COMP4650/6490 Document Analysis – Semester 2 / 2023

Assignment 1

Due 17:00 on Wednesday 16 August 2023 AEST (UTC +10)

Last updated July 28, 2023

Overview

In this assignment, your task is to implement a basic boolean and ranked information retrieval (IR) system using a document collection, and then measure search performance based on predefined queries. A document collection containing more than $30,000$ government site descriptions is provided for this assignment, along with a set of queries (in file `gov/topics/gov.topics`) and the expected returned documents (in file `gov/qrels/gov.qrels`). The provided code implements most of an IR system. Throughout this assignment you will make changes to the provided code to improve or complete existing functions. Note that the provided code is designed to be simple to understand and modify, it is not efficient nor scalable. When developing a real-world IR system you would be better off using high performance software such as Apache Lucene[1].

Throughout this assignment:

1. You will develop a better understanding of indexing, including the tokeniser, parser, and normaliser components, and how to improve the search performance given a predefined evaluation metric;
2. You will develop a better understanding of search algorithms, and how to obtain better search results, and
3. You will find the best way to combine an indexer and search algorithm to maximise the performance of an IR system.

Submission

- You will produce an answers file with your responses to each question. Your answers file must be a PDF file named `u1234567.pdf` where `u1234567` should be replaced with your Uni ID.
- The answers to this assignment (including your code files) have to be submitted online in Wattle.
- You should submit a ZIP file containing all of the code files and your answers PDF file, **BUT NO DATA**.
- **No late submission will be permitted without a pre-arranged extension**. A mark of 0 will be awarded if not submitted by the due date.

Marking

This assignment will be marked out of 100, and it will contribute 10% of your final course mark.

Your answers to coding questions will be marked based on the quality of your code (is it efficient, is it readable, is it extendable, is it correct) and the solution in general (is it appropriate, is it reliable, does it demonstrate a suitable level of understanding).

Your answers to discussion questions will be marked based on how convincing your explanations are (are they sufficiently detailed, are they well-reasoned, are they backed by appropriate evidence, are they clear, do they use appropriate aids such as tables and plots where necessary).

[1]`https://lucene.apache.org/`

This is an individual assignment. Group work is not permitted. Assignments will be checked for similarities.

Question 1: Implement Boolean Queries (40%)

The construction of inverted index is implemented in `indexer.py`. You should first run `indexer.py` to store the following index data:

- A dictionary (called `index`) mapping a token string to a sorted list of (`doc_id, term_frequency`) tuples,
- a dictionary `doc_freq` mapping a token string to its document frequency,
- a dictionary (called `doc_ids`) mapping a `doc_id` to the path of the document, and
- the number of documents in the collection (called `num_docs`).

Please double check that you are using `process_tokens_original` within the function `process_tokens` in `string_processing.py` before running `indexer.py`.

Your task is to implement the ability to run boolean queries on the built index. The starting code is provided in `query_boolean.py`. Specifically, you shall implement a simplified boolean query grammar. You may assume that input queries consist of only `AND` and `OR` operators separated by single tokens. For example, `cat AND dog` is a valid query while `cat mask AND dog` is not a valid query since `cat mask` is not a single token. You are not required to implement `NOT`. The order of operations will be left to right with no precedence for either of the operators, for example the query `cat AND dog OR fish AND fox` should be done in the following order: `((cat AND dog) OR fish) AND fox`. The brackets are provided as an example; you can assume that the queries provided to your system will not contain brackets. To score full marks on this question, your solution should implement $O(n+m)$ sorted list intersection and sorted list union algorithms – $O(n+m)$ sorted list intersection was covered in the lectures, while union is very similar – where n and m refer to the lengths of the two lists. **Solutions using data structures such as sets or dictionaries to implement the intersection and union operations will not score full marks**.

Once you have completed your boolean query system please run it on the following queries and list the relative paths (e.g., `./gov/documents/14/G00-14-2198849`) of the retrieved documents in your answers PDF file.

HINT: none of the queries below give more than 10 *results, and Query* 0 *has been done for you so that you can check your system.*

Query 0: `Workbooks`
Answer:

- `./gov/documents/14/G00-14-2198849`
- `./gov/documents/36/G00-36-2337608`
- `./gov/documents/50/G00-50-0062475`
- `./gov/documents/69/G00-69-1400565`
- `./gov/documents/69/G00-69-2624147`

Query 1: `Australasia OR Airbase`
Query 2: `Warm AND WELCOMING`
Query 3: `Global AND SPACE AND economies`
Query 4: `SCIENCE OR technology AND advancement AND PLATFORM`
Query 5: `Wireless OR Communication AND channels OR SENSORY AND INTELLIGENCE`

Make sure you submit your code for this question (i.e., `query_boolean.py`) as well as your answers.

Question 2: Implement TF-IDF Cosine Similarity (30%)

Your task is to implement the `get_doc_to_norm` function and the `run_query` function in `query_tfidf.py` to use TF-IDF and the cosine similarity applied to TF-IDF. In your solution both the query and the document vectors should be TF-IDF vectors. Your implementation could be similar to the `get_doc_to_norm` and `run_query` functions in `query.py` but should use TF-IDF instead of term frequency.

The TF-IDF variant you should implement is:

$$\text{TF-IDF} = n_t \ln \frac{N}{1 + n_d},$$

where n_t is the term frequency, n_d is the document frequency, and N is the total number of documents in the collection. This is almost the standard TF-IDF variant, except that 1 is added to the document frequency to avoid division by zero errors.

Once you have implemented TF-IDF cosine similarity, run the `query_tfidf.py` file and record the top-5 retrieved documents as well as their similarity scores in your answers PDF file for two queries below:

Query 1: `Is nuclear power plant eco-friendly?`
Query 2: `How to stay safe during severe weather?`

You should then run `evaluate.py` to evaluate the query results (for the queries in `gov/topics/gov.topics`) against the ground truth, and record the evaluation results in your answers PDF file. Make sure you submit your `query_tfidf.py`.

Question 3: Explore Linguistic Processing Techniques (30%)

For this question you will explore ways to improve the `process_tokens` function in `string_processing.py`. The current function removes stopwords and lowercases the tokens. You should modify the function and explore the results. To modify the function, you should make changes to the functions `process_token_1`, `process_token_2`, and `process_token_3` and then uncomment the one you want to test within the main `process_tokens` function. You should pick **at least three different modifications** and evaluate them (you can add new process tokens functions if you want to evaluate more than three modifications). See lectures for some possible modifications. You might find the Python `nltk` library useful. The modifications you make do not need to require significant coding, the focus of this question is choosing reasonable modifications and explaining the results.

To evaluate each modification you made, you should

(1) run `indexer.py` to rebuild the index data, then
(2) run `query.py` (**not** `query_tfidf.py`), and
(3) run `evaluate.py` to evaluate the query results.

For each of the modification you make you should describe in your answers:

- What modifications you made.
- Why you made them (in other words why you thought they might work).
- What the new performance is.
- Why you think the modification did/did not work. Making sure to give (and explain) examples of possible failure or success cases.

Finally, you should **compare all the modifications** by choosing one appropriate metric and decide which modification (or combination of modifications) performed the best. Your comparison should **make use of a table or chart** as well as some **discussion**. Make sure to report all of this and your justification in your answers, and to submit your `string_processing.py` showing each of the changes you made.