# COURSEWORK 1 DETAILS

**Based on lectures 3-7**

DOWNLOAD Coursework 1 data (collection and queries) from:

https://www.inf.ed.ac.uk/teaching/courses/tts/CW/CW1/CW1collection.zip

## IMPORTANT DATES

## SUBMISSION DEADLINE: MONDAY, 15 FEBRUARY 2021, 11:59PM

## ASSIGNMENT OBJECTIVES

Implement a simple IR tool that does the following:

- Pre-processes text
  - Tokenisation
  - Stopword removal using this list of words. You MUST use this list.
  - Porter stemming. You can use packages for this part, such as Snowball or NLTK.
- Creates a positional inverted index
  Your index can have whatever structure you like, and can be stored in any format you like, but you will need to output it to a text file using the format specfied below.
- Uses your positional inverted index to perform:
  - Boolean search
  - Phrase search
  - Proximity search
  - Ranked IR based on TFIDF

## ADDITIONAL DETAILS

- Use the collections from Lab2 for testing your system. You can download from here. Focus on the trec collection which contains 1000 sample news articles. *Note 1:* use the file in XML format. You need to make your code able to parse it. It worth noting that the XML format is a standard TREC which might not be parsed directly using XML parsers. You might need to add header and footer to the file to make it parsable by existing tools. It is allowed to do so if needed (or feel free to code your own parser).
  *Note 2:* For the trec collection, please include the headline of the article to the index. Simply the document text should include the headline and text fields. For positions of terms, please start counting from the headline and continue with the text.
  *Note 3:* Term position is counted AFTER stop words removal.
- The **test collection** to be released 4 days before the deadline. It will be of the same exact format of trec.sample.xml. The size of this collection will be around 5000 documents (5 times of the current version). If your system is running fine with the current collection, it should be straightforward to run smoothly with the new collection.

- For **tokenisation**, you can simply split on every non-letter character, or you can have special treatment for some cases (such as **-** or **'**). Please explain in your report your selections and why you did so.
- For **stopping** Please use the stop words list mentioned above.
- Again, for **stemming**, you do NOT need to write your own stemmer. Use any available package for Porter stemmer. Write down in your report which one used. You need to use Porter stemmer, not anything else.
- For the TFIDF search function, please use the formula from lecture 7, slide 13 with title "TFIDF term weighting". **Note:** this is different from other implementations of TFIDF in some toolkits, so please implement it as shown in lecture.
- Please use the queries in lab 2 and lab 3 for testing your code. A new list of queries to be released with the collection 4 days before the deadline. These 4 days should be more than enough to run the new queries and get the results.
- Notes about the expected queries:
  - Queries are expected to be very similar to that in the labs.
  - Two query files will be provided, one for Boolean search, and the other for ranked retrieval.
  - Boolean queries will not contain more than one "AND" or "OR" operator at a time. But a mix between phrase query and one logical "AND" or "OR" operator can be there (like query q9 in lab 2). Also "AND" or "OR" can be mixed with NOT, e.g. Word1 AND NOT Word2.
  - 10 queries would be provided in a file named `queries.boolean.txt` in the following format:

    ```
    1 term11 AND term12
    2 "term21 term22"
    ```

  - For proximity search queries, it will have the format "`#15(term1,term2)`", which means find documents that have both term1 and term2, and distance between *term1* and *term2* is less than or equal to 15 (after stop words removal).
  - 10 free text queries for ranked retrieval will be provided in a file named `queries.ranked.txt` in the following format:

    ```
    1 this is a sample query
    ```

## SUBMISSIONS AND FORMATS

You need to submit the following 5 files:

1. **`index.txt`**: a formatted version of your positional inverted index. Each line of this file describes a token, a document it appears in, and its position within that document:

   ```
   term:df
      docID: pos1, pos2, ....
      docID: pos1, pos2, ....
   ```

   where df is the document frequency of the term. Example:

   ```
   newspaper:2
      23: 2,15
      93: 234
      .......
   ```

   Here, the token "newspaper" appeared in 2 documents, where it appeared in document 23 twice at positions 2 and 15, and document 93 once in position 234. Each document ID (docID) should be in a separate line that starts with a "tab" ("\t" in Python). Positions of term should be separated by a comma. Lines should end with a line break ("\n" in Python).

2. **`results.boolean.txt`**: contains results of the `queries.boolean.txt` in the following `query_number,document_id` format:

```
1,710
1,213
2,103
```

This means that for query "1" you retrieved two documents - numbers "710" and "213".
For query "2" you retrieved one document - "103".
The two values on each line should be separated by a comma. Lines should end with a line break ("\n" in Python).
Your boolean results file should list every matching document, per query.

3. **`results.ranked.txt`**: contains results of the `queries.ranked.txt` in the following `query_number,document_id,score` format:

```
1,710,0.6234
1,213,0.3678
2,103,0.9761
```

This means that for query "1" you retrieved two documents - document number "710" (with score 0.6234) and document number "213" (with score 0.3678).
For query "2" you retrieved one document, number "103", with score 0.9761.
Scores should be rounded to four decimal places.
The three values on each line should be separated by a comma. Lines should end with a line break ("\n" in Python).
Print results for queries in order of their score - that is, all results for query "1" are sorted by score, then results for query 2, 3, ... 10.
Your ranked results file should list only up to the top **150** results per query.

4. **`code.py`**: a single file containing the code used to generate `index.txt`, `results.boolean.txt` and `results.ranked.txt`.
If you will use something other than Python, let us know before submission!
Please try to make your code as readable as possible: commented code is highly recommended.
Please **DO NOT** submit the collection file with the code!

5. **`report.pdf`**: Your report on the work.
This is 2 to 4 pages and should include:
- details on the methods you used for tokenisation and stemming
- details on how you implemented the inverted index
- details on how you implemented the four search functions
- a brief commentary on the system as a whole and what you learned from implementing it
- what challenges you faced when implementing it
- any ideas on how to improve and scale your implementation

Your report **SHOULD NOT** contain any code or screenshots of code.
It should be a high-level description of how your code works and the decisions you made while implementing your information retrieval system.

Submit ONLY these five files! Do not submit any of the assignment files.