COMP3009J – Information Retrieval
Programming Assignment

This assignment is worth **30% of the final grade** for the module.
Due Date: Friday 31th May 2024 at 23:55 (i.e. end of Week 14)

Before you begin, download and extract the files ``small_corpus.zip'' and ``large_corpus.zip'' from Brightspace. These contain several files that you will need to complete this assignment. The README.md file in each describes the files contained in the archive and their format[1].

The main **objective** of the assignment is to create a basic Information Retrieval system that can perform preprocessing, indexing, retrieval (using BM25) and evaluation.

The small corpus is intended to show the correctness of your code. The large corpus is intended to show the efficiency. Efficiency is only important if the code is firstly correct.

Both corpora are in the same format, **except for the relevance judgments**. For the small corpus, all documents not included in the relevance judgments have been judged non-relevant. For the large corpus, documents not included in the relevance judgments have not been judged.

For this assignment, you should write several independent programs, each of which is contained in one file[2]. The list of programs is below, with descriptions of each. You may choose not to implement all the programs (see the "Grading" section below). However, an A+ grade can only be awarded if all these programs have been written correctly *and* efficiently.

It is **ESSENTIAL** that all programs can be run as a standalone command-line program, without requiring an IDE/environment such as IDLE, PyCharm, Jupyter, etc.

Non-standard libraries (other than the Porter stemmer provided) may not be used. Do not use absolute paths (the path to the corpus will always be provided to your program).

What you should submit

Submission of this assignment is through Brightspace. You should submit a single .zip archive containing the programs you have written.

---

[1] This is a Markdown file. Although you can open and read it as plain text, proper programming editor (e.g. Visual Studio Code) will provide syntax highlighting for better readability.

[2] Here, "independent programs" means that they should not import anything from one another. If you write a function that is helpful in multiple programs, copy/paste it. This is, of course, not good programming practice in terms of reusability of code. However, it helps with the grading process.

Programs:

`index_small_corpus.py`

This program is intended to read the small corpus, process its contents and create an index.

It must be possible to pass the path to the (unzipped) small corpus to this program as a command-line argument named "`-p`"[3]:

```
./index_small_corpus.py -p /path/to/comp3009j-corpus-small
```

This program must perform the following tasks:

1. **Extract the documents** contained in the corpus provided. You must divide the documents into terms in an appropriate way (these are contained in the ``documents'' directory of the corpus. The strategy must be documented in your source code comments.

2. Perform **stopword removal**. A list of stopwords to use can be loaded from the `stopwords.txt` file that is provided in the ``files'' directory of the corpus.

3. Perform **stemming**. For this task, you may use the `porter.py` code in the ``files'' directory.

4. Create an appropriate **index** so that IR using the BM25 method may be performed. Here, an index is any data structure that is suitable for performing retrieval later.

This will require you to calculate the appropriate weights and do as much pre-calculation as you can. This should be stored in a single external file in some human-readable[4] format. Do not use database systems (e.g. MySQL, SQL Server, SQLite, etc.) for this.

The output of this program should be a single index file, stored in the current working directory, named "`21888888-small.index`" (replacing "21888888" with your UCD student number).

[3] This path might, for example be "`/Users/david/datasets/comp3009j-corpus-small`" or "`C:/Users/datasets/comp3009j-corpus-small`".

[4] Here, "human-readable" means some text-based (i.e. non-binary) format. It should be possible to see the contents and the structure of the index using a standard text editor.

`query_small_corpus.py`

This program allows a user to submit queries to retrieve from the small corpus, or to run the standard corpus queries so that the system can be evaluated. The BM25 model must be used for retrieval.

Every time this program runs, it should first load the index into memory (named "21888888-small.index" in the current working directory, replacing "21888888" with your UCD student number), so that querying can be as fast as possible.

This program should offer two modes, depending on a command-line argument named "`-m`". These are as follows:

1. **Interactive mode**

   In this mode, a user can manually type in queries and see the first 15 results in their command line, sorted beginning with the highest similarity score. The output should have three columns: the rank, the document's ID, and the similarity score. A sample run of the program is contained later in this document. The user should continue to be prompted to enter further queries until they type "QUIT".

   Example output is given below.

   Interactive mode is activated by running the program in the following way:

   ```
   ./query_small_corpus.py -m interactive -p /path/to/comp3009j-corpus-small
   ```

2. **Automatic mode**

   In this mode, the standard queries should be read from the ``queries.txt'' file (in the ``files'' directory of the corpus). This file has a query on each line, beginning with its query ID. The results[5] should be stored in a file named "218888880-small.results" in the current working directory (replacing "21888888" with your UCD student number), which should include four columns: query ID, document ID, rank and similarity score. A sample of the desired output can be found in the "`sample_output.txt`" file in the "`files`" directory in the corpus.

   Automatic mode is activated by running the program in the following way:

   ```
   ./query_small_corpus.py -m automatic -p /path/to/comp3009j-corpus-small
   ```

[5] You will need to decide how many results to store for each query.

evaluate_small_corpus.py

This program calculates suitable evaluation metrics, based on the output of the automatic mode of `query_small_corpus.py` (stored in "`218888880-small.results`" in the current working directory (replacing "21888888" with your UCD student number).

The program should calculate the following metrics, based on the relevance judgments contained in the ``qrels.txt'' file in the ``files'' directory of the corpus):

- Precision
- Recall
- R-Precision
- P@15
- NDCG@15
- MAP

The program should be run in the following way:

```
./evaluate_small_corpus.py -p /path/to/comp3009j-corpus-small
```

index_large_corpus.py

This program should perform the same tasks as `index_small_corpus.py`, except that the output file should be named "`21888888-large.index`" (replacing "21888888" with your UCD student number).

query_large_corpus.py

This program should perform the same tasks as `query_small_corpus.py`, except that the output results file should be named "`21888888-large.results`" (replacing "21888888" with your UCD student number).

evaluate_large_corpus.py

In addition to the evaluation metrics calculated by `evaluate_small_corpus.py`, this program should **also calculate bpref** (since the large corpus has incomplete relevance judgments).

Otherwise, this program should perform the same tasks as `evaluate_small_corpus.py`, except that the input results file should be named "`21888888-large.results`" (replacing "21888888" with your UCD student number).

Sample Run (Interactive)

```
$ ./query_small_corpus.py -m interactive -p /Users/david/comp3009j-corpus-small
Loading BM25 index from file, please wait.
Enter query: library information conference

Results for query [library information conference]
1 928 0.991997
2 1109 0.984280
3 1184 0.979530
4 309 0.969075
5 533 0.918940
6 710 0.912594
7 388 0.894091
8 1311 0.847748
9 960 0.845044
10 717 0.833753
11 77 0.829261
12 1129 0.821643
13 783 0.817639
14 1312 0.804034
15 423 0.795264

Enter query: QUIT
```

Note: In all of these examples, the results, and similarity scores were generated at random for illustration purposes, so they are **not** correct scores.

Sample Run (Evaluation)

```
$ ./evaluate_large_corpus.py -p /Users/david/comp3009j-corpus-large

Evaluation results:
Precision:   0.138
Recall:      0.412
R-precision: 0.345
P@15:        0.621
NDCG@15      0.123
MAP:         0.253
bpref:       0.345
```

Grading

Grading is based on the following (with the given weights)[6]:

- Document reading and preprocessing: 15%
- Indexing: 20%
- Retrieval with BM25: 20%
- Evaluation: 15%
- Efficiency: 15% (as evidenced by the performance on the large corpus)
- Programming style (comments/organisation): 15%

Other notes

1. This is an *individual* assignment. All code submitted must be your own work. Submitting the work of somebody else or generated by AI tools such as ChatGPT is **plagiarism**, which is a serious academic offence. Be familiar with the UCD Plagiarism Policy and the UCD School of Computer Science Plagiarism Policy.

2. If you have questions about what is or is not plagiarism, ask!

Document Version History

v1.0: 2024-04-26, Initial Version.

[6]This assignment will be graded using the "**Alternative Linear Conversion Grade Scale 40% Pass"** Mark to Grade Conversation Scale**:** https://www.ucd.ie/students/exams/gradingandremediation/understandinggrades/)