Document analysis
Lab and assignment 1: Information retrieval

shaodi.you@anu.edu.au

July 2016

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:	8
Programming language:	Java (only)
Assignment questions:	Post to the Wattle discussion forum
Deadline:	Q1: Lab1
	Q2–Q6: Wednesday 10 August, 23:59

Marking scheme and requirements For question 1, full marks will be given for working, readable, reasonably efficient, documented code that succeeds on all test cases.

For questions 2 to 6, full marks will be given for an answer that provides a *well-reasoned* and succinct response to the question that addresses all requested points. There may be more than one answer for each question that achieves full marks.

Academic misconduct policy All submitted written work and code must be your own (except for any provided Java starter code, of course)—submitting work other than your own will lead to both a failure on the assignment and a referral of the case to the ANU academic misconduct review procedures.

How to submit your work Answers to questions 2 to 6 should be in a PDF file. MS Word or other document formats are not accepted.

Please submit your writeup and *all your source code*, zipped into a single file called `assignment1_yourname.zip`, with a `README.txt` stating what each file is for and how to run your code. You don't need to submit code that you haven't changed.

Submission is via the course Wattle page.

1 Getting started: lab part

This question will be evaluated in 'Lab1'. You must show lucene talking to trec_eval, either with the "AIR" topics of this part or with the "government" topics of the next part.

In the lab you'll familiarise yourself with two pieces of software: *lucene*, a Java library for building simple search engines, and *trec_eval*, the standard software for evaluating search engines with test collections.

You can download all the software you need from the Wattle page.

The lucene bundle includes a README file with instructions for getting started; you will need to unzip it, then import this as a new project in Eclipse. Some help with Eclipse will be available in the lab.

The trec_eval bundle also contains a README. This software is distributed as C source. To build an executable, unzip the bundle then run `make` in the directory. `make quicktest` will run some quick tests to be sure it's working properly[1]. 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 "**h**elp").

Q1 [1pt]. Basic setup

First, build the lucene demo and trec_eval.

Next, download the "lab1-q1" test collection from Wattle. 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.

[1] Sometimes this test shows an error, due to floating point rounding: if the results are different by a very small amount, this is probably okay. You're welcome to check with the tutors.

Your job for this part You will need to get lucene talking to trec_eval. There is some good sample code in the bundle, but this will involve at least two modifications:

1. Read topics from a file (`topics/air.topics`) instead of having them in the program directly. (For the part marked in the lab, you can also read them from the keyboard or hard-code them; but for the written part you'll need to read them from a file as there are just too many.)

2. Produce output file (*e.g.*, `retrieved.txt`) 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[2]; "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).

Once you've done this, index and rank the documents for any or all of the six test queries and run trec_eval which compares the qrels file provided in `air.qrels` with your result `retrieved.txt`.

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.

2 Written part

Answers to these questions, numbers 2 to 6, should be submitted via Wattle as described above. Please ensure you address all the questions.

In the 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.

The data you need is available from Wattle. 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.

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

Q2 [0.5 pt]. 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.

▷ Which of trec_eval's measures might be appropriate for measuring a search system for government web sites? ▷ Why do you think this measure is appropriate?

(Note, there might be more than one good answer and you don't need to find more than one. We discussed different measures in lecture 3. You can also read the textbook: Section 8.3 and optionally 8.4.)

Q3 [0.5 pt]. Indexing and querying

Index the government documents, run the queries ("topics") through lucene, and run trec_eval to compare lucene's results with human judgements.

▷ How did lucene do on your chosen measure? ▷ Are there any particular topics where it did very well, or very badly?

(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.)

Q4 [2 pt]. Improving performance

Look at where it did well, or badly. ▷ What do you think would improve lucene's performance on this test collection, and why?

(Note, you might want to think about: stemming terms; stopwords; changing the weights of terms; doing relevance feedback; tuning the ranking formula somehow; or something else. We've discussed lots of options in the lectures, and you're free to try your own ideas as well.)

Q5 [2 pt]. Modifications to Lucene

Make any changes you think will help, based on your analysis. ▷ Please submit one to two pages (maximum) describing what you changed, and illustrating it with sections of code.

Some starting points:

- The constructor for `FileIndexBuilder` creates an `Analyzer`, which is the lucene class responsible for doing things like stemming and removing stopwords. 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 re-do the scoring entirely, you will need to write a new version of the `Similarity` class.

2.1 Q6 [2 pts]. Rerunning the modified Lucene

Run your modified version of lucene, and look again at the evaluation measure you chose. ▷ Did your changes improve things overall? ▷ Did certain queries/topics get better or worse? ▷ What do you think this means for your idea: was it good? ▷ Why or why not?

> **Note, you will *not* be marked down if you had a good idea, and you can explain it, but it just didn't work. You *will* be marked down if your idea works but you don't say why!**