Document Analysis
Lab and Assignment 1: Information Retrieval

dongwoo.kim@anu.edu.au

July 2017

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:	Python (only)
Assignment questions:	Post to the Wattle discussion forum
Deadline:	Q1: Week3's Lab, depends on your enrolled lab.
	Q2–Q6: Friday 11 August, 23:59

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

For question 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 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. Do not incorporate the dataset files into the submission.

You can submit the zipped file via course Wattle page.

1 Getting Started: Lab Assignment

Question 1 will be evaluated in 'Lab1'. You must show *Elasticsearch* 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 will familiarise yourself with two pieces of software: *Elasticsearch*, a commercial search engines, and `trec_eval`, the standard software for evaluating search engines with test collections. To control the functionality of *Elasticsearch*, you will learn how to use *Elasticsearch* Python library.

We will provide two bundle softwares for this assignment. You can download the bundle softwares and dataset from the Wattle page except *Elasticsearch* . Please download *Elasticsearch* at its homepage[1], and install the software on your machine. *Elasticsearch* is built using Java and requires at least Java 8 in order to run. We recommend installing Java version 1.8.0_131 or later.

Two provided bundle softwares include:

- `elastic.zip`: contains *Elasticsearch* demo code. README file gives instructions for getting started; you will need to unzip it, and then open files from your favourite text editor or IDE. Note that the demo code is compatible with python 2.7. If you are using python 3, we recommend you to use a python virtual environment. To run the demo code, you must install python Elasticsearch library[2] which provide common ground for all *Elasticsearch*-related code in Python.
- `trec_eval.zip`: contains a software for evaluating performance of a search engine. 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[3]. Once you have made an executable, typing `./trec_eval` will run the program and `./trec_eval -h` will list all the options available ("h" is for "help").

To build a working environment for the lab session, we already prepared requirement files on Wattle. First, download `lab_env.zip` from Wattle and unzip. It contains two install files: `Miniconda2-latest-Linux-x86_64.sh`[a] and `elasticsearch-5.5.1.zip`.

1. Go to the directory where you unzipped the file.
2. Run

   ```
   bash Miniconda2-latest-Linux-x86_64.sh
   ```

   This will install miniconda on your lab computer, which includes python2.7. Make sure you update the bash environment. If you added the miniconda path to `.bashrc`, then type `source .bashrc`.

[1] https://www.elastic.co/downloads/elasticsearch

[2] The library can be installed via `pip install elasticsearch` command. Location of library API document is: `https://elasticsearch-py.readthedocs.io/en/master/`

[3] 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.

3. To verify miniconda is installed properly, run following command:
   `python --version`
   you should see the python version from Continuum Analytics, e.g:
   `Python 2.7.13 :: Continuum Analytics, Inc.`

4. Unzip `elasticsearch-5.5.1.zip`. Run

   `./elasticsearch-5.5.1/bin/elasticsearch -d`

   It will run *Elasticsearch* on your background. To check the *Elasticsearch* running correctly, open your browser and goto `http://localhost:9200`. If you get the response then the process is running properly. Note that starting up *Elasticsearch* may take a few seconds.

5. To run the demo code, install the python *Elasticsearch* client via `pip`. Run

   `pip install elasticsearch`

6. To check the client installed correctly, run python and import the library.

   `python -c "import elasticsearch"`

   If you haven't got any error message, you install it correctly.

7. Now you are ready to run the demo code!

[a] https://conda.io/miniconda.html

Q1 [1pt]. Basic setup

First, unzip both files and build `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 (11 email messages), in the `documents` directory.
- A set of queries, also called 'topics', in `topics/air.topics` file. The format of `*.topic` file is "`query_id query_terms`". For example, the first line of '`air.topics`' file is

  ```
  01 ducks
  ```

  which means that the ID of query is `01` and the corresponding query is `ducks`.
- A set of judgements, saying which documents are relevant for each query, in the `qrels/air.qrels` file. The format of `*.qrels` file is "`query_id 0 document_name binary_relevance`". For example, the first line of '`air.qrels`' is

  ```
  01 0 email01 0
  ```

  which means that the document 'email01' is not relevant to the given query_id 01. The binary relevance is 1 if the file is relevant to the query, otherwise 0. Please ignore the second argument 0 as it is always 0.

The goal of the Lab assignment is to build a simple search engine which takes queries from user and generates outputs. The output can then be compared with the judgements to say how good or bad the search engine is for these tasks. To make the search engine work:

1. The search engine needs to build an index from the set of documents, then
2. The search engine needs to read queries (topics) from 'air.topics' file, and
3. The search engine needs to produce output for each query (topic). Then,
4. `trec_eval` compares the engine's output with the judgements and measures the performance of the engine.

Your job for this part You will need to get *Elasticsearch* 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 (`lab1-q1/topics/air.topics`) instead of having them in the program directly, unlike the sample search file which has hard-coded queries in it.
2. Search documents indexed by *Elasticsearch* . You may choose one of search algorithms used in the sample search file. Produce result file (e.g., `retrieved.txt`) according to `trec_eval` standard output format:

   ```
   01 Q0 email09 0 1.23 my_IR_system1
   01 Q0 email06 1 1.08 my_IR_system1
   ```

   where '01' is the query ID (01 to 06); ignore 'Q0'[4]; 'emailxx' is the name of the file; '0' (or '1' or some other integer number) is the rank of this result; '1.23' (or '1.08' or some other number) is the score of this result; and 'my_IR_system1' is the name for your retrieval system. In particular, note that the rank field will be ignored in `trec_eval`; internally ranks are assigned by sorting by the score field with ties broken deterministicly (using file name).
3. Once you have 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 are 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.

[4]This field was used before, but not anymore. `trec_eval` still keeps this field for the backward compatibility.

2 Written Assignment

Answers to Question 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 *Elasticsearch* 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 (gov-test-collection). The test collection is about 34,000 documents from US Government web sites; and the topics need for government information. Both were part of the TREC conference in 2003.

Q2 [0.5 pt] Appropriate TREC measures

`trec_eval` can report dozens of measures: for example 'p5' (precision@5), '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 *Elasticsearch*, and run `trec_eval` to compare *Elasticsearch*'s results with human judgements.

▷ How did *Elasticsearch* 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 *Elasticsearch*'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 some options in the lectures, and you're free to try your own ideas as well.

Q5 [2 pt] Modifications to *Elasticsearch*

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 basic configuration of *Elasticsearch* is in `es_settings.json` and `es_settings_sample.json` files where you can configure index time analyser and search time analyser. For example, you can specify tokeniser, stemmer, and lemmatiser here. For more details about the configurations, refer https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis.html.
- `es_helper.py` contains some useful functions for improving your search engine. For example, `get_tokens` function takes query string and returns tuples of (filename, term vectors). `example_term_vector.json` file contains an example of a term vector of a document. If you want to check how your tokeniser works then use `analyze_query` function to see the tokenised query.
- Read the demo and README files carefully. These files contain some useful links and explains how the index and search work through *Elasticsearch* APIs.

Q6 [2 pts] Rerunning the modified *Elasticsearch*

Run your modified version of *Elasticsearch*, 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!