RMIT
School of Computer Science & IT

ISYS 1078 / 1079 — Information Retrieval

Assignment 2: Searching

Due: 5:30pm, Friday 7 October 2011

This assignment is worth 25% of your overall mark.
The two assignments together form a hurdle.
This is an individual assignment.

Introduction

In this assignment, you will implement ranked querying and query expansion, run a set of sample queries, and evaluate system performance. You will need to design and implement appropriate data structures for efficient searching.

In assignment 1, you implemented text processing and indexing. This second assignment focuses on search: the aims of the assignment are to enhance your understanding of retrieval models and evaluation, and to give you practical experience with the implementation of search algorithms.

For this assignment, you should re-use (and modify where required) your indexing code from assignment 1. As usual, wherever code is re-used (including your own) this must be clearly identified in your submission.

The "Information Retrieval" blackboard contains further announcements, a list of frequently asked questions, and a discussion board for this assignment. You are expected to check the discussion board on daily basis. Login through `http://my.rmit.edu.au`.

Plagiarism

All assignments will be checked with plagiarism-detection software; any student found to have plagiarised will be subject to disciplinary action as described in the course guide. Plagiarism includes submitting code that is not your own or submitting text that is not your own. Submitting one comment in your source code, or a sentence from someone else's report, is plagiarism, as is submitting work from previous years. Allowing others to copy your work is also plagiarism. All plagiarism will be penalised; there are no exceptions and no excuses. For further information, please see: `http://www.rmit.edu.au/compsci/current#rules`.

General Requirements

This section contains information about the general requirements that your assignment must meet. *Please read all requirements carefully before you start.*

- You must implement your programs in Java or C. Your programs should be well written, using good coding style and including appropriate use of comments. Your markers will look at your source code, and coding style will form part of the assessment of this assignment.
- Your programs may be developed on any machine, but must compile *and* run on `yallara`.
- You must include a file called "README.txt" with your submission. This file should include at the top your name and student id. It needs to clearly explain how to compile and run your programs on `yallara`.
- Paths should not be hard-coded.
- Parts of this assignment will include a written report, this may be in *plain text* or *PDF* format. If you choose to submit a PDF file, it is your responsibility to test whether your file can be successfully displayed on `yallara` using the `xpdf` viewer.
- Please ensure that your submission follows the file naming rules specified in the tasks below. File names are case sensitive, i.e. if it is specified that the file name is `gryphon`, then that is exactly the file name you should submit; `Gryphon`, `GRYPHON`, `griffin`, and anything else but `gryphon` will be rejected. If you do not follow the file naming rules, you risk yielding zero marks for the corresponding task.
- For some tasks, you may need to generate large output files. If you do not have enough space in your own account, you can create a directory under `/scratch/InformationRetrieval` on `yallara`. The directory that you create should have the same name as your username. Please bear in mind that this directory should *only* be used to store temporary output files from your programs. It is *not* safe to store your source code or other files that you wish to keep here, as the lifespan of these directories is only short-term and they may need to be deleted without prior notice. Also, please clean up your subdirectories every day (i.e. delete them before you log out of `yallara`) to free up disk space.

Programming Tasks

Have a look at the file */public/courses/MultimediaInfoRetrieval/2011/a2/collection* on yallara. It is part of the TREC *ad hoc* retrieval test collection. This collection comprises 35472 English documents from a newspaper.

This data follows the same markup format that you worked with in assignment 1. You will need to write programs to perform searches on this data.

As a preliminary step, you should therefore be able to index the data efficiently, for example using an inverted index. This will enable to you to easily access the various term occurrence statistics that you need to use for the searching tasks described below.

1 Ranked Retrieval (35%)

Write a program called `search` that implements the Okapi BM25 similarity function to rank documents in a text collection. An example of how your program should be invoked is:

```
./search  -BM25 -q <query-label> -n <num-results> <lexicon> <invlists>
<map> <queryterm-1> [<queryterm-2> ...  <queryterm-N>]
```

(or equivalent Java invocation) where

- `-BM25` specifies that the BM25 similarity function is to be used.
- `-q <query-label>` is an integer that identifies the current query. For example, you might indicate that the current query label is "1133". This item is only used for output formatting (see below).
- `-n <num-results>` is an integer number specifying the number of top-ranked documents that should be returned as an answer.
- `<lexicon>` and `<invlists>` are the inverted index files that your `index` program created.
- `<map>` is the mapping table from internal document numbers to actual document identifiers. (See Assignment 1 for details. You are encouraged to re-use your indexing code from Assignment 1, with modifications where required.)
- `<queryterm-1> [<queryterm-2> ...  <queryterm-N>]` are a variable number of query terms (but at least one query term) that the program will receive as command-line arguments. Each of these terms should be looked up in the lexicon, and the appropriate inverted list data fetched from the inverted list file.

Your search program should be efficient, using suitable data structures and algorithms. As a minimum, your program should:

- Process a query, one term at a time.
- Use *accumulators* to accrue partial similarity scores as each term is processed (see Retrieval Models I lecture notes).

- Use a *min-heap* data structure to select the top a accumulators with the highest similarity scores.
- Calculate document weights at *indexing* time, and store these in the mapping table (`map` file) for fast lookup at query time.

The specific *simplified* version of the BM25 similarity function that you should implement is:

$$BM25(Q, D_d) = \sum_{t \in Q} \log \left(\frac{N - f_t + 0.5}{f_t + 0.5} \right) \cdot \frac{(k_1 + 1) f_{d,t}}{K + f_{d,t}} \tag{1}$$

where:

- Q is a query consisting of terms t;
- D_d is a document in the collection;
- N is the number of documents in the collection;
- $f_{d,t}$ is the number of occurrences of t in d;
- f_t is the number of documents containing term t.
- $K = k_1 \cdot \left((1 - b) + \frac{b \cdot L_d}{AL} \right)$
- L_d and AL are the document length and average document length, measured in bytes
- $k1$ and b are parameters that should initially be set as $k_1 = 1.2$ and $b = 0.75$

As output, your program should produce answer lists using the following format:

```
401 LA010189-0003 1 110.541
401 LA010190-0064 2  95.259
401 LA010389-0019 3  86.806
401 LA010489-0026 4  65.082
401 LA010689-0066 5  21.118
```

where

- column 1 is the query-label (passed to your program as a command-line argument);
- column 2 is the document identifier;
- column 3 is the rank position of the document (starting from rank 1); and
- column 4 is the similarity score produced by the BM25 similarity function (the numbers given are examples only).

As the final line of output, your program should print the running time in milliseconds, for example:

```
Running time: 298 ms
```

2 Query Expansion (35%)

Extend your program `search` to implement automatic query expansion using pseudo-relevance feedback (PRF).

In PRF, the top R documents returned from an initial ranked evaluation of the query are assumed to be relevant. From these top R documents, E terms need to be selected.

An example of how your program should be invoked is:

```
./search  -PRF -q <query-label> -n <num-results> -R <number> -E
<number> <lexicon> <invlists> <map> <queryterm-1> [<queryterm-2> ...
<queryterm-N>]
```

(or equivalent Java invocation) where

- `-PRF` specifies that pseudo relevance feedback is to be used.
- `-q <query-label>` is an integer that identifies the current query. For example, you might indicate that the current query label is "1133". This item is only used for output formatting (see below).
- `<lexicon>` and `<invlists>` are the inverted index files that your `index` program created.
- `<map>` is the mapping table from internal document numbers to actual document identifiers. (See Assignment 1 for details. You are encouraged to re-use your indexing code from Assignment 1, with modifications where required.)
- `-n <num-results>` is an integer number specifying the number of top-ranked documents that should be returned as an answer.
- `-R <number>` specifies the number of top-ranked documents from the initial query that will be used for expansion (i.e. assumed to be relevant for PRF).
- `-E <number>` specifies the number of terms that should be selected from the top R documents and added to the original query.
- `<queryterm-1> [<queryterm-2> ... <queryterm-N>]` are a a variable number of query terms (but at least one query term) that the program will receive as command-line arguments. Each of these terms should be looked up in the lexicon, and the appropriate inverted list data fetched from the inverted list file.

As in the previous section, your search program should be efficient, using suitable data structures and algorithms. As output, your program should produce answer lists using the same format as given in section 1.

To select terms for PRF, you should:

- Evaluate the initial query.

- Fetch the top R documents from disk.
- For each *term* that occurs in the top document set, calculate a TF.IDF weight using the formula:
$$weight = \ln(1 + \frac{N}{f_t}) \cdot (1 + \ln(f_{d,t}))$$
(Note that here the quantities f_t and $f_{d,t}$ refer to frequencies within the top set, and not the whole collection.)
- Append the E highest scoring terms to the original query.
- Run the new query across the collection, and output the results.

3 Report (30%)

Create a plain text (or PDF) file called `report.txt`. Remember to clearly cite any sources (including books, research papers, course notes, etc.) that you referred to while designing aspects of your programs.

3.1 Ranked Retrieval (5%)

Create a heading called "Ranked Retrieval" in your report file. Explain your implementation of the BM25 similarity function. Clearly describe how you process a query of one or more terms. Explain the algorithms and data structures that you have used. *This section should be no longer than a page.*

3.2 Query Expansion (5%)

Create a heading called "Query Expansion" in your report file. Explain your implementation of query expansion. Explain the algorithms and data structures that you have used. *This section should be no longer than a page.*

3.3 Evaluation (20%)

You have implemented two retrieval models. The next task is to evaluate their performance. The file */public/courses/MultimediaInfoRetrieval/2011/a2/topics* contains five queries (one on each line) with the following format:

```
401 foreign minorities germany
402 behavioral genetics
403 osteoporosis
405 cosmic events
408 tropical storms
```

where the first item is a query number (identifier) followed by a space, and the subsequent terms on the line are the query that is to be submitted to your search system.

A corresponding file, */public/courses/MultimediaInfoRetrieval/2011/a2/qrels*, contains relevance judgements for these queries and the collection that you have indexed. These judgements were made by a human assessor, who considered the query and the answer document, and decided whether the document is relevant for the given query. An example line from the file is:

```
401 0 LA010189-0003 0
401 0 LA010489-0026 1
401 0 LA010689-0066 0
```

where the first item is a query number, the second item should be ignored, the third is a document identifier, and the fourth item is a relevance value (a "1" indicates that this document is relevant for the query, and a "0" indicates not relevant). So, the example above indicates that the document `LA010489-0026` is relevant for query number 401, and the documents `LA010189-0003` and `LA010689-0066` are not relevant.

Create a heading called "Evaluation" in your report file, and answer the following questions.

1. Your first task is to run the queries using both of your retrieval models. You should generate answer lists of length **20**. For query expansion, terms should be selected from the top 10 documents, with 20 terms being added to the query.

   To run the first query in the file, for example, you would use the following commands:

   ```
   ./search -BM25 -q 401 -n 20 lexicon invlists map foreign minorities germany
   ./search -PRF -q 401 -n 20 -R 10 -E 20 lexicon invlists map foreign
                              minorities germany
   ```

   You **must** include the full answer lists for both systems in your report.

2. Compare the systems using precision at 10 documents retrieved (P@10) as a metric. Explain this performance measurement, and discuss which system is better based on the scores. (You do not need to carry out statistical significance tests.)

   Is this a sensible metric for comparing the two systems that you have implemented? Why or why not? *You should write no more than half a page for this section.*

3. Based on your knowledge of retrieval evaluation, choose a different evaluation metric to compare the two systems.

   Explain whether query expansion (with $R = 10$, $E = 20$) or plain ranked retrieval is better according to your chosen metric. Describe why your chosen metric is appropriate for comparing the performance of these two systems. *You should write no more than half a page for this section.*

4 Optional Extension: More on Query Expansion (10% Bonus Marks)

- Carry out a small set of experiments investigating the query expansion parameters R and E. (5%)

  In your report, create a section called "Query Expansion Parameters" and discuss your findings. Suggested items to include in your discussion include: the role of the two parameters; which settings lead to good performance for individual queries; which settings lead to good performance on average; and, what general conclusions can be drawn about parameterised query expansion (if any).

  You may wish to include graphs as part of your answer. *Write no more than one page for this section.*

- Propose and implement a *possible* improvement for query expansion. (5%)

  In your report, create a section called "Improved Query Expansion". Briefly explain your proposed improvement, why you would expect it to work, and how you implemented it.

  Evaluate your new query expansion method using the two metrics from section 3.3, reporting the scores and briefly discussing how the technique compares to simple query expansion, and plain ranked retrieval. *Write no more than one page for this section.*

What to Submit, When, and How

The assignment is due on 5:30pm, Friday 7 October 2011.

You should submit your source code, reports, and README file using `turnin`. (Do not submit executables, object files, or data.)

Use the following process:

1. Put all your submission files in a directory
2. Run `turnin` using (*for example*):
   `turnin -c ir -p Assignment2 *.h *.c Makefile README.txt`

   (Your command will differ depending on your choice of report file format and programming language.)
3. You can check your submission using:
   `turnin -c ir -p Assignment2 -v`
   More information about `turnin` can be found by typing `man turnin` on `yallara`.

Your turnin submission must have a timestamp showing 5:30pm, Friday 7 October 2011 (or earlier). Late submissions should be submitted using the same turnin procedure, but be aware that these will suffer a mark deduction as explained in the course guide.