COMP4650/COMP6490 Document Analysis
2025 Semester 2

Assignment 1

Due 23:55 on Friday 15 August 2025 AEST (UTC/GMT +10)

Overview

This assignment consists of two tasks. First, you will implement a ranked information retrieval (IR) system using a document collection, and then measure the retrieval performance based on a set of predefined queries and their relevance judgments. Next, you will implement a logistic regression text classifier for a text classification problem, using TF-IDF features.

Throughout this assignment, you will develop a better understanding of

1. inverted index used in information retrieval, including the text pre-processing steps,
2. some standard document retrieval algorithms,
3. how ranked retrieval results are evaluated using different metrics,
4. how machine learning models are trained in practice, including partitioning of datasets, evaluation, and tuning hyper-parameters, and
5. how `scikit-learn`[1] package can be used for text classification.

Submission

- The answers to this assignment (including your Python code files) have to be submitted online through Wattle.
- 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. You will also need to submit several Python files that you have modified as part of the submission. Specifically, follow the instructions below to prepare your final submission file:
  1. Create a folder named after your Uni ID (e.g., u1234567).
  2. Inside the folder, place the following files: your answers PDF (e.g., u1234567.pdf), `query_tfidf.py`, `string_processing.py`, `query_bm25.py`, `features.py`, `classifier.py`.
  3. Compress the folder into a ZIP file named after your Uni ID (e.g., u1234567.zip)
- Submit this single ZIP file to Wattle. **Please make sure NOT to include any data file.**
- **No late submission will be permitted without a pre-arranged extension**. A mark of 0 will be awarded if your answers (including your code files) are not submitted by the due date without a valid extension request.

[1] `https://scikit-learn.org/`

Marking

This assignment will be marked out of 20, and it will count towards 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 clearly written, are they sufficiently detailed, are they well-reasoned, are they backed by appropriate evidence, do they use appropriate aids such as tables and plots where necessary).

This is an individual assignment. Group work is not permitted. Assignments will be checked for similarities. You are allowed to use generative AI tools to help you with non-essential parts of the assignment, such as to check how a Python package or function works, to understand a concept you are not sure about, or for proofreading. You are not allowed to use any generative AI tool to help you directly answer the questions (including writing code for you). Please also refer to Question 3.

Question 1: Information Retrieval (13 marks)

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 or scalable. When developing a real-world IR system you would be better off using high performance software such as Apache Lucene[2].

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`.

After running `indexer.py` to build the index (which creates a file called `my_index.pkl` in the current directory), run `query.py` followed by `evaluate.py`. The Python program `query.py` uses TF-weighted cosine similarity to retrieve a ranked list of documents for the queries in `gov/topics/gov.topics`, and the retrieved results can be found in the file `retrieved_documents.txt`. The Python program `evaluate.py` uses the package `trectools` to calculate various IR evaluation metrics on the retrieved results.

You have three sub-tasks in this question, as detailed below.

Question 1.1: TF-IDF Cosine Similarity (4 marks)

Your first task in this question is to implement a TF-IDF-weighted cosine similarity retrieval function. To do so, you will need to implement the `get_doc_to_norm` function and the `run_query` function in `query_tfidf.py`. In your solution both the query and the document vectors should be TF-IDF vectors.

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

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:

$$\text{TF-IDF}_{t,d} = \text{TF}_{t,d} \times \text{IDF}_t = \text{TF}_{t,d} \times \ln \frac{N}{1 + \text{DF}_t},$$

where t is a term, $\text{TF}_{t,d}$ is the term frequency of t inside document d, DF_t is the document frequency of t, and N is the total number of documents in the collection. This is almost the standard TF-IDF variant that we have introduced in the lecture, 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 the two queries below. (The two queries have already been hardcoded in the file query_tfidf.py.)

Query 1: `Food Safety`
Query 2: `Ozone Layer`

The Python program query_tfidf.py also performs retrieval for the queries in gov/topics/gov.topics and stores the retrieval results in retrieved_documents.txt.

You should now run evaluate.py to evaluate the query results returned by query_tfidf.py and record the evaluation results (i.e., the values of the various metrics including map, RPrec, recip_rank, etc.) in your answers PDF file.

Make sure you submit your query_tfidf.py in your ZIP file.

Question 1.2: Text Pre-processing Techniques (4 marks)

For this question you will explore ways to improve the process_tokens function in string_processing.py. The current function only turns tokens into lowercase. You should modify the function to explore more text pre-processing techniques. To modify the function, you should make changes to the function process_token_new and then uncomment the corresponding line of code in the process_tokens function.

We expect you to implement the following pre-processing steps and experiment with their combinations:

- Stemming. We have already imported the Porter Stemmer for you to use.
- Removing the punctuation marks in the tokens. We have already provided a translation table that can be used by str.translate for this purpose.
- Removing stop words. We have already provided code that takes the English stop word list from NLTK for you to use.

You should consider in what order these pre-processing steps are to be performed and implement the function process_token_new accordingly.

The modifications you need to make to implement the three pre-processing techniques above do not require significant coding. The focus of this question is experimenting with these pre-processing techniques and explaining the results.

To evaluate any combination of the three steps above, you should

(1) run indexer.py to rebuild the index data, then
(2) run query_tfidf.py, and
(3) run evaluate.py to evaluate the query results.

Answer the following questions in your answers PDF:

(A) If all three techniques are to be used, in which order should they be applied (e.g., should stemming be done before or after stop word removal)? Why?

(B) When all three techniques are used, record the top-5 retrieved documents as well as their similarity scores in your answers PDF file for the same two queries below:

Query 1: `Food Safety`
Query 2: `Ozone Layer`

Inspect the top-5 retrieved documents for each query and compare them with those you obtained in Question 1.1. For those documents that are retrieved in both cases, compare their similarity scores to the query and state whether the similarity scores have generally increased or decreased after the three text pre-processing steps. Explain what you think might caused the increase or decrease that you have observed.

(C) Choose one of the evaluation metrics as your main metric for comparison. Explain what this metric measures. Then compare the performance of different combinations of the three pre-processing techniques using the metric you have chosen. You should use a table, a plot, or a chart to illustrate the comparison. Sate which combination you find to be the best.

Make sure you submit your `string_processing.py` in your ZIP file.

Question 1.3: The Okapi BM25 Ranking Function (5 marks)

The document ranking function that uses TF-IDF weighting and cosine similarity in Question 1.1 is easy to understand and generally works well in practice. However, it is not the most effective ranking function. A well-known ranking function that has been shown to perform very well empirically is the Okapi BM25 ranking function.[3] This ranking function is based on the probabilistic retrieval framework developed in the 1970s and 1980s. Without explaining the theory behind, let us look at how this ranking function looks like:

$$\text{score}(q, d) \quad = \quad \sum_{t \in q} \text{IDF}_t \cdot \frac{\text{TF}_{t,d} \cdot (k_1 + 1)}{\text{TF}_{t,d} + k_1 \cdot (1 - b + b \cdot \frac{|d|}{\text{avgdl}})}.$$

Here q is a query and d is a document. t is a term in the query. IDF_t is the inverse document frequency of t and $\text{TF}_{t,d}$ is the term frequency of t in d. $|d|$ is the document length, i.e., the number of words in d, and avgdl is the average document length in the collection. k_1 and b are parameters to be manually set.

Inside the file `query_bm25.py`, implement the function `run_query` that returns a ranked list of documents for a given query based on the Okapi BM25 ranking function above. The function can be implemented in a way similar to the `run_query` function in `query_tfidf.py`, but you need to sort the documents based on the scoring function above. Also implement the function `get_doc_to_length` in `query_bm25.py`, which pre-computes the length of each document and the average document length to be used by `run_query`.

After the two functions in `query_bm25.py` are implemented, you should

(1) run `indexer.py` to rebuild the index data (using the best combination of pre-processing steps that you have found), then
(2) run `query_bm25.py`, and
(3) run `evaluate.py` to evaluate the query results.

In your answers PDF, report the evaluation results in terms of all the metrics that `evaluate.py` returns.

[3] See `https://en.wikipedia.org/wiki/Okapi_BM25`.

Finally, in your answers PDF, use a table, a plot, or a chart to compare the performance of the following systems that you have tested earlier. Include all the performance metrics that `evaluate.py` returns in your table/plot/chart.

- Sys 1: Tokenisation with `process_tokens_original`, TF-weighted cosine similarity
- Sys 2: Tokenisation with `process_tokens_original`, TF-IDF-weighted cosine similarity
- Sys 3: Tokenisation with `process_tokens_new` and your best combination of pre-processing techniques, TF-IDF-weighted cosine similarity
- Sys 4: Tokenisation with `process_tokens_new` and your best combination of pre-processing techniques, BM25 ranking function

In your answers PDF, briefly discuss how each of the techniques you have implemented (TF-IDF weighting, text pre-processing, and BM25 ranking) have affected the retrieval performance.

Make sure you submit your `query_bm25.py` in your ZIP file.

Question 2: Text Classification (7 marks)

In this question, you are provided with a file called `data_file.csv` that contains documents labeled with either `A` or `B`. These documents are extracted from a well-known dataset called 20newsgroups for studying text classification.[4]

Question 2.1: Understanding the Two Classes (3 marks)

We did not provide meaningful class labels for the two classes of documents in `data_file.csv`. Although you can randomly sample some documents from each class and read them to get an understanding of the two classes, your random sample of documents may not reflect the whole collection of documents. Your first task in Question 2 is to come up with an approach that processes the given data file and collects some corpus statistics from the two classes of documents (e.g., the most common words) to help you quickly summarise what each class of documents is about. Implement your approach using a Python script (which you do not need to submit).

In your answers PDF, explain what your approach is, present the corpus statistics you have collected, and state how from the corpus statistics you can infer the topic of each class of documents.

Question 2.2: Training and Testing a Classifier (4 marks)

A simple approach to classifying documents is to train a logistic regression classifier using TF-IDF features. This approach is relatively straightforward to implement and can be very hard to beat in practice.

To do this you should first implement the `get_features_tfidf` function (in `features.py`) that takes a set of training sentences as input and calculates the TF-IDF (sparse) document vectors. You may want to use the `TfidfVectorizer`[5] in the `scikit-learn` package. You should use it after reading the documentation. For text pre-processing, you could set the `analyzer` argument of `TfidfVectorizer` to the `tokenise_text` function provided in `features.py`. Alternatively, you may set appropriate values to the arguments of `TfidfVectorizer` or re-use the text pre-processing code from Question 1.

Next, implement the `search_C` function in `classifier.py` to try several values for the regularisation parameter C and select the best based on the accuracy on the validation data. The `train_model` and

[4] See `https://archive.ics.uci.edu/dataset/113/twenty+newsgroups`.

[5] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

eval_model functions provided in the same Python file might be useful for this task. To try regularisation parameters, you should use a hyper-parameter search method presented in the lectures.

You should then run text_classification.py, which first reads in the dataset and splits it into training, validation and test sets; then it trains a logistic regression classifier and evaluate its performance on the test set. Make sure you first uncomment the line with the text_classification_tfidf function (which uses your get_features_tfidf function to generate TF-IDF features, and your search_C function to find the best value of C) in the top-level code block of text_classification.py (i.e., the block after the line "if __name__ == '__main__':") and then run text_classification.py.

Answer the following questions in your answers PDF:

(A) What search technique did you use and what range of values for C did you try?

(B) Use a table to present the performance under the different values of C. What was the best performing C value?

(C) What was your accuracy on the test set?

Make sure you submit your features.py and classifier.py in your ZIP file.

Q3 (Optional): Use of Generative AI Tools (0 mark)

This question is not graded but it allows us to better understand how generative AI is being used to facilitate learning.

If you have used any generative AI tool to help you with this assignment (e.g., to use it to look for suitable Python packages to use, to understand how a function works, or to understand a new concept such as Okapi BM25), briefly describe how you have used AI tools in your answers PDF.

If you have not used any generative AI tool for this assignment, you do not need to answer Question 3.