Lab 7 – Wikipedia Search

CC5212-1

April 29, 2015

Today we're going to build a search engine for Wikipedia.

First we're going to index the keywords in the title and abstract of all the articles in Spanish-Language Wikipedia using Apache Lucene (an open source package for doing inverted indexing of text in documents). Then we'll implement the search function: you type in a keyword, you get back the articles that match.

- Download the code project from `http://aidanhogan.com/teaching/cc5212-1/code/mdp-lab7.zip` and open it in Eclipse.
- Start downloading the data from `http://aidanhogan.com/teaching/cc5212-1/data/lab7/`. You can download both files. `es-wiki-abstracts.tsv.gz` contains 964,843 lines, with an article encoded in each line (*there is no need to unzip this file; we can read a GZIP input stream from the Java program later*). If you open up the second file, `es-wiki-abstracts-1k.tsv`, it contains the first 1,000 lines so you can see the format. Each line has the article URL, the title, and the abstract (delimited by tab).
- While that's downloading, open up the code project again. The first Java class we'll work with will be `IndexTitleAndAbstract`, which will index the large data file into the Lucene inverted index. The second class we'll work with will be `SearchIndex`, which will take your keyword searches and use the index to find relevant Wikipedia articles.[1] To help you in this, there were some Lucene code examples in the slides from Lecture 7. The examples are (by design :P) slightly different so you cannot copy them symbol for symbol, but the main steps are exemplified there.
- So first open up `IndexTitleAndAbstract`. The main method is already written for you. From the command line, it takes an input file (which will be `es-wiki-abstracts.tsv.gz`) and an output directory (an empty existing directory) to write the inverted index to. You can jump to the `indexTitleAndAbstract(.,.)` method.
  - Open the directory at `indexDir`, set the analyser for `Version.LUCENE_48`, and open a new `IndexWriter` configured for writing.
  - For each line in the input, if the line is not empty (or just whitespace), split the line by tab. The first split contains the article URL, the second the title, the third the abstract. If the array does not contain at least a URL and a title, skip it.
    - Create a new Lucene document for every valid line.
    - Every `TICKS` lines, print progress to standard error.
    - For the following field names, use the `FieldNames` enum, for example, `FieldNames.URL.name()`.
    - Index the URL as a string-field with store set to yes.
    - Index the title as a text-field with store set to yes.
    - If an abstract is present and non-empty (some articles don't have one), index the abstract as a text-field with store set to yes.

[1] You can ignore the `ExtractTitleAndAbstract` class (which prepares the data) and the `Main` class (which would be useful if you want to build a jar, for example); I left these in there in case you want to re-use the code later for a class project or something.

    - Index the time the document was indexed (current time) as a long field with store set to no.
    - Add the document to the writer.
  - Don't forget to close the writer at the end.
  - Once the class is finished,run it with `-i [input-file] -igz -o [any-output-dir]`. You can run it over the big file (the `-igz` says that the input is GZipped).
  - Assuming all runs well, congrats, you've made a search engine index! But now to see that it works.

- Open up the `SearchIndex` class.
  - Again the `main` method is already done. Jump to `startSearchApp(.)`. This will take the directory of the index you just wrote and accepts keyword searches from the command line over that index.
    - First open an index reader for the directory name passed as input.
    - Open a searcher over the index and create the same analyser as before.
    - We want to search over both title AND abstract so we want to create a multi-field query parser. We also want title matches to score more highly: you can pass the `BOOSTS` object at the top of the class for this.
    - Next we need to read lines from the input. There's some code left there for you. Inside the `try` you need to parse the query, get the hits from the searcher (get `DOCS_PER_PAGE` number of hits), and for each results, print the details of the document (the url, title, etc.; you can print anything that had store set to true during indexing).
    - Once you're finished, time to test it! Run the class with the argument `-i [index-dir]`. Try a search like "`Obama`". Do the results make sense? Play around with some other searches.
    - OPTIONAL: Try vary the boost factors between title and abstract and see how it affects the results of a few queries. How do the results change?

- Submit your two classes to u-cursos by next Monday.