COMP4650/6490 Document Analysis – Semester 2 / 2022

Assignment 1

Due 17:00 on Wednesday 17 August 2022 AEST (UTC +10)

Last updated July 25, 2022

Overview

In this assignment, your task is to index a document collection into an inverted index, 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 (gov/topics/gov.topics) and the expected returned documents (gov/qrels/gov.qrels). The provided code implements most of an information retrieval system. This 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 tokenizer, 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 your performance.

Submission

- The answers to this assignment (including your modified code files) have to be submitted online in Wattle, see the link **Assignment 1 Submission** in Week 4 (*15 to 19 August*).

- You can edit your answers many times and they will be saved by Wattle.

- Note that **Wattle does not allow us to access any earlier edited versions of your answers, so check very carefully what you submit as the final version**.

- **You can only submit your assignment once. Make sure you submit the final version of your assignment answers before the submission deadline**.

Marking

This assignment will be marked out of 15, and it will contribute 15% 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/

Question 1: Implement the Indexer and Evaluate (3 marks)

Your first task is to implement index construction. You should complete the function stub `index_from_tokens` in `indexer.py`. This function takes as input a sorted list of tuples of the form (`token_string`, `doc_id`), indicating that the token `token_string` was in the document represented by `doc_id`. For each occurrence of a token in a document there is a tuple in this list – this means duplicate tuples indicate term count. **The input list is sorted in ascending order by** `token_string` **then** `doc_id`.

Once complete the `index_from_tokens` function should output two dictionaries, the first dictionary `index` is a mapping from token strings to a sorted list of (`doc_id`, `term_frequency`) tuples. These lists should have their elements sorted in ascending order by `doc_id`, and contain only unique `doc_ids` – duplicates are used to compute term frequency. The second dictionary `doc_freq` is a mapping from `token_string` to document frequency.

Once you have implemented `index_from_tokens` you should run `indexer.py` to store the index, then run `query.py` to run a set of test queries, finally run `evaluate.py` to evaluate the query results against the ground truth. Record your evaluation results in your answers and make sure you submit your `indexer.py`.

Example Input for `index_from_tokens`:
`[("cat", 1), ("cat", 1), ("cat", 2), ("door", 1), ("water", 3)]`

Example Output for `index_from_tokens`:
`index: {"cat": [(1, 2), (2, 1)], "door": [(1, 1)], "water": [(3, 1)]}`
`doc_freq: {"cat": 2, "door": 1, "water": 1}`

Question 2: Implement TF-IDF Cosine Similarity (3 marks)

Currently `query_tfidf.py` uses cosine similarity applied to term frequency (it is currently a copy of `query.py` but you will change that). Your task is to implement cosine similarity applied to TF-IDF. In your solution both the query and the document vectors should be TF-IDF vectors. You will need to modify the `run_query` function and the `get_doc_no_norm` both of which are in `query_tfidf.py`.

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, then run `evaluate.py`, and record the results in your answers. Make sure you submit your `query_tfidf.py`.

Question 3: Explore Linguistic Processing Techniques (3 marks)

For this question you will explore ways to improve the `process_tokens` function in `string_processing.py`. The current function removes stopwords. 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.

Note: for this question you should use the unmodified `query.py` **file. Do not use your TF-IDF implementation**.

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** using 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 answer.

Make sure to submit your `string_processing.py` showing each of the changes you made.

Question 4: Implement Boolean Queries (3 marks)

Your task is to implement the ability to run boolean queries on the existing index. The starting code is provided in `query_boolean.py`. Specifically, you will 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/92/G00-92-0775281`) of the retrieved documents in your answers (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). Please **double check** that you are **not** using an index built with a modified version of `process_tokens` in `string_processing.py`.

Query 0: `Welcoming`
Answer:

- `./gov/documents/31/G00-31-2565694`
- `./gov/documents/42/G00-42-4180551`
- `./gov/documents/85/G00-85-0255215`
- `./gov/documents/86/G00-86-2161870`
- `./gov/documents/86/G00-86-4087434`
- `./gov/documents/97/G00-97-2878104`
- `./gov/documents/98/G00-98-1962568`

Query 1: `Australasia OR logistic`
Query 2: `heart AND warm`
Query 3: `global AND space AND wildlife`
Query 4: `engine OR origin AND record AND wireless`
Query 5: `placement AND sensor OR max AND speed`

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

Question 5: Evaluating IR Systems (3 marks)

Answer the following two questions:

(a) Explain how you can evaluate your boolean query system.

(b) Now consider the case where the output of your boolean query system is going to be re-ranked and only the top 10 results displayed. Explain how you could evaluate this combined boolean query and re-ranking system.

Your answers to both (a) and (b) should at a minimum consider:

- What data you would need (e.g. queries and ground truth);
- The challenges that you would have to face getting this data;
- What metrics are appropriate and why these metrics are appropriate.