# CS6200: Information Retrieval

## Project 2

Return to basic course information.

**Assigned:** Thursday, 24 October 2013
**Due:** Email TAs with subject "CS6200 Project 2" by Thursday, 14 November 2013, 6pm

---

Lemur details

Sample:q57_d3_laplace

---

This project requires that you use a server to build five variations on a simple search engine. For each variation you will run and evaluate on the same set of 25 queries.

The description of the project is long, but that's because we've tried to describe carefully what you have to do.

### Building an IR system

This project requires that you construct a search engine that can run queries on a corpus and return a ranked list of documents. Rather than having you build a system from the ground up, we are providing a Web interface to an existing search engine index. That interface provides you with access to corpus statistics, document information, inverted lists, and so on. You will access it via the web, so you can implement your project in any language that provides you with Web access, and you can run it on any computer that has access to the Web. Computers in the CCIS department will work, but you are more than welcome to use your own computer.

One of the nice things about the CGI interface is that you can get an idea of what it does using any browser. For example, the following CGI command:

> http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?v=encryption

will provide you a list of the documents that contain the word "encryption". Go ahead and click on the link to see the output (in a separate window). It lists the number of times the term occurs in the collection (ctf), the total number of documents that contain it (df) and then, for each of the documents containing the term, it lists the documents id, its length (in terms), and the number of times the term occurs in that document.

The format is easy for a human to read and should be easy for you to parse with a program. However, to make it even simpler, the interface provides a "program" version of its interface that strips out all of the easy-to-read stuff and gives you a list of numbers that you have to decipher on your own (mostly it strips the column headings). Here is a variation of the link above that does that. Click on it to see the difference.

> http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?g=p&v=encryption

From class, it should be pretty straightforward to imagine how to create a system that reads in a query, gets the inverted lists for all words in the query, and then comes up with a ranked list. You may need other information, and the interface makes everything accessible (we hope). There is a help command, as well:

http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?h=

Play around with the engine to ensure it makes sense.

Three very important tips:

1. First, fiji4.ccs.neu.edu is not accessible via name from all CCIS internal networks; you may need to use the (local) IP address 10.0.0.176 instead. From outside CCIS, the fiji4.ccs.neu.edu name should always work.
2. Second, in order to get efficient processing out of this server, you should string as many commands together as you can. For example, if you want to get term stemming, stopping, and statistics information for three terms, ask for them all at once as in:

   http://fiji4.ccs.neu.edu/~zerg/lemurcgi/lemur.cgi?c=encryption&c=star&c=retrieval

   (If you use the program mode version of the same command, the Lemur images are dropped.) This is because there is some overhead getting the process started, and putting commands together amortizes that over several requests. It *works* to do them separately, but it might take longer.
3. And third, some things will be inefficient this way, no matter what. You may find it useful to use the file listing stem classes and the file that maps between internal and external docids rather than using calls to the server.

## The Databases

We have set up a database for you to use for this assignment. They are taken from the AP89 subset of the TREC corpora and consist of 84,678 news stories from the Associated Press in 1989 (here is more information about the corpora if you're curious). In fact, we've set up four versions of the database: all four combinations of using or not using each of stemming and stopping. The collections contain the same documents, but the inverted lists will obviously be different. The "d=?" command will provide you with useful information about the databases.

The stop list that we used (for the two versions with stopping) is the Inquery stop list and includes 418 words. You can download a list of those words here, since you will probably find it useful for your own query processing. Note that the "c=term" command (and "t=term") also shows the stopword status of a term (though be careful that you only pay attention ot the stopword status of a word on a database that uses stopping).

When stemming was applied, we used the KStem stemming algorithm. If you are running on an unstemmed collection, use the "c=term" command to get term statistics about the term. It will return stemming information, but you should ignore it. If you are running on a stemmed collection, the "t=term" command is also useful. Both "c=" and "t=" tell you the stemming and stopping status of the word; however, the former gives you term statistics for the *original* word rather than the stem and the latter gives you term statistics for the *stem* rather than the original word. Having both commands available can be useful because of the way Lemur does stemming: it does not stem capitalized words that are not at the start of a sentence. As a result, stemmed collections will have *american* converted to *america* but *American* will not be changed. Whether

this behavior is "correct" or not is a matter of opinion; you just have to cope with it. (By and large you will be safe assuming that stemming always happens.) Here is a file that lists all stem classes appearing in this database (the format is "root | variant1 variant2 ..."). It actually includes more words than are in AP89, but everything in AP89 definitely appears.

## The systems you build

Here is a set of 25 queries taken from TREC in the format "querynum. query". These queries have been judged by NIST against AP89 (using the pooling method). To help you with your analysis below, the queries are sorted in order of the number of relevant documents in the collection: the first query has 393 relevant documents and the last has only one (the complete distribution is also available). Your assignment is to run those queries against the provided collection, return ranked lists in a particular format, and then submit the ranked lists for evaluation.

Implement the following variations of a retrieval system:

1. Vector space model, terms weighted by Okapi tf only (also referred to as "Robertson's" tf; see note), inner product similarity between vectors. Note that you currently do not have sufficient information to calculate the document length vector in the denominator easily, so pure cosine similarity isn't used.

   For Robertson's tf (also referred to as "Okapi tf") on documents use `OKTF=tf/(tf + 0.5 + 1.5*doclen/avgdoclen)`.

   On queries, Okapi tf can be computed in the same way, but use length of the query instead of `doclen`.

2. Vector space model, terms weighted by Okapi tf (see note) times an IDF value, inner product similarity between vectors. The database information command provides collection-wide statistics that you need.

   You will have to use for the weights `OKTF * IDF` where `OKTF` is as shown above.

3. Language modeling, maximum likelihood estimates with Laplace smoothing only, query likelihood. The database information command provides collection-wide statistics that you need.

   If you use multinomial model, for every document, only the probabilities associated with terms in the query must be estimated because the others are missing from the query-likelihood formula (consult slides on retrieval models).

   For model estimation use maximum-likelihood and Laplace smoothing. Use formula (for term i)

   $$p_i = ( c_i + 1 ) / (n + k)$$

   where $c_i$=freq count (tf) of i, n=number of terms in document (doc length) , k=number of unique terms in corpus.

4. Language modeling, Jelinek-Mercer smoothing using the corpus, $\lambda = 0.2$ of the weight attached to the document probability, query likelihood. The database information command provides collection-wide statistics that you need.

   The formula for Jelinek-Mercer smoothing is

   $$p_i = \lambda P + (1 - \lambda) Q$$

   where P is the estimated probability from document (max likelihood = $c_i/n$) and Q is the estimated probability from corpus. For

simplicity, you may use background probability = cf / total number of terms in the corpus).

5. BM25, as described in the lecture on retrieval models in class.

   As a starting point, you may use the parameter settings suggested in the BM25 example given in those slides.

## Testing the models

Run all 25 queries and return the top 1,000 ranked documents for every query, placing all this data in one single file. (If, for a given query, there are fewer than 1,000 documents that contain the query terms, you may return fewer than 1,000 documents.) Thus, your output file will have at most 25*1,000 = 25,000 lines in it. To work with the evaluation program, the lines in the file must have the following format:

*query-number* Q0 *document-id rank score* Exp

where *query-number* is the number of the query (the number between 51 to 100, for this project), *document-id* is the external ID for the retrieved document (here is a map between internal and external IDs , *rank* is the position in the ranked list of this document (1 is the best; 1000 is the worst; do not include ties), and *score* is the score that you system creates for that document against that query. Scores should descend while rank increases. "Q0" (Q zero) and "Exp" are constants that are used by some evaluation software and must be present, but won't make a difference for your score. The overall file should be sorted by ascending *rank* (so descending *score*) within ascending *query-number* (remember that the queries are not in sorted order in the file).

There are four databases available, each corresponding to various combinations of stemming and/or stop wording. We suggest that you intially work with database d=3, corrsponding to an index created with stemming and stop wording. You may work with the other databases for additional credit; see below.

Run all five models against this database, creating five output files, each with approximately 25,000 lines of output (1,000 documents retrieved for each of 25 queries). In TREC parlance, each output file is referred to as a 'run'.

To evaluate a single run, you first need to download `trec_eval` and a qrel file which contains relevance judgments. You will use two different qrel files:

1. the NIST qrel_file, as used in the TREC conference, and
2. the IS4200/CS6200 qrel file, as generated by former IS4200/CS6200 students.

To perform an evaluation, run

```
trec_eval [-q] qrel_file results_file
```

The `-q` option shows a summary average evaluation across all queries, followed by individual evaluation results for each query; without the `-q` option, you will see only the summary average. The trec_eval program provides a wealth of statistics about how well the uploaded file did for those queries, including average precision, precision at various recall cut-offs, and so on. You will need some of those statistics for this project's report, and you may find others useful. You should evaluate all five runs using both qrel files, for a total of 10 evaluations.

Using a simple tf-idf retrieval scheme, our baseline system achieves a mean average precision of approximately 0.2 using the TREC qrel file. If your system is dramatically under-performing that, you probably have a bug.

## What to hand in

Provide a short description of what you did and some analysis of what you learned. This writeup should include at least the following information:

- Uninterpolated mean average precision numbers for all 10 evaluations (five runs * two qrel files).
- Precision at 10 and precision at 30 documents retrieved for all 10 evaluations.
- A comparative analysis of the evaluations obtained with the TREC qrel file and the IR class generated qrel file.

Feel free to try other runs to better explore the issues, e.g., How does Okapi tf compare to raw tf or other tf formultions?; What about other kinds of smoothing (or smoothing parameters) for language modeling or other language modeling formulations besides query likelihood?; and so on. Also, you may wish try the other databases to see what effect (lack of) stemming and/or stop wording has on your results.

Email an electronic copy of your report to the TAs. The report should not be much longer than a dozen pages (if that long). In particular, do not include `trec_eval` output directly; instead, select the interesting information and put it in a nicely formatted table.

## Grading

This project is worth 200 points. You will be graded on correct implementations of the models and on the clarity of your report, as described above.

You may gain up to an extra 50 points for additional analyses such as:

- Extra runs that make interesting changes to the models. Don't just, for example, try 10 values of $\lambda$ for Jelinek-Mercer smoothing. The more the better, though a small number of really interesting runs will be worth more than a large number of minor variants.
- Graphs of the recall/precision numbers are worth up to 15 points since they take some time and present a much nicer illustration of the effectiveness, particularly when multiple plots are combined on one graph for comparison.
- Analysis beyond the minimum required could result in up to 25 points, though that would be an extreme. Good analysis looks at something that was intriguing and investigates it in more detail. It might require doing some additional runs (which would give points for extra runs, too!). Examples include looking at query-by-query performance to explore what makes some queries fail, or playing with some query processing to get rid of stop structure ("documents will") or to otherwise process the queries.