CS 419/519 Document Analysis
Lab and Assignment 1: Information Retrieval

September 28, 2015

This lab and assignment involves creating a simple search engine, evaluating it to understand its performance, and making changes to improve performance where you can.

- Maximum marks: 10
- Programing language: Java (only)
- Assignment questions: CS419/519 Canvas Site
- Deadline: Friday Oct 9 @ 2pm (lab attendance mandatory for discussing answers)

Marking scheme and requirements: Full marks will be given for (1) working, readable, reasonably efficient, documented code that achieves the assignment goals and (2) for providing appropriate answers to Q1-Q6 in a file `ANSWERS_lastname.pdf` submitted with your solution.

Please adhere to the collaboration policy on the course website – people you discussed the assignment solution with, or websites with source code you used should be listed in `ANSWERS_lastname.pdf`.

To verify that your code runs end-to-end, you may be asked to make some changes to your search engine during the evaluation lab and then re-evaluate results.

What/how to submit your work: Please submit `ANSWERS_lastname.pdf` and all your source code, zipped into a single file called `Assignment1_lastname.zip` and submit to **Canvas**. *Please do not submit data or other large binary files — Canvas will not accept large submissions.*

Credit: This assignment derives from one originally prepared by Paul Thomas (CSIRO, Australia) for the Australian National University version of this course.

1 Before the Introductory lab

In the lab you'll familiarize yourself with two pieces of software: Lucene, a Java library for building search engines, and trec_eval, the standard software for evaluating search engines with test collections. *It's important that you pre-build all of this software before the lab (and install all requisite software to do this)* so that instructors can answer questions relevant to the actual lab during the lab time.

All code should be provided in the github repository that contains this assignment handout.

The lucene-demo directory includes a README file with instructions for getting started. It is suggested to use an IDE such as Eclipse, IntelliJ, etc. Make sure you can run the classes listed in the README and obtain error-free output.

The trec_eval.8.1 bundle also contains a README. This software is distributed as C source. To build an executable, you'll need "gcc" and "make" installed. Use "make quicktest" to build the code and run some quick tests to be sure it's working properly. Once you've made an executable, typing *./trec_eval* will run the program and *./trec_eval −h* will list all the options available ("h" is for "help"). It's OK if someone else provides you with a working binary for this part – you don't need to modify it, you just have to run it on your particular system.

2 In the Introductory lab

By the end of the Introductory lab, you should be able to show Lucene talking to *trec_eval*, either with the "AIR" topics of this part or with the "government" topics of the next part.

Find testing data from the "lab1-q1-test-collection" folder. This is a very small data set which contains three things:

- A set of documents (email messages), in the documents directory.
- A set of queries – also called "topics" – in the topics directory.
- A set of judgements, saying which documents are relevant for each topic, in the qrels directory.

The overall goal here is to search the documents for each query, and produce some output. The output can then be compared with the judgements to say how good (or bad) Lucene is for these tasks. So:

1. Lucene needs to make an index of the set of documents, then
2. Lucene needs to read queries (topics) from the set given, and
3. Lucene needs to produce output for each query (topic). Then,
4. trec_eval can compare Lucene's output with the judgements and say how good (or bad) the output is.

Once you understand how Lucene works from running and investigating the classes in lucene-demo, you will need to get Lucene talking to *trec_eval*. This will involve a few modifications to the provided code:

1. You'll need to index the correct document set.
2. Ideally, you'll need to read topics from a file (topics/air.topics) instead of having them in the program directly. (For this lab, you can hard-code them to save time, but you should not do this for the final assignment submission – this will be checked.)

3. Produce output in the format trec_eval expects. This is:

   ```
   01 Q0 email09 0 1.23 myname
   01 Q0 email06 1 1.08 myname
   ```

   where "01" is the topic number (01 to 06); "Q0" is the literal string Q0, that is exactly those two characters[1]; "email*n*" is the name of the file you're returning; "0" (or "1" or some other number) is the rank of this result;"1.23" (or "1.08" or some other number) is the score of this result; and "myname" is some name for your software (it doesn't matter what, just be consistent).

Once you've done this, index and rank the documents for any or all of the six test queries (e.g., using a default Lucene index and ranking) and run *trec_eval* with the qrels file provided in *air.qrels*.

If trec_eval runs correctly and produces numbers which you think are sensible, you're done with this part. You might want to look at the output, though, and get some understanding of what it means; later you will be asked to interpret this and to choose evaluation measures you prefer.

3 Main Assignment (Assessed in Evaluation Lab)

In the Introductory lab section, you were asked to get Lucene to index the documents from a very small test collection, run a few queries ("topics"), and produce output in the format expected by *trec_eval*. You were also asked to run *trec_eval*, to compare your output to the human relevance judgements ("qrels"), and to check you understand the output.

In this part of the assignment, you will run tests with a bigger collection of documents, and more queries. You will also need to improve on the baseline Lucene configuration.

The data you need should be available in the government directory of the assignment repository. The test collection is about 34,000 documents from US Government web sites; and the topics are 31 needs for government information. Both were part of the TREC conference in 2003.

Q1. Appropriate TREC measures

trec_eval can report dozens of measures: for example "p5" (precision, in the first five documents returned), "num_rel_ret" (the number of relevant documents retrieved over all queries), and "recip_rank" (the reciprocal rank of top relevant document: e.g., 0.25 if the first relevant document is the fourth in the ranking). You can get a list by running *trec_eval* $-h$ or in trec_eval's README file.

(a) Which of trec_eval's measures might be appropriate for measuring search system performance for government web sites? [List the measure]

(b) Why do you think this measure is appropriate? [1 sentence]

Q2. Indexing and querying

Index the government documents, run the queries ("topics") through (vanilla) Lucene as a baseline system, and run trec_eval to compare Lucene's results with human judgements.

(a) How well did the baseline Lucene system do on your chosen measure? [Provide the number.]

[1]Once upon a time, this field meant something to *trec_eval*. It is not used for anything now but it's still required.

(b) Are there any particular topics where it did very well, or very badly? [If so, list a few topic IDs for each]

(Note: *trec_eval* $-q$ will report measures for each query/topic separately as well as the averages. This will help you pinpoint good or bad cases.)

Q3. Improving performance

Look at where the baseline Lucene system did well, or badly.

(a) What do you think would improve Lucene's performance on this test collection, and why? [1-3 sentences]

(Note: Try to justify your answers by understanding why Lucene failed to rank the relevant documents highly on poor-performing queries. You might consider: stemming terms; modifying the stopword list (if used); changing the weights of terms; doing relevance feedback; fielded indexing and retrieval; proximity-based retrieval; tuning the ranking formula somehow; or something else.)

Q4. Modifications to Lucene

Based on your analysis, make any changes you think can improve your baseline.

(a) What modifications did you make and what were the improvements? [1-3 sentences, any single improvement over the baseline is sufficient for full credit, but nonetheless, you are encouraged to explore]

Some starting points:

- The constructor for FileIndexBuilder creates an Analyzer, which is the Lucene class responsible for doing stemming, stopword removal, etc. If you change this in the indexer, make sure you use the same analyzer when you process queries! (Otherwise for example you might put "duck"–the stemmed word–in the index, then look for "ducks"–the unstemmed word.)
- DocAdder builds fields in the index for "content" and "first line". If you want to include other fields, maybe so you can weight them differently, this is a good place to start.
- See the comments in SimpleSearchRanker for some of the things you can do with Lucene queries: for example, wild cards, changing weights, ...
- If you want to redo scoring entirely, you need to write a new version of the Similarity class.

Q5. Rerunning the modified Lucene

Run your modified version of Lucene, and look again at the evaluation measure you chose.

(a) Did your changes improve things overall? [yes/no]

(b) Did some queries get better while others got worse? [yes/no]

(c) What do you think this means for your idea: was it good? Why or why not? [1-3 sentences]

Q6. Documentation

(a) List the files you modified that contain your solution. [filename, brief descriptive phrase]

(b) Provide the list of commands to reproduce your best overall result reported above.