Experiments Using the Lemur Toolkit

Paul Ogilvie and Jamie Callan
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{pto, callan}@cs.cmu.edu

Introduction

This paper describes experiments using the Lemur toolkit. We participated in the ad-hoc retrieval task of the Web Track. First, we describe Lemur in the System Description section and discuss parsing decisions. In the Experimental Results section, we discuss the official Lemur run and its retrieval parameters and we also discuss several unofficial runs. Finally, we summarize and draw conclusions.

System Description

The Lemur Toolkit [Lemur] is an information retrieval toolkit designed with language modeling in mind. Information about the toolkit is available at http://www.cs.cmu.edu/~lemur/. The toolkit is being developed as part of the Lemur Project, a collaboration between the Center for Intelligent Information Retrieval (CIIR) at the University of Massachusetts and the Language Technologies Institute (LTI) at Carnegie Mellon University. Lemur is written in C++ and C for use under UNIX and Windows NT.

Index

Lemur supports two types of indexes: one storing a bag-of-words representations for documents, the other storing term location information. In addition to storing standard information (document lengths, document frequency, collection term frequency, etc.) a user can build auxiliary files storing information useful for language modeling algorithms. For example, smoothing support files enable efficiency comparable to that of standard IR algorithms.

Retrieval

Lemur currently supports several retrieval algorithms. The primary retrieval model is a unigram language-modeling algorithm based on Kullback-Leibler divergence [Cover and Thomas 1991], also known as relative entropy. Also included is the OKAPI retrieval algorithm [Walker et al. 1998] and a dot-product function using TF-IDF weighting [Zhai 2001]. Our official submission was based on the Kullback-Leibler (KL) divergence algorithm.

The KL-divergence algorithm is derived as follows. Documents are ranked according to the negative of the divergence of the query's language model from the document's language model. Let θ_Q be the language model for the query Q and θ_D be the language model for document D. The documents are ranked by $-\mathrm{D}(\theta_Q \| \theta_D)$, where the function D denotes KL-divergence, as defined below.

$$\mathrm{D}(\theta_Q \| \theta_D) = \sum_w p(w|\theta_Q) \log \frac{p(w|\theta_Q)}{p(w|\theta_D)} \tag{1}$$

We assume that $p(w|\theta_D)$ has the following form:

$$p(w|\theta_D) = \begin{cases} p_s(w|\theta_D) & w \in D \\ \alpha_D p(w|\theta_C) & otherwise \end{cases} \tag{2}$$

where p_s is short for the probability given that the word was seen in the document, and α_d is a document dependent constant. α_d is used to reserve some portion of the probability distribution of the document's language model for words that are not present in the document. We also assume

$$p(w|\theta_Q, w \notin Q) = 0 \tag{3}$$

which is equivalent to stating

$$\sum_{w \in Q} p(w|\theta_Q) = 1 \tag{4}$$

Equation 4 states the sum of probabilities of all of the query terms is equal to 1. Equation 1 can be rewritten using properties of logarithms as

$$\sum_w p(w|\theta_Q)\log p(w|\theta_Q) - \sum_w p(w|\theta_Q)\log p(w|\theta_D) \tag{5}$$

Recall that we rank by $-D(\theta_Q \| \theta_D)$. Since the first summation of Equation 5 is independent of the document, we can ignore this during ranking, yielding a ranking function of:

$$\sum_w p(w|\theta_Q)\log p(w|\theta_D) \tag{6}$$

Given equations 2 and 7, this can be rewritten as

$$\sum_{w \in D \cap Q} p(w|\theta_Q)\log p_s(w|\theta_D) + \sum_{w \in \overline{D} \cap Q} p(w|\theta_Q)\log \alpha_D p(w|\theta_C) \tag{7}$$

We can add any number and subtract it and get the same results:

$$\sum_{w \in D \cap Q} p(w|\theta_Q)\log p_s(w|\theta_D) + \sum_{w \in \overline{D} \cap Q} p(w|\theta_Q)\log \alpha_D p(w|\theta_C) + \sum_{w \in D \cap Q} p(w|\theta_Q)\log \alpha_D p(w|\theta_C) - \sum_{w \in D \cap Q} p(w|\theta_Q)\log \alpha_D p(w|\theta_C) \tag{8}$$

Grouping the middle terms together and the first and last terms we get

$$\sum_{w \in D \cap Q} p(w|\theta_Q)\log \frac{p_s(w|\theta_D)}{\alpha_D p(w|\theta_C)} + \sum_{w \in Q} p(w|\theta_Q)\log \alpha_D p(w|\theta_C) \tag{9}$$

We can split the second summation by the terms in the logarithm. The cross-entropy of the query with the collection is a constant irrelevant to ranking. Using Equation 4, we can reduce the rest to:

$$\sum_{w \in D \cap Q} p(w|\theta_Q)\log \frac{p_s(w|\theta_D)}{\alpha_D p(w|\theta_C)} + \log \alpha_D \tag{10}$$

Equation 10 is the ranking equation implemented in Lemur.

As shown in [Lafferty and Zhai 2001], using the negative divergence of the query from the document is identical to the query likelihood model when using a maximum likelihood estimator for the query's language model. The query likelihood model was used in [Zhai and Lafferty 2001]. To see this, simply multiply equation 6 by the query length:

$$\sum_w c(w;Q)\log p(w|\theta_D) = \log \prod_w p(w|\theta_D)^{c(w;Q)} = \log P(Q|\theta_D) \propto P(Q|\theta_D) \tag{11}$$

where $c(w;Q)$ denotes the count of word *w* in the query *Q*.

Lemur can estimate $p_s(w|\theta_D)$ using maximum likelihood, a simple form of Jelinek-Mercer smoothing [Jelinek and Mercer 1985], Bayesian smoothing using Dirichlet priors [Berger 1985], and absolute discounting [Ney et al. 1994]. For TREC-10, we only considered Bayesian smoothing using a Dirichlet prior. The formula is given below.

$$p_s(w|\theta_D) = \frac{c(w;D) + \mu\ p(w|\theta_C)}{\mu + \sum_w c(w;D)} \tag{12}$$

where μ is a parameter and $c(w;D)$ denotes the count of word *w* in document *D*. The higher the value of μ, the more emphasis is placed on the collection language model. We estimate $p(w|\theta_C)$ using a maximum likelihood estimate:

$$p(w|\theta_C) = \frac{c(w;C)}{\sum_{w' \in C} c(w';C)} \tag{13}$$

The numerator is the collection term frequency of the word *w*, and denominator is simply the number of word occurrences in the collection.

Before we discuss parsing, we examine Lemur in terms of speed and space efficiency. We ran all tests on a computer with 2 GB RAM running Solaris. Table 1 displays the time spent on various indexing and retrieval tasks. The indexing speed of Lemur is about 2 GB per hour, including the time required to generate the smoothing support files. Table 2 shows the size of the built database. Currently, none of the data in Lemur is compressed by the index storing term locations, resulting in a database as large as the original data files. The bag-of-words index does compress data. Lemur stores a document posting list, which stores the words that occur in each document. This is not necessary for our retrieval algorithms after the smoothing support has been built, so it can be ignored or removed. If we do this, the size of database goes down from 10.3 GB to 5.2 GB.

Task	Time Required (hours:minutes:seconds)
Build Index	4:29:13
Generate Smoothing Support	0:30:55
Load Index	0:02:37
Run Queries (LM)	0:03:10
Run Queries (Okapi)	0:02:00

Table 1: Speed of Lemur

Component	MB	GB	Compression
Inverted index w/ locations	5110	5.0	none
Term / Doc Ids	130	0.1	none
Smoothing support	34	0.0	none
Total needed	5274	5.2	none
Doc posting list (sequential)	5232	5.1	none
Total	10506	10.3	none

Table 2: Database Size

Parsing

For document parsing, we used case-folding, the Porter stemmer [Porter 1980], and Inquery's stopword list (418 words) [Allan et al. 1980]. Possessive endings ("'s") were removed. We removed HTML tags and ignored text in script tags and HMTL comments.

We also used a very simple acronym recognizer that converts acronyms such as "F.H.A.", "FHA's", "FHAs", and "F.H.A.'s" into "FHA". We do not recognize acronyms containing numbers. If an uppercase word is found in text, it is checked against an acronym list. If the word is in the list, it is indexed uppercase. Otherwise it is converted to lowercase. To generate an acronym list, we assumed that text found in uppercase was generated by one of two language models. That is, an uppercase word is generated by either a language model describing general English or a model for special uppercase words. This is quantified by the following equation.

$$p(w \mid uppercase) = \lambda\ p(w \mid \theta_C) + (1-\lambda) p(w \mid \theta_{ucase}) \tag{14}$$

We empirically set λ equal to 0.9. We estimate $p(w \mid \theta_C)$ as above and $p(w \mid uppercase)$ using a maximum likelihood estimate based on all uppercase words in the corpus. We then use the Expectation Maximization algorithm to recover $p(w \mid \theta_{ucase})$. From this we choose the acronym list by selecting all words occurring at least 10 times and where $(1-\lambda) p(w \mid \theta_{ucase}) > p(w \mid \theta_C)$. This gives a rather long list of 9,863 words for the WT10g corpus. We inspected this list manually. It looked reasonable, but it mostly consisted of 3 or 4 letter acronyms that would not be mistaken as words, such as "UMCP" and "AFROTC". However, it did contain some highly desirable words, such as "AIDS" and "US". An alternative to automatically generating an acronym list would be to compile a shorter list by hand that contained known problem acronyms, like "AIDS".

Query parsing was done almost identically to document parsing. Additionally, if a word occurred more times in the index in uppercase than in lowercase, the uppercase word was added to the query. This was done because sometimes users submit queries containing lowercase forms of the acronyms (e.g., "aids" instead of "AIDS"). We used no query expansion.

Experimental Results

We only submitted one official run, given the time constraints and youth of the toolkit. The official Lemur run used the toolkit's unigram language modeling retrieval algorithm, described above. We also present an unofficial Okapi run, in hopes that it will be interesting and demonstrate the validity of our Okapi implementation.

Official Run

For our official run, we use Dirichlet prior smoothing with a μ of 1600. We set this parameter empirically by tuning it to the WT10g data set using queries constructed from the web TREC9 topics. Figure 1 compares Lemur at relevant retrieved at 100 to the best and median systems. The Lemur run is marked with a cross. The median performance for each query is marked with a plus. The bar marks the best performance for the query.

The graph shows that for most queries the performance of the Lemur run was well above the median performance. For the few runs where Lemur's performance was below the median, the Lemur run was usually close to the median performance. Since "who and whom" were in our stopword list, we retrieved no documents for query 542.

Table 3 is a more quantified description of Lemur's performance. For precision at 100, precision at 1000, and average precision we counted the number of queries where Lemur performed the best (not counting cases where the median performance was also the best), better than the median (but not the best), equal to the median, and worse than median performance. From this table, it is easy to see that the performance of Lemur on the majority of the queries is above the median performance. Table 4 at the end of the paper shows recall and precision measures.

Unofficial Runs

In order to validate Lemur's implementation of Okapi, we present an unofficial Okapi run using Lemur. For this run, we do not use any pseudo-relevance-feedback. The parameters we use are those suggested by Walker et al. [1998]: avdl = 900, b = 0.75, k1 = 1.2, and k3 = 1000. Note that these parameters are probably not optimal for the web documents. Figure 2 and Table 3 illustrate that our OKAPI implementation performs reasonably well. Table 4 at the end of the paper shows recall and precision measures.

We also wanted to evaluate the effectiveness of the acronym list and recognizer. Only two queries had acronyms in their title, and both were lowercase in the original topic. Topic 526 was "bmi" and topic 538 was "fha". In both cases, the acronym was the only word. Our heuristic query parser added the uppercase acronyms to the original queries, resulting in "bmi BMI" and "fha FHA". For query 526, using acronyms had 14 relevant retrieved documents in the top 100, while the index without acronyms returned only 8 relevant documents in the top 100. Average precision dropped 42% when disabling acronyms. On the other hand, disabling acronyms worked better

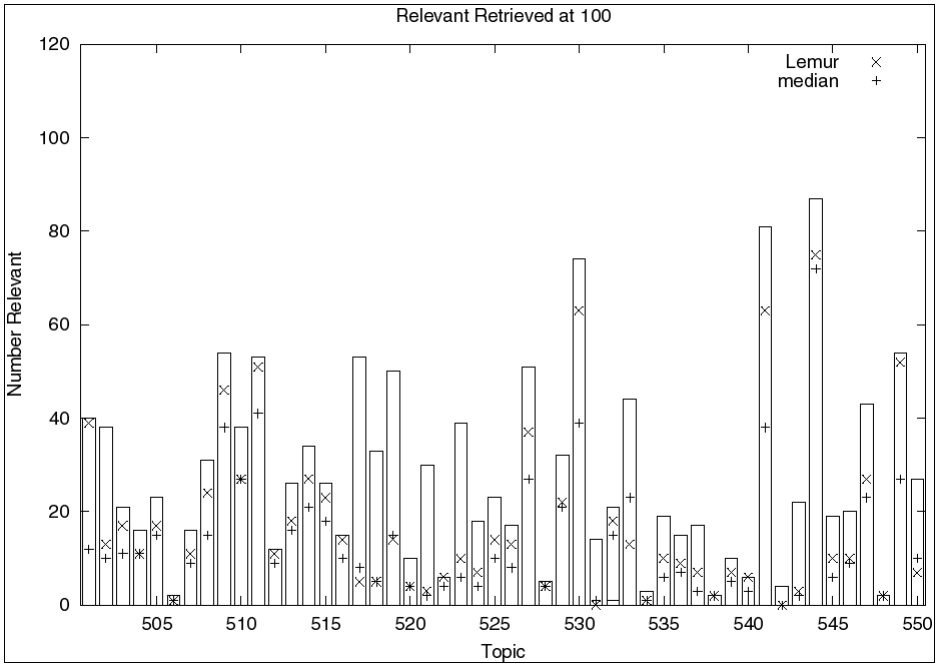

Figure 1: Relevant retrieved at 100 documents comparing Lemur with other systems

	Official (Lemur) Language Modeling			Unofficial Okapi		
	Precision at 100	Precision at 1000	Average Precision	Precision at 100	Precision at 1000	Average Precision
best && !median	2	2	2	0	0	0
> median && !best	33	32	43	33	26	38
= median	10	11	1	9	15	0
< median	5	5	4	8	9	12

Table 3: Performance of Lemur compared with other systems

for query 538. Both precision at 100 and 1000 were identical, but disabling the acronyms caused the average precision to rise 164%. All other queries performed almost identically whether acronyms were disabled or enabled.

Conclusions

Our primary goal was to demonstrate experiments using the new open-source Lemur toolkit to the Information Retrieval community on a commonly used research collection. Given Lemur's official run, simple language modeling techniques compare well with other TREC systems. This is particularly pleasing, given the lack of automatic query expansion or pseudo-relevance feedback.

Acknowledgements

We thank Thi Nhu Truong for assistance with the Lemur toolkit and getting it running on large data sets. We thank Chengxiang Zhai for help with Lemur and the paper. We thank John Lafferty for contributions to the toolkit and the paper. This work was sponsored in full by the Advanced Research and Development Activity in Information Technology (ARDA) under its Statistical Language Modeling for Information Retrieval Research Program. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect those of the sponsor.

References

[Allan et al. 2000] J. Allan, M. Connell, W.B. Croft, F.F. Feng, D. Fisher, X. Li. INQUERY and TREC-9. In *TREC-9*, 2000, 504-513.

[Berger 1985] J. Berger. Statistical Decision Theory and Bayesian Analysis. New York: Springer-Verlag, 1985.

[Cover and Thomas 1991] T.M. Cover and J.A. Thomas. Elements of Information Theory. New York: Wiley, 1991.

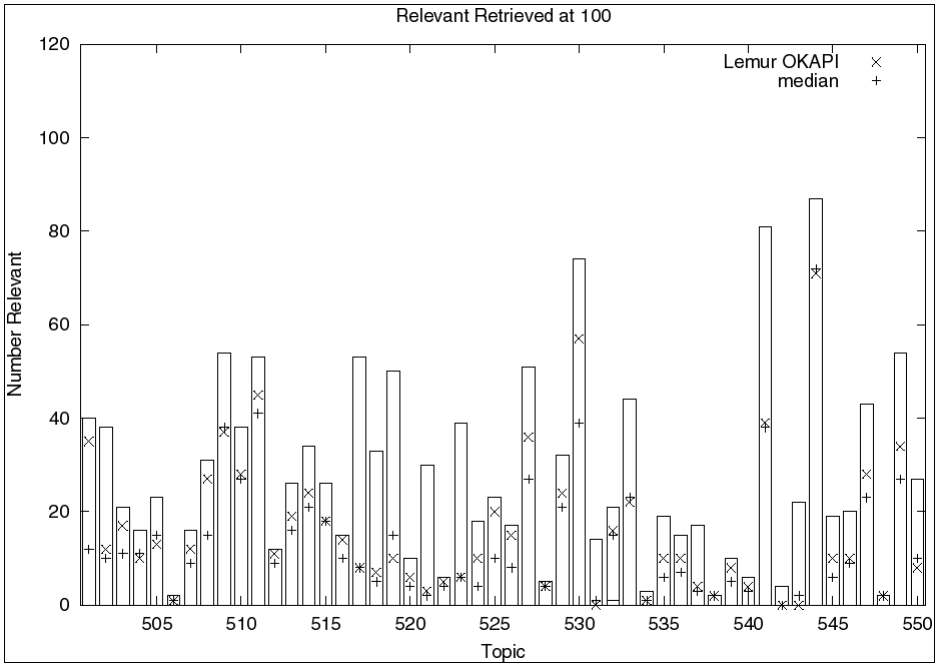

Figure 2: Relevant retrieved at 100 documents comparing Lemur's OKAPI with other systems

[Jelinek and Mercer 1985] F. Jelinek and R. Mercer. Probability Distribution Estimation from Sparse Data. *IBM Technical Disclosure Bulletin 28*, 1985, 2591-2594

[Lafferty and Zhai 2001] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 2001 ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01)*, 2001.

[Lemur] The Lemur Toolkit. http://www.cs.cmu.edu/~lemur

[Ney et al. 1994] H. Ney, U. Essen, R. Kneser. On structuring probabilistic dependencies in stochastic language modeling. In *Computer Speech and Language*, 8, 1994, 1-38.

[Porter 1980] M. Porter. An algorithm for suffix stripping. In *Program*, 14(3), July 1980, 130-137.

[Walker et al. 1998] S. Walker, S.E. Robertson, M. Boughamen, G.J.F. Jones, K. Sparck-Jones. Okapi at TREC-6: Automatic ad hoc, VLC, routing, filtering and QSDR. In *TREC-6*, 1998, 125-136.

[Zhai 2001] C. Zhai. Notes on the Lemur TFIDF model. http://www.cs.cmu.edu/~lemur/tfidf.ps

[Zhai and Lafferty 2001] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 2001 ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01)*, 2001.

	Official (Lemur) LM	**Unofficial** Okapi
Total number of documents over all queries		
Retrieved:	48667	48667
Relevant:	3363	3363
Rel_ret:	2400	2334
Interpolated Recall - Precision Averages:		
at 0.00	0.6894	0.6146
at 0.10	0.4018	0.4073
at 0.20	0.3106	0.3306
at 0.30	0.2773	0.2785
at 0.40	0.2355	0.2315
at 0.50	0.2072	0.1929
at 0.60	0.1342	0.1212
at 0.70	0.0965	0.0878
at 0.80	0.0718	0.0589
at 0.90	0.0437	0.0245
at 1.00	0.0138	0.0140
Average precision (non-interpolated) for all rel docs(averaged over queries)	0.1985	0.1950
Precision:		
At 5 docs:	0.3720	0.3440
At 10 docs:	0.3200	0.3300
At 15 docs:	0.3093	0.3187
At 20 docs:	0.2920	0.2920
At 30 docs:	0.2580	0.2573
At 100 docs:	0.1758	0.1630
At 200 docs:	0.1305	0.1183
At 500 docs:	0.0770	0.0736
At 1000 docs:	0.0480	0.0467
R-Precision (precision after R (= num_rel for a query) docs retrieved):		
Exact:	0.2299	0.2304

Table 4: Summary Statistics for Official and Unofficial Runs