Lemur Tutorial

Chunye Wang; Bin An
ISM293 - Spring 09
What is Lemur?

- Lemur is a toolkit designed to facilitate research in language modeling and information retrieval.
- The system's underlying architecture was built to support the technologies such as ad hoc and distributed retrieval, with structured queries, cross-language IR, summarization, filtering, and categorization.
- They provide many useful sample applications, but have designed the toolkit to allow you to easily program your own customizations and applications.
What can Lemur do?

- The Lemur toolkit supports the construction of basic text retrieval systems using language modeling methods, as well as traditional methods such as vector space model and probabilistic model.
Lemur is particularly useful for researchers in language modeling and information retrieval who do not want to write their own indexers but would rather focus on developing new techniques and algorithms.

In addition to indexing, Lemur provides some baseline retrieval algorithms, such as Okapi and KL Divergence for use and comparisons.

You can use Lemur to build your own search systems.

Lemur has implemented and included basic ad hoc IR, distributed IR, IR using structured queries, IR using distributed indexes, clustering documents, and summarization.
How can we use Lemur?

- Lemur has many applications for indexing and retrieval that are fully functional for many purposes, so you can use them “out of the box”.
- In addition, since Lemur was written to facilitate research on LM and IR, the design allows you to try out new retrieval methods by using abstract interfaces, or writing new applications based on existing methods.
- The source code is provided to encourage users to modify the toolkit in support of their own research, development, or teaching activities.
What does Lemur come with?

- Lemur comes with all the source code and makefiles necessary to build the libraries for indexing and retrieval. For windows, you can download the pre-compiled libraries and executables.
- Lemur currently supports the following features:

**Indexing:**
- English, Chinese and Arabic text
- word stemming (Porter and Krovetz stemmers)
- stopwords
- recognizing acronyms
- passage indexing
- incremental indexing
- in-line and offset annotation support
What does Lemur come with?

- Retrieval:
  - ad hoc retrieval (TF-IDF, Okapi, and InQuery)
  - passage retrieval
  - cross-lingual retrieval
  - language modeling (KL-divergence)
    - query model updating for pseudo feedback
    - two-stage smoothing
  - relevance feedback
  - structured query language
What does Lemur come with?

- Distributed IR:
  - query-based sampling
  - database ranking (CORI)
  - results merging (CORI, single regression and multi-regression merge)

- Document Clustering
- Summarization
- Simple text processing
Lemur internals

- Lemur was written primarily in C++
- The GUI is written with Java
- It is compatible with Windows XP, Linux (including x86_64), Mac OS/X and Solaris-based systems.
Installing & Compiling Lemur

- How to download, install and compile

- There are pre-compiled binaries for Windows. For other platforms, you can download the sources and compile.

- **Lemur Wiki** answers most of beginner’s questions on installing, compiling, indexing, retrieval, programming...
  - [http://www.lemurproject.org/](http://www.lemurproject.org/) (the old website)
Indexing using Lemur

- The Lemur Toolkit can inherently index several types of documents (html, xml, trecweb, trectext, doc, ppt, pdf...)
- The two most common types are TREC text and TREC web.
- If you have a set of plaintext documents that you wish to index, one of the easiest ways to prepare the documents is to use a PERL script to iterate through the documents adding TREC tags for <DOC> and a unique <DOCNO>
The most common document is a plaintext document in TREC format.

A document in TREC format must have a `<DOC>` tagset surrounding the document.

The document must also at a minimum include a `<DOCNO>` tagset enclosing the document ID and a `<TEXT>` tagset enclosing the text to be indexed.

As an example:

```
<DOC>
  <DOCNO>document_id</DOCNO>
  <TEXT> Index this document text. </TEXT>
</DOC>
```
## Index Types

The Lemur Toolkit currently supports two indexing types.

### Indexing

<table>
<thead>
<tr>
<th>Feature</th>
<th>Keyfile Index</th>
<th>Indri Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexable Document Formats</td>
<td>TREC Text; TREC Web; HTML</td>
<td>TREC Text; TREC Web; HTML; Plain Text; XML; PDF; MBox; Microsoft Word and PowerPoint (Windows-only)</td>
</tr>
<tr>
<td>Stored Metadata</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fields / Annotations support</td>
<td>No</td>
<td>Yes. Can be either in-line or in the form of offset annotations</td>
</tr>
</tbody>
</table>

### Retrieval

<table>
<thead>
<tr>
<th>Feature</th>
<th>Keyfile Index</th>
<th>Indri Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Language</td>
<td>Uses an implementation of the InQuery Query Language</td>
<td>Uses the Indri Query Language</td>
</tr>
</tbody>
</table>

### Applications

<table>
<thead>
<tr>
<th>Feature</th>
<th>Keyfile Index</th>
<th>Indri Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Building</td>
<td>BuildIndex</td>
<td>IndriBuildIndex or BuildIndex</td>
</tr>
<tr>
<td>Batch Retrieval</td>
<td>RetEval</td>
<td>IndriRunQuery or RetEval</td>
</tr>
</tbody>
</table>
Indri

- A text search engine developed by UMass; a part of Lemur toolkit.
- Indri has different classes, functions, API from lemur.
- Indri can index fields information.
- A useful tutorial about Indri:
  - http://ciir.cs.umass.edu/~strohman/indri/
Using Lemur

- Lemur provides:
  - Parameter files to build your system
  - Application Programming Interface (API)
  - Graphical User Interface (GUI)
Writing parameter file to index

- Creating a parameter file
  After your documents are prepared, you should create a parameter file that will tell the indexer how to index.
  - “application + parameter file” combination
  - A sample parameter file (“paramFile.txt”) looks like:

```xml
<parameters>
  <dataFiles>D:\Program Files\Lemur\bin\datafileList.txt</dataFiles>
  <index>D:\Program Files\Lemur\data\index</index>
  <indexType>key</indexType>
  <memory>512M</memory>
  <docFormat>trec</docFormat>
  <stopwords>D:\Program Files\Lemur\bin\stoplist.txt</stopwords>
  <countStopWords>true</countStopWords>
  <stemmer>porter</stemmer>
</parameters>
```

* datafileList.txt is not the data to be indexed. If you want to index 1.dat and 2.dat, write the paths, e.g., “C:\1.dat” and “C:\2.dat” in separate line in datafile.txt
Building an Index

- Using BuildIndex.exe application
- Run the command
  - BuildIndex paramFile.txt
- This command creates the inverted index for your document collection using the options you specified
- More about indexing
  - http://lemur.wiki.sourceforge.net/Building+Indexes
Lemur API

- For special-purpose applications that are not implemented in the toolkit itself.
- Using Lemur classes at three different levels:
  - Utility level
  - Indexer level
  - Retrieval level
- API User's Guide
  http://www.lemurproject.org/lemur/api.php
Using Lemur API

- To use API with C/C++, please read this page:
  http://lemur.wiki.sourceforge.net/Using+the+Lemur+Toolkit+with+C+and+CPlusPlus

- To use API with Java, please read this page:
  http://lemur.wiki.sourceforge.net/Using+the+Lemur+Toolkit+with+Java

- To use the Lemur API on Linux, you will need both the library file and all the header files
Using Lemur API

- Lemur Toolkit Documentation
  - very important reference for coding
  - provides: source code, classes, functions usage, …
# Example 1- Gathering index statistics

```cpp
#include "Index.hpp"
#include "IndexManager.hpp"
using namespace lemur::api;
using namespace std;

void main(int argc, char *argv[]) {
  // we assume the index path is the first argument
  char *indexPath=argv[1];

  // open the index
  Index *theIndex=IndexManager::openIndex(indexPath);

  // get the count of documents
  int numDocuments=theIndex->docCount();
```
// get the average document length (in words)
float avgDocLength=theIndex->docLengthAvg();

// get the count of total terms
int totalTermCount=theIndex->termCount();

// get the count of _unique_ terms
int uniqueTermCount=theIndex->termCountUnique();

cout << "# documents: " << numDocuments << endl;
cout << "# terms: " << totalTermCount << endl;
cout << "# unique terms: " << uniqueTermCount << endl;

// finally delete the index
delete theIndex;
}
Example 2 – Dumping out term dictionary

```cpp
#include "Index.hpp"
#include "IndexManager.hpp"
using namespace lemur::api;
using namespace std;

void main(int argc, char *argv[]) {
    // we assume the index path is the first argument
    char *indexPath=argv[1];

    // open the index
    Index *theIndex=IndexManager::openIndex(indexPath);

    // get a the total number of unique terms
    // which will give us our max term ID
    int maxTermID=theIndex->termCountUnique();
}
```
Example 2 – Dumping out term dictionary

// now loop over the terms
for (int t=1; t <= maxTermID; t++) {
    // get the term as a string
    TERM_T thisTermString=theIndex->term(t);

    // get the TF of this term
    int termCount=theIndex->termCount(t);
    // get the document count for this term
    int docCount=theIndex->docCount(t);

    cout << t << \t << thisTermString << \t;
    cout << termCount << \t << docCount << endl;
}

// finally delete the index
delete theIndex;
More Examples

- http://lemur.wiki.sourceforge.net/Example+Applications+in+CPlusPlus

Conclusion

- Lemur is great! 😊

Questions?