] D. Hawking. Overview of the trec-9 web track. In *TREC 9*, 2000.

[13] D. Hawking and N. Craswell. Overview of the trec-2001 web track. In *TREC 10*, 2001.

[14] S. Heinz and J. Zobel. Efficient single-pass index construction for text databases. *JASIST*, 54(8):713–729, 2003.

[15] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[16] T. Joachims. A statistical learning learning model of text classification for support vector machines. In *SIGIR 2001*, pp. 128–136.

[17] I.-H. Kang and G. C. Kim. Integration of multiple evidences based on a query type for web search. *Info. Proc. and Mgt.*, 40(3):459–478, 2004.

[18] J. Karlgren and D. Cutting. Recognizing text genres with simple metrics using discriminant analysis. In *15th conference on Computational linguistics*, pp. 1071–1075. ACL, 1994.

[19] M. Kaszkiel and J. Zobel. Passage retrieval revisited. In *SIGIR 1997*, pp. 178–185.

[20] B. Kessler and G. N. H. Schütze. Automatic detection of text genre. In *ACL/EACL 1997*, pp. 32–38.

[21] R. Krovetz. Viewing morphology as an inference process. In *SIGIR 1993*, pp. 191–202.

[22] G. Kumaran and J. Allan. Text classification and named entities for new event detection. In *SIGIR 2004*, pp. 297–304.

[23] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *SIGIR 2001*, pp. 111–119.

[24] L. S. Larkey, P. Ogilvie, M. A. Price, and B. Tamilio. Acrophile: an automated acronym extractor and server. In *5th Conf. Digital libraries*, pp. 205–214. ACM, 2000.

[25] V. Lavrenko. *A Generative Theory of Relevance*. PhD thesis, UMass, 2004.

[26] X. Li and W. B. Croft. Time-based language models. In *CIKM 2003*, pp. 469–475.

[27] X. Li and W. B. Croft. An answer-updating approach to novelty detection. TR IR-359, CIIR, UMass, 2004.

[28] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[29] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[30] D. Metzler and W. B. Croft. Combining the language model and inference network approaches to retrieval. *Info. Proc. and Mgt.*, 40(5):735–750, 2004.

[31] D. Metzler, V. Lavrenko, and W. B. Croft. Formal multiple bernoulli models for language modeling. In *SIGIR 2004*, pp. 540–541.

[32] R. Nallapati. Discriminative models for information retrieval. In *SIGIR 2004*, pp. 64–71.

[33] P. Ogilvie and J. Callan. Experiments using the lemur toolkit. In *TREC-10*, pp. 103–108, 2001.

[34] P. Ogilvie and J. Callan. Combining document representations for known item search. In *SIGIR 2003*.

[35] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR 1998*, pp. 275–281.

[36] I. Ruthven. Re-examining the potential effectiveness of interactive query expansion. In *SIGIR 2003*, pp. 213–220.

[37] E. Stamatatos, N. Fakotakis, and G. Kokkinakis. Text genre detection using common word frequencies. In *18th Int. Conf. on Comp, Linguistics*, 2000.

[38] H. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *TOIS*, 9(3):187–222, 1991.

[39] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Info. Proc. and Mgt.*, 31(6):831–850, 1995.

[40] R. Yan, Y. Liu, R. Jin, and A. Hauptmann. On predicting rare classes with SVM ensembles in scene classification. In *ICASSP*, 2003.

[41] Y. Yang and X. Liu. A re-examination of text categorization methods. In *SIGIR 1999*, pp. 42–49.

[42] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.