IR Experiments with Lemur

Nancy McCracken

October 21, 2004

Adapted from a presentation in IST 657

Lemur assistance: Shuyuan Mary Ho

IR model slides: Liz Liddy and Anne Diekema

Example experiment: Sijo Cherian

Outline

- Overview of Lemur project goals and capabilities
- Standard IR experiments made possible by TREC
- Steps for IR using Lemur
  - Document preparation and indexing
  - Query preparations
  - Retrieval using several models
  - Other applications
- Evaluation for TREC experiments
- Example Experiment

- The Lemur Toolkit is designed to facilitate research in language modeling and information retrieval
- The toolkit supports
  - indexing of large-scale text databases,
  - the construction of simple language models for documents, queries, or subcollections, and
  - the implementation of retrieval systems based on language models as well as a variety of other retrieval models.

* A *Lemur* is a nocturnal, monkey-like African animal that is largely confined to the island of Madagascar. "Lemur" was chosen for the name of the UMass-CMU project in part because of its resemblance to LM/IR. (The fact that the language modeling community has until recently been an island to the IR community is also suggestive.) (Lemur documentation).

Lemur Facts

- Written in C++ for Unix platforms, but also runs on Windows. Includes both
  - API to program your own applications or modules
  - User applications to perform IR experiments without programming
    - Note that Lucene has an API but lacks user applications
- Maintained by UMASS and CMU
- Public forum for discussion, also invites code submissions

Lemur Features

- From the Documentation Overview at the Lemur Home page
  - http://www-2.cs.cmu.edu/~lemur/
- Indexing:
  - English, Chinese and Arabic text
  - word stemming (Porter and Krovetz stemmers)
  - omitting stopwords
  - recognizing acronyms
  - token level properties, like part of speech and named entities
  - passage indexing
  - incremental indexing

More Lemur Features

- Retrieval:
  - ad hoc retrieval
    - TFIDF (vector model)
    - Okapi (probabilistic model)
    - relevance feedback
  - structured query language
    - InQuery (Boolean queries weighted from other models)
  - language modeling
    - KL-divergence
    - query model updating for relevance feedback
    - two-stage smoothing
    - smoothing with Dirichlet prior or Markov chain

More Lemur Features

- Distributed IR (using multiple indexes):
  - query-based sampling
  - database ranking (CORI)
  - results merging (CORI, single regression and multi-regression merge)
- Summarization and Clustering
- Simple text processing
- CGI script and stand-alone GUI (written in Java Swing) for retrieval
  - Provides a user interface to submit single queries with a prepared index
  - Under development

Text REtrieval Conference (TREC)

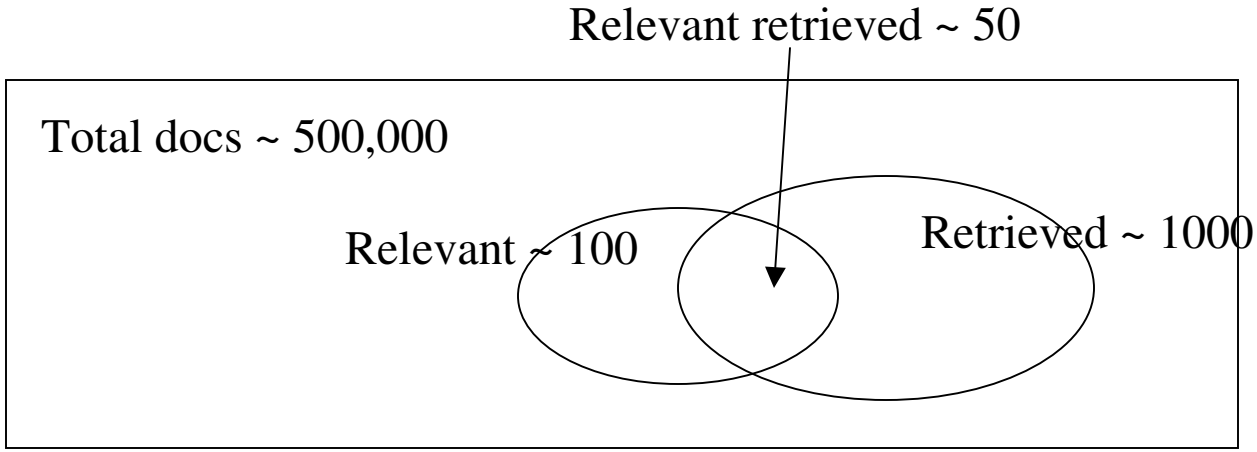
- NIST provides the infrastructure for IR experiments
  - Large data collections
    - protected by intellectual property rights, available for research purposes
  - Human evaluators for yearly experiments
- Results of experiments presented at yearly workshops, since 1992
  - 93 groups from 22 countries participated in 2003
  - Standard IR experiments were in the Ad Hoc track
    - Current tracks include Cross Language, Filtering, Question Answering, Robust Retrieval, Terabyte, Video
- http://trec.nist.gov
  - co-sponsored by the National Institute of Standards and Technology (NIST) and U.S. Department of Defense

TREC retrieval evaluation measures

- Evaluation based on relevance:
  - TREC's definition:
    - "If you were writing a report on the subject of the topic and would use the information contained in the document in the report, then the document is relevant. "
  - TREC relevance judgments are binary: relevant or not (1 or 0)
  - Human relevance judgments
- Evaluation measures based on precision and recall
  - For each query:
    - Recall: number of relevant retrieved docs / total relevant docs
    - Average Precision at seen relevant docs (includes ranking)
      - (Standard precision: number of relevant retrieved docs / total retrieved docs)
    - Average Precision versus standard Recall levels
  - Total result: compute the average precision and recall over all queries
    - Micro-average: total precision/recall divided by the number of queries

TREC evaluation process

- Problem is to identify the relevant documents for each query by humans:



- Solution is pooling: take all retrieved documents from all groups in the evaluation to form a "pool". Human evaluators judge those documents.
  - Solution has proved to be effective

TREC data

- Document collection
- For each Ad Hoc track evaluation:
  - Queries, describing information need. Includes
    - Title
    - Description
    - Narrative
  - Relevance judgments
    - Includes entire pool
      - 0 for non-relevant
      - 1 for relevant

Goal

- Use Lemur for IR experiments using TREC data

Recall "Steps in the IR Process"

- Documents
  1. Normalize document stream to predefined format.
  2. Break document stream into desired retrievable units.
  3. Isolate & meta-tag sub-document pieces.
  4. Identify potential indexable elements in documents.
  5. Delete stop words.
  6. Stem terms.
  7. Extract index entries.
  8. Compute weights.
  9. Create / update inverted file.

Lemur accepts TREC formatted documents and provides capabilities for (most of) the other steps in indexing applications.

Document Preparation and Initial Steps

- Lemur takes TREC document files and breaks each one into individual “documents” based on the <DOC> formatting tags in the files.
  - These are newswire documents.
- Lemur provides a standard tokenizer (called a parser) that has options for stemming and stopwords
  - Stemming: either Porter and Krovetz are standard strong stemmers
  - Stopwords: user provides a stopwords file
    - We will start with the stopwords file from the SMART retrieval project at Cornell.

Indexing Application Programs

- The main differences between the indexers provided by Lemur are
  - Whether the index keeps word position information
  - Performance characteristics

Index Name	File Exten-sion	File Limit	Stores Positions	Loads Fast	Disk Space Usage	Increment al
InvIndex	.inv	no	no	no	less	no
InvFPIndex	.ifp	no	yes	no	more	yes
KeyfileIncIndex	.key	no	yes	yes	Even more	yes
BasicIndex (deprecated)	-	yes	-	-	-	-

Setting Up Indexing

- Command line invocation of indexing application, specifying parameter file
    - **BuildKeyfileIncIndex** build_param
- Parameters include:
  - Name of the index
    - index = pindex;
  - File with a list of document files to process:
    - dataFiles = filelist;
  - Stemming and stop word options:
    - stemmer = porter;
    - stopwords = stopwords.dat;
- As a result of this command, approximately 10 files are produced, starting with the index name.
  - Inverted index is not humanly readable
  - Use query processor to view tokenization, stopping and stemming on a document.

Recall "Steps in the IR Process"

- Queries
  1. Recognize query terms vs. special operators.
  2. Tokenize query terms.
  3. Delete stop words.
  4. Stem words.
  5. Create query representation.
  6. Expand query terms.
  7. Compute weights.

  ----------> Matcher

- Lemur has Parse applications that create the query representation
- Matching is provided with a number of Retrieval applications

Query Representation: Keywords

- Lemur Parse application provides a keyword representation of the query
  - Queries in a file in the same format as documents
    - Must choose which parts of the TREC queries to include and add DOC tags
  - Tokenization, stemming and stop word options are provided by Lemur
- Command line
    - **ParseToFile** query_param queries.sgml
  - Parameter options
    - outputFile = query.lemur.in;
    - stemmer = porter;
    - stopwords = stopwords.dat;

Retrieval Applications
for queries represented as keywords

- General purpose retrieval for the three models: vector, probabilistic and language
  - Command: **RetEval** retrieval_param
  - Option to choose retrieval model
    - retModel = 0; /* 0 = TFIDF; 1 = Okapi; 2 = KL-divergence */
  - Other parameters
    - index = pindex.key; /* database index */
    - textQuery = query.lemur.in; /* query text stream */
    - resultFile = retrieval_file; /* result file */
    - resultCount = 1000; /* how many docs to return as the result */
    - resultFormat = 1; /* 0 = simple-format; 1 = TREC-format */
  - Options for pseudo-relevance feedback
    - feedbackDocCount: the number of docs to use for pseudo-feedback
      - (0 means no-feedback)
    - feedbackTermCount: the number of terms to add to a query when doing feedback.

TFIDF: Vector Space Model

- Standard formula for *tf-idf* weight of a term *i* in document *j*

$$w_{ij} = \frac{(freq_{ij} + 1)}{(length_j)} * \log\left(\frac{N}{n_i}\right)$$

- Simularity between document vector d_j and query vector q

$$sim(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

- Lemur also allows for some probabilistic weighting to be added to the vector similarity. Details are found in
  - Chengziang Zhai, Notes on the Lemur TFIDF model. http://www-2.cs.cmu.edu/~lemur/tfidf.ps

TFIDF Model parameters

- Pseudo-feedback parameter
  - feedbackPosCoeff: the coefficient for positive terms in (positive) Rocchio feedback. Only the positive part is implemented and non-relevant documents are ignored.
- Additional weighting parameters (for documents and queries)
  - doc.tfMethod: document term TF weighting method:
    - 0 for RawTF, 1 for log-TF, and 2 for BM25TF
  - doc.bm25K1: BM25 k1 for doc term TF
  - doc.bm25B : BM25 b for doc term TF
  - query.tfMethod: query term TF weighting method:
    - 0 for RawTF, 1 for log-TF, and 2 for BM25TF
  - query.bm25K1: BM25 k1 for query term TF.
  - bm25B is set to zero for query terms

Okapi*: Probabilistic Model

- Robertson/Sparck-Jones weight (BM1)
  - Where R is # rel docs, r is # rel docs term k, N is #docs, n is #docs term k

$$w = \log \frac{(r+0.5)/(R-r+0.5)}{(n-r+0.5)/(N-n-R+r+0.5)}$$

- Latest formula (BM25)

$$\sum_{T \in Q} w \frac{(k_1 + 1)tf}{K + tf} \frac{(k_3 + 1)qtf}{k_3 + qtf}$$

$$K = k_1((1-b) + b\frac{dl}{ave.dl})$$

* An *Okapi* is a nocturnal, giraffe-like African animal that was unknown to zoologists until the 20th century. Apparently, "Okapi" in the City University of London IR system originally stood for "Online Keyword Access to Public Information." (attributed to Stephen Robertson in the Lemur documentation).

Okapi model parameters

- Probability parameters
  - BM25K1 : BM25 K1
  - BM25B : BM25 B
  - BM25K3: BM25 K3
  - BM25QTF: The TF for expanded terms in feedback (the original paper about the Okapi system is not clear about how this is set, so it's implemented as a parameter.)
- For more information on model parameters, check resources
  - IST 657 class notes
  - K. Sparck Jones, S. Walker and S. Robertson, A probabilistic model of information retrieval: development and comparative experiments, (Part 1). *Information Processing and Management*, 36, pp. 779-808, 2000.
  - Okapi web site: http://www.soi.city.ac.uk/~andym/OKAPI-PACK/

KL-divergence: Language Modeling

- In general, each document has its own language model. The retrieval problem is then

  $$P(Q|D)$$

  the probability that the query was generated by the document.
- Lemur uses a unigram language model algorithm based on Kullback-Leibler divergence, also known as relative entropy.
  - Documents are ranked according to the negative of the divergence of the query language model from the document language model
  - But like the original maximum likelihood estimate of P(Q|D), KL-divergence involves the probability, for each term t in the query:

    $$P(t|D) = P_s(t|D) \text{ if the term is in the document}$$
    $$= a_D P(t|C) \text{ otherwise}$$

    where P_s is the smoothed probability of the term seen in the document
    P(t|C) is the probability of the term in the collection
    and a_D is the constant controlling how much probability to give unseen words

Smoothing Parameters

- Parameter for smoothing support file – this file is generated by the application GenerateSmoothSupport after indexing and computes collection statistics to increase the performance of retrieval
  - smoothSupportFile: The name of the smoothing support file
- Three smoothing functions
  - smoothMethod: One of the three: Jelinek-Mercer (0), Dirichlet prior (1), and Absolute discounting (2)
  - smoothStrategy: Either interpolate (0) or backoff (1)
  - JelinekMercerLambda: The collection model weight in the JM interpolation method. Default: 0.5
  - DirichletPrior: The prior parameter in the Dirichlet prior smoothing method.
    - Default: 1000
    - Can be computed using the application EstimateDirPrior
  - discountDelta: The delta (discounting constant) in the absolute discounting method. Default 0.7.
- Chengxiang Zhai and John Lafferty, A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval, SIGIR 2001.

Relevance Feedback (all models)

- The relevance feedback retrieval application is implemented by a simplified Rocchio feedback (see Zhang Lemur notes)
    - RetFBEval feedback_param
- Uses a relevance judgment file (that you provide for your document collection and query set) to provide relevance feedback.
  - feedbackDocuments: judgment_file
  - Format of this file is slightly different from standard TREC qrels files: must only have relevant file names and 3 columns
- Has similar options for the three models as RetEval.

Relevance Feedback for Language Modeling

- In language modeling, relevance feedback is achieved by using relevant judgment files to redefine the query model.
- Lemur implements this as a separate application
  - GenerateQueryModel
  - Input is the index, the queries and judgment files
  - Output is a query model file.
  - Input may also be a query model file, so iterative feedback is possible.
- Then there is a retrieval method using KL-divergence on the revised query model
  - QueryModelEval
  - Input is query model
  - Output is retrieval results

Structured Queries

- In Lemur, structured queries have boolean operators that can be used with keywords.
  - Includes AND, OR, NOT, proximity operators (both ordered and unordered) and weighted sum.
    - AND(t_1, . . . , t_n)
    - WSUM (w_1 t_1, . . . , w_n t_n)
- Based on the InQuery query language
  - Inquery rules for each structured operator to compute weight based on weights of individual terms
    - $P_{AND} = p_1 * \ldots * p_n$
    - $P_{WSUM} = (w_1*p_1 + \ldots + w_n*p_n) / (w_1 + \ldots + w_n)$
  - James Callan, Bruce Croft, Stephen Harding, The INQUERY Retrieval System, Proceedings of the 3rd International Conference on Database and Expert Systems Applications, 1992.

Structured Query Retrieval

- User must provide boolean representation of the query
  - Could also use operators to represent query expansion
  - Command to process query
    - ParseInQueryOp struct_query_param queryfile
  - Options are the same as for the general RetEval
- Retrieval application has the same options as the general RetEval, including the model to use with the boolean ops
    - StructQueryEval struct_param

TREC Evaluation

- Lemur includes a (perl) version of the standard trec_eval evaluation program, which you run with the judgments and retrieval results to produce an evaluation report.
    - ireval.pl -j qrels.trec8.adhoc.parts1-5 -trec < retrieval_file > results
- The evaluation report includes for each query a report on
  - Relevant docs,
  - Retrieved relevant docs,
  - Average (non-interpolated) precision,
  - Interpolated precision at recall levels
- Also produces a summary for all the queries.
- I have made a version that adds the computation of recall.

Query Clarity

- The clarity score is a way of measuring how well the query terms fit the document collection
  - QuerityClarity
    - Input is query model
    - Output is queries with clarity score
- Reference:
  - Steve Cronen-Townsend, Andres Corrada-Emmanuel, Bruce Croft, Predicting Question Effectiveness, SIGIR 2002.

Additional Applications

- Summarization
  - will use sentences if sentence separator tags are present in the input text, otherwise will just break text into chunks
  - Two algorithms to select summary sentences, a basic one and the Maximum Marginal Relevance (MMR) algorithm
- Clustering
  - Basic clustering algorithm over the index
    - Cosine similarity function
    - Can be used for TDT
  - Off-line clustering with k-means and bisecting k-means partitional clustering.
  - Probabilistic Latent Semantic Analysis (PLSA)
  - "A Comparison of Document Clustering Techniques", Michael Steinbach, George Karypis and Vipin Kumar. TextMining Workshop. KDD. 2000.

Example Experiment

- Conducted as part of an independent study by Sijo Cherian.
- Goal was to improve retrieval for QA by using NLP to include “concept” phrases as keywords and varying the forms and weights of the phrases.
  - Study primarily used QA track questions but also did some Ad Hoc
  - TREC 7/8 documents (528K docs indexed with 142 million terms)
  - TREC 8 queries, 401-450, description field

Structured Queries with Phrases

- Use Lemur structured query language to represent phrases and weights
  - Weighted sum gives a weight for each term (WSUM)
  - Ordered proximity - 1 gives adjacent terms (#1)
  - Unorder proximity – N gives within n terms (#uwN)
- Wrote program to tokenize, stem, remove stop words, and produce structured representation
  - Example (slightly simplified) shows phrase weights of 0.5, term weights of 1.0

What was the monetary value of the Nobel Peace Prize in 1989?

```
#q2 = #WSUM(  0.5 #1(monetary value)
              1 monetary
              1 value
             0.5 #1(Nobel Peace Prize)
              1 Nobel
              1 Peace
              1 Prize
              1 1989  );
```

Experimental Results

- Experiments were run using the TFIDF model, returning 50 documents.
- **Baseline:** Precision 0.2233, 2612 relevant documents retrieved out of 4728 relevant documents (recall 0.5524)

Table 1: Average Precision and Recall for TREC8 Ad-hoc Topics 401-450

Phrase Weight	**Proximity Parameter [Precision | Recall]**							
	Ordered 1		Unordered 5		Unordered 10		Unordered 50	
0.5	**.2298**	.5626	.2294	.5495	.2283	.5389	.2296	.5535
1.0	**.2301**	.5472	.2293	.5516	.2178	.5404	.2131	.5364

2.91% improvement on Precision, over the baseline