Indri at TREC 2004: Terabyte Track

Donald Metzler, Trevor Strohman, Howard Turtle, W. Bruce Croft

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

Abstract

This paper provides an overview of experiments carried out at the TREC 2004 Terabyte Track using the Indri search engine. Indri is an efficient, effective distributed search engine. Like INQUERY, it is based on the inference network framework and supports structured queries, but unlike INQUERY, it uses language modeling probabilities within the network which allows for added flexibility. We describe our approaches to the Terabyte Track, all of which involved automatically constructing structured queries from the title portions of the TREC topics. Our methods use term proximity information and HTML document structure. In addition, a number of optimization procedures for efficient query processing are explained.

1 Introduction

The Indri search engine[1], developed as part of the Lemur Project, is designed to be both efficient and effective over a wide range of collections, especially large, semi-structured text collections, such as the web. This year's TREC Terabyte Track provided a useful platform to test our new system. Indri makes use of INQUERY's underlying inference network retrieval framework, which allows complex structured queries to be constructed and evaluated [1]. However, Indri makes use of language modeling probabilities instead of INQUERY's *tf.idf*-based probabilities, which provides increased robustness, as reflected in the Indri query language.

For this year's track we have two objectives. First, we wish to evaluate the effectiveness of our retrieval model. We devise several methods of automatically constructing complex structured queries from natural language descriptions of information needs. We explore several such methods, including the use of phrases and query expansion. Second, we are interested in evaluating the efficiency of the engine. We use document-at-a-time scoring, and explore several query optimization techniques.

[1] Available for download at http://www.lemurproject.org

Therefore, the goal of this paper is to provide a broad overview of Indri and our approaches to the Terabyte Track. In the remainder of this paper we describe Indri's underlying retrieval model, its indexing and query processing infrastructures, details of the runs we submitted, and an evaluation of the results.

2 Task

The focus of this year's track is content-based search over a large web collection. The collection, named GOV2, consists of a crawl of the entire .gov web domain. Despite it being the Terabyte Track, the collection only weighs in at 426GB uncompressed, which is still significantly larger than most past TREC collections. The collection is made up of 25,205,179 documents, which are mostly HTML documents (91.7%), but also includes plain text versions of crawled PDF, PS, and MS Word documents.

3 Model

The retrieval model implemented in the Indri search engine is an enhanced version of the model described in [9], which combines the language modeling [12, 3] and inference network [13] approaches to information retrieval. The resulting model allows structured queries similar to those used in INQUERY to be evaluated using language modeling estimates within the network, rather than *tf.idf* estimates. Figure 3 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 [9] and [13] to fill in any missing details.

3.1 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*.

3.2 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 [10]. This overcomes the theoretical issues encountered in [9]. 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). The Beta is chosen for simplicity, as it is the conjugate prior to the Bernoulli distribution. Thus, $P(D|\theta) \sim MultiBernoulli(\theta)$ and $P(\theta|\alpha,\beta) \sim MultiBeta(\alpha,\beta)$. Our belief at node θ is then:

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

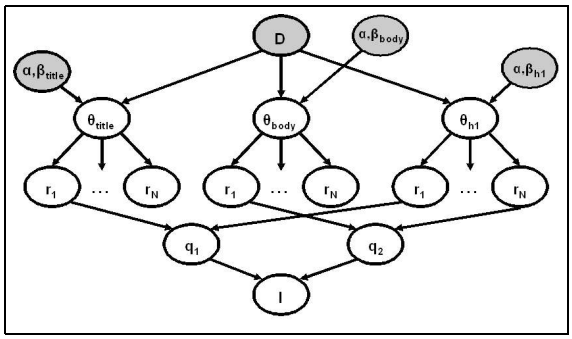

Figure 1: Indri's inference network retrieval model.

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 text of a document. Additionally, we estimate 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 Section 6.

3.3 Representation Nodes

The r_i nodes correspond to document features that can be represented in an Indri structured query. Indri implements all of the term and proximity operators available in INQUERY, including single terms, #N (ordered window N), and #uwN (unordered window N). See [9] 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 [15] 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.

3.4 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 [9] 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 [9]. In fact, it can be shown that the Indri query #combine($q_1 \ldots q_N$) using the estimates just described returns exactly the same ranked list as the query $q_1 \ldots q_N$ using the traditional (multinomial with Dirichlet smoothing) query likelihood 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 Track (6 of these machines were used) at a total cost of $9000 USD.

4 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.

5 Indri Retrieval Engine

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 Track as a proving ground for this engine.

The indexing algorithm is quite similar to the one described in [6], 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 [6]. This technique dramatically improved our indexing times.

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 compressed version of the corpus text, including byte offsets of indexed terms

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.

5.1 Anchor Text

For the anchor text run, we followed the model presented by Ogilvie and Callan in [11]. 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 (where server identity is defined by DNS name), and L_E consists of those documents on other servers. Let $A(d)$ represent the anchor 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.

5.2 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 statistics for terms.

In the results we reported for the conference, 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 [14]. 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.

6 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 [8] 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 ... q_N )
         0.5 #combine( e_1 ... 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 [2]: *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. #uw$4|s|$).

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:

$$\texttt{\#weight(}\ 0.7\ Q_{orig}\ 0.3\ \texttt{\#combine(}\ e_1 \ldots e_{10}\ \texttt{)}\ \texttt{)}$$

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 [4, 5, 7]. 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:

$$\begin{aligned}\texttt{\#weight(}\ & 0.15\ Q_{inlink} \\ & 0.25\ Q_{title} \\ & 0.10\ Q_{heading} \\ & 0.50\ Q_{mainbody}\ \texttt{)}\end{aligned}$$

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 [4, 5].

Run ID	AvgP	P10
indri04QL	0.2515	0.4959
indri04QLRM	0.2531	0.4714
indri04AW	0.2686	0.5735
indri04AWRM	0.2838	0.5510
indri04FAW	0.2787	0.5837

Table 2: Official submission results.

Table 1 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 421 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.

7 Results

The results from our official runs are given in Table 2. In the table, AvgP denotes mean average precision and P10 is the precision at 10 retrieved documents. The results seem to indicate that both phrase expansion and query expansion improve performance, where our best run, indri04AWRM, uses both. In fact, the indri04AWRM run was the best automatic, title-only run at the track.

After our official runs were submitted we discovered a number of serious bugs in Indri. Since the code was relatively young at the time and largely untested, this was not unexpected. After the bugs were fixed, we decided to carry out our runs again and see what impact the fixes had on effectiveness. Note that we did not change anything else in the system such as parsing, weights, smoothing parameters, query formulation methodologies, etc. Only core bug fixes were applied to the system. Also, since many other groups used other fields from the TREC topics, such as the description and narrative fields, we decided to run experiments using these fields.

Table 3 summarizes the results using the corrected system and various combinations of fields from the TREC topics. For the QL and QLRM runs, all of the text from all the fields under consideration is concatenated and included in the query with equal weighting. For example, for the title+desc run, the query formulation is:

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.

	Title		Title+Desc		Title+Desc+Narr	
	AvgP	P10	AvgP	P10	AvgP	P10
QL	0.2565	0.4980	0.2730	0.5510	0.3088	0.5918
QLRM	0.2529	0.4878	0.2675	0.5673	0.2928	0.5796
AW	0.2839	0.5857	0.2988	0.6184	0.3293	0.6306
AWRM	0.2874	0.5653	0.2974	0.6102	0.3237	0.6367

Table 3: Summary of corrected results using different combinations of TREC topic fields to construct queries.

$$\texttt{\#weight(}\ 1.0\ Q_{QL,title}\ 1.0\ Q_{QL,desc}\ \texttt{)}$$

where $Q_{QL,title}$ is the QL (bag of words) formulation of the text in the title field and $Q_{QL,desc}$ is the QL formulation of the text in the description field.

For the AW and AWRM runs, a phrase expansion query is constructed using only the title portion of the topic. The text from any additional fields is included in the query and given equal weight. Here is an example of the title+desc+narr query formulation:

$$\texttt{\#weight(}\ 1.0\ Q_{AW,title}\ 1.0\ Q_{QL,desc}\ 1.0\ Q_{QL,narr}\ \texttt{)}$$

where $Q_{AW,title}$ is the AW (phrase expansion) formulation of the text in the title field and $Q_{QL,desc}$ and $Q_{QL,narr}$ are the QL formulations of the text in the description and narrative fields, respectively.

Based on the corrected results, we see that the phrase expansion technique still yields improvements, but the query expansion runs lead to degraded performance. One explanation for the poor performance of the query expansion runs is the fact that our query expansion parameters were trained using the "broken" code. Another possible explanation is that the noise inherent in a collection the size of GOV2 produces poor expansion terms. We plan to further investigate this matter in the future.

Furthermore, the results show that naively using the description and narrative portions of the TREC topics can lead to further improvements. In fact, the corrected AW run using the title, description, and narrative fields outperforms the overall best official Terabyte Track run by approximately 7% (the run, uogTBQEL, submitted by the University of Glasgow, had an average precision of 0.3075 and made use of the same topic fields). Although web users have been trained by commercial search engines to formulate short keyword queries, they do so at the expense of precision. Considering more complex queries can lead to significant improvements in both average precision and precision at 10 documents. We wish to explore the issue of complex queries, including how to best express and represent them, in the future.

8 Conclusions

This year's Terabyte Track provided a forum to evaluate how well existing retrieval architectures and models scale to (almost) terabyte sized collections. Based on the outcome of the track, we are confidant the Indri search engine is both efficient and effective on such large scale collections. Distributed across six machines, Indri indexed the 25 million document, 426GB GOV2 collection in 6 hours (approximately 12 GB/hr/machine) and processed approximately one query every second. In terms of effectiveness, phrase expansion via Indri's structured query operators proved to be a powerful asset. In the official runs, Indri had the best title only run using this technique. Despite all of this, we hope to improve our system for next year. There are a number of things we aim to explore, including faster indexing, improved query processing times, more use of HTML document structure, looking into further use of complex queries, more effective query expansion techniques for noisy data, document priors, and link analysis.

9 Acknowledgments

This work was supported in part by the Center for Intelligent Information Retrieval and in part by Advanced Research and De-

velopment Activity and NSF grant #CCF-0205575. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor.

References

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

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

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

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

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

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

[7] 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.

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

[9] 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.

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

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

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

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

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

[15] 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.