NLP and IR

Building your first Search Engine with Lucene

Aliaksei Severyn
University of Trento, Italy

April 06, 2012
Plan for the lab

- Introduction to Lucene Search Engine
- Lucene concepts
- Hands-on experience with indexing and searching
  - HelloWorld example
- Using search engine to retrieve answer passages for Question Answering system
  - Indexing and searching 180k of QA corpus
What is Lucene

- software library for search
- open source
- not a complete application
- set of java classes
- active user and developer communities
- widely used, e.g., IBM and Microsoft.
High level overview

- Lucene is a full-text search library
- Designed to add search to your application
- Maintains a full-text index.
- Searches the index and returns results ranked by either the relevance to the query (or by an arbitrary field such as a document's last modified date.)
Typical architecture of a Lucene search app
Our first HelloWorld app with Lucene

- Create an in-memory index
- Add a few documents
- Construct a query
- Search an index
- Display results
Setting up our first example

Download Lucene sources and binary from:
http://www.apache.org/dist/lucene/java/3.5.0/

Or download everything from:
http://disi.unitn.it/~severyn/NLPIR.2012/lab01/intro.tar.gz

E.g. try the following in your terminal:

$ wget http://disi.unitn.it/~severyn/NLPIR.2012/lab01/intro.tar.gz
$ tar xvfz lab01.tar.gz
$ cd lab01
$ javac -cp .:lucene-core-3.5.0.jar HelloLucene.java
$ java HelloLucene
Setting up the project in Eclipse/Netbeans

- Create a new project NLPIR
- Drag HelloLucene.java src file to the project
- Go to project properties->Libraries
- Click on “Add External Jars…”
- Locate lucene-core-3.5.0.jar
- To enable documentation add path to the Javadoc
Create a new project
Name your project
Drag HelloLucene.jar to the src folder
Add lucene-core-3.5.0.jar
Run your first Lucene app!

```java
w.addDocument(doc);
}
```

Found 2 hits.
1. Lucene in Action
2. Lucene for Dummies
Adding Lucene documentation to the project

Go to Project properties->Libraries
Select lucene-core-3.5.0.jar
Select javadoc location
Locate lucene-core-3.5.0.jar
Adding javadoc
Setting the working environment

To be able to look at the Lucene internals:

Go to Project properties->Libraries
Select lucene-core-3.5.0.jar
Select source attachment
Locate src folder
Adding sources to the JAR
Now we can examine the sources and documentation.
Basic concepts in Lucene Search Engine

- Indexing
- Documents
- Fields
- Searching
- Queries
Indexing

- Instead of searching the text directly it searches the index
- Uses inverted index - inverts a document-centric data structure (document->words) to a keyword-centric data structure (word->documents)
Documents

- In Lucene, a Document is the unit of search and index.
- An index consists of one or more Documents.
- **Indexing** – adding Documents to an IndexWriter
- **Searching** - retrieving Documents from an index via an IndexSearcher.
Fields

- A Document consists of one or more Fields.
- A Field is simply a name-value pair.
- For example, a Field commonly found in applications is *title*.
- Indexing in Lucene thus involves creating Documents of one or more Fields, and adding these Documents to an IndexWriter.
Adding documents to the index

```java
Directory index = new RAMDirectory();

IndexWriterConfig config = new IndexWriterConfig(Version.LUCENE_35, analyzer);

IndexWriter w = new IndexWriter(index, config);
addDoc(w, "Lucene in Action");
addDoc(w, "Lucene for Dummies");
addDoc(w, "Managing Gigabytes");
addDoc(w, "The Art of Computer Science");
w.close();
```

```java
private static void addDoc(IndexWriter w, String value) throws IOException {
    Document doc = new Document();
    doc.add(new Field("title", value, Field.Store.YES, Field.Index.ANALYZED));
w.addDocument(doc);
}
```
Queries

Lucene has its own mini-language for performing searches.

Allows the user to specify which field(s) to search on, which fields to give more weight to (boosting), the ability to perform boolean queries (AND, OR, NOT) and other functionality.
We read the query from stdin, parse it and build a lucene Query out of it.

```java
String querystr = args.length > 0 ? args[0] : "lucene";

// the "title" arg specifies the default field to use
// when no field is explicitly specified in the query.
Query q = new QueryParser(Version.LUCENE_35, "title", analyzer).parse(querystr);
```
Searching

Searching requires an index to have already been built.

Very simple process:

- Create a **Query** (usually via a QueryParser)
- Handle this Query to an **IndexSearcher**
- Process a list of results
Searching

- Using the Query we create a Searcher to search the index.
- Then instantiate a TopScoreDocCollector to collect the top 10 scoring hits.

```java
int hitsPerPage = 10;
IndexSearcher searcher = new IndexSearcher(index, true);
TopScoreDocCollector collector = TopScoreDocCollector.create(hitsPerPage, true);
searcher.search(q, collector);
ScoreDoc[] hits = collector.topDocs().scoreDocs;
```
Display of results

Now that we have results from our search, we display the results to the user.

```java
System.out.println("Found " + hits.length + " hits.");
for(int i=0;i<hits.length;++i) {
   int docId = hits[i].doc;
   Document d = searcher.doc(docId);
   System.out.println((i + 1) + ". " + d.get("title"));
}
```
Let’s get more practical

Build a Search Engine for answer passage retrieval in the Question Answering system

Use community QA site: Answerbag*

Use ~180k of automatically scraped question/answer pairs from over 20 categories

To reduce the amount of junk content focus only on professionally answered questions

http://www.answerbag.com/
QA system with AnswerBag data
Build high-quality QA corpus
Antique drinking fountain bowls, if they are ceramic, can be restored to their original smoothness and luster. Almost all vintage drinking fountain bowls are ceramic. There are companies that specialize in reglazing the ceramic, bringing it back to a near-new condition.

References:
Dea Bath: Antique Drinking Fountains
Setting up QA example

Download the code from:

http://disi.unitn.it/~severyn/NLPIR.2012/lab01/qa.tar.gz

E.g. try the following in your terminal

$ wget http://disi.unitn.it/~severyn/NLPIR.2012/lab01/qa.tar.gz
$ tar xvfz qa.tar.gz
answers.txt
evalSearchEngine.py
QAIndex.java
QASearch.java
questions.5k.txt
Example of the QA pair: 2503031

Q: What software was used in making the special effects for "Pirates of the Caribbean?"

A: The software used in making the effects for the "Pirates of the Caribbean" films was the Electric Image Animation Software. Made by the El Technology Group, the software runs on both Macintosh and Windows operating systems.
Let’s create an index of our collection

Compile:

$ javac -cp .:lucene-core-3.5.0.jar QAIndex.java

Index QA collection:

$ java -cp .:lucene-core-3.5.0.jar QAIndex index answers.txt

OR:

$ export CLASSPATH=.:lucene-core-3.5.0.jar
$ javac QAIndex.java
$ java QAIndex index answers.txt
Now we can perform search

Compile:
$ javac -cp ::lucene-core-3.5.0.jar QASearch.java

Search:
$ java -cp ::lucene-core-3.5.0.jar QASearch index questions.5k.txt 15 > results.5k.txt

OR:
$ export CLASSPATH=::lucene-core-3.5.0.jar
$ javac QASearch.java
$ java QASearch index questions.5k.txt 15 > results.5k.txt
Evaluate the results

$ python evalSearchEngine.py results.5k.txt

MRR^: 66.43

#   REC-1   ACC
01: 57.30   57.30
02: 67.94   33.97
03: 73.12   24.37
04: 76.00   19.00
05: 78.22   15.64
06: 79.72   13.29
07: 80.70   11.53
08: 81.88   10.23
09: 82.64    9.18
10: 83.58    8.36
The baseline is very naive

You are encouraged to try at least some of this ideas to improve the SE results:

- No stemming or stop words removal
- Lucene is using a simple weighting model to score documents (Cosine similarity)
- Next time we’re going to build a re-ranker for our QA system to improve the SE results
Homework

Play around with indexing and searching
Try out state of the art weighting models, e.g. Okapi BM25 and BM25F.
Add your own evaluation metrics