CS 469 Information Retrieval Assignment 1

Instructor: Dr. Prajna Upadhyay

Deadline: 29 February, 2024, 11:55 PM

This assignment has to be done in groups of 4 people. Read the statement very carefully. Total points: 30

1 Datasets and Prerequisites

In this assignment, you will implement **boolean retrieval** with multiple **indexing** methods. Refer to the code and dataset that can be downloaded from the zip file here. Section 1.1 describes the datasets, 1.2 describes the steps to run the given code, and 3 defines the problem statement.

1.1 Datasets

The dataset comprises of 8K research papers (or documents), 100 queries downloaded from Semantic Scholar [1], 30 wildcard queries, and 30 wildcard boolean queries. These files are located inside `s2/` folder of the zip:

1. `s2/s2_doc.json`: All (8K) research papers in the corpus. Each research paper has 3 fields.
   (a) `docno`: A unique document id for each research paper
   (b) `title`: Title of the research paper
   (c) `paperAbstract`: Abstract of the research paper

   A single paper entry looks like:

```
{
"docno": "6e4eddf4d6671c37537bb5d1c9623353b62e8531",
   "title": [
   "Duality Theorems for Finite Structures (Characterising
       Gaps and Good Characterisations)"
        ],
   "paperAbstract": [
```

[1] https://www.semanticscholar.org/

```
    "We provide a correspondence between the subjects of
        duality and density in classes of finite relational
        structures. ......"
    ]
}
```

2. `s2/s2_query.json`: All the 100 queries. Each query has 2 fields:
   (a) `qid`: The unique id of the query
   (b) `query`: The text of the query

   Some query entries from the file are:

```
{
      "qid": "1",
      "query": "deep learning"
},
{
      "qid": "2",
      "query": "artificial intelligence"
},
{
      "qid": "3",
      "query": "information retrieval"
},
.
.
.
```

3. `s2/s2_wildcard.json`: A set of 30 wildcard queries, each query contains only 1 term

4. `s2/s2_wildcard_boolean.json`: A set of 30 wildcard queries containing more than 1 term. This can be represented as a boolean query as described in Section 3.5

1.2 Prerequisites (Steps to be followed before starting to code)

1. Download the zip file. A `main.py` file is present under the extracted folder. It has been tested with `Python 3.10`, so it is recommended to use the same version for subsequent steps.

2. Run `python main.py`. If everything goes well, you will see the following output:

```
Dictionary size: 73759

Process finished with exit code 0
```

This means your indexes were successfully created.

3. Navigate to `s2/intermediate/` folder where the indexes are stored.

   `s2/intermediate/postings.tsv` is the inverted index file.

2 Profiling your code

Code profiling tools help you assess the time and memory needs for different segments of your code. `cProfile` measures the time taken by different funtions in a python code. `mem-profiler` returns the memory consumed by your code. More details are provided in the docs `cProfile` and `mem-profiler` for analyzing a python code.

3 Problem Statement

There are 5 experiments. Points for each are written beside the experiment name within brackets. You have to measure the time and memory needs for the experiments by benchmarking against the provided queries.

P.S. Include your system memory and processor details when reporting times.

3.1 Experiment 1: Comparing `grep` with inverted index based boolean retrieval (Total 5, 2 for report)

This should be benchmarked on `s2/s2_query.json`. Implementation correctness and report findings will be examined during the demo.

1. Execute each query in `s2/s2_query.json` with boolean retrieval model using the indexes constructed (steps described in Section 1.2). Convert a query, for instance, `deep learning` to a boolean query `<deep and learning>` by replacing the white spaces with `and` operator.

2. Execute each query using a `grep` command.

3. For both the methods, note the individual, maximum, minimum, and average time taken. Introduce a timeout after 2 minutes for each query. Use profilers when using the python code.

3.2 Experiment 2: Linguistic post-processing of the vocabulary (Total 3, 1 for report)

Perform stemming, lemmatization, and stop word removal on the vocabulary. No benchmarking is needed. Results will be examined during demo.

3.3 Experiment 3: Compare hash-based and tree-based implementation of dictionaries (Total 5, 2 for report)

To be benchmarked on `s2/s2_query.json`. Implementation correctness and report findings will be examined during the demo.

1. Implement the dictionary using a tree-based data structure (b-tree or trie).
2. Perform boolean retrieval using this dictionary for all 100 queries. Note individual, maximum, minimum, and average time taken. Compare time and memory consumed with hash-based implementation.

3.4 Experiment 4: Wild card querying using Permuterm and Tree based Indexes (Total 12, 5 for report)

To be benchmarked on `s2/s2_wildcard.json`. Implementation correctness and report findings will be examined during the demo.

3.4.1 Permuterm Indexes

1. Construct Permuterm indexes for each term in the dictionary.
2. Implement prefix-based search for a wild card query using permuterms. Note time taken per query and minimum, maximum, and average over the 30 wildcard queries. Note the memory consumed by the index.

3.4.2 Tree based Indexes

1. Maintain a forward and a backward tree-based index on all terms.
2. To answer a wildcard query `<na*al>`, execute `<na*>` on the forward index and `<la*>` on the backward index. Return the intersection of the 2 sets of results as the answer. Note time taken per query and minimum, maximum, and average over the 30 wildcard queries. Note the memory consumed by the index.

3.5 Experiment 5: Tolerant Retrieval (Total 5, 2 for report)

(To be benchmarked on `s2/s2_wildcard_boolean.json`. Implementation correctness and report findings will be examined during the demo.)

Add support for tolerant retrieval using the wildcard indexes constructed in Experiment 4. You are free to implement any technique to handle a wild card query with boolean retrieval. Convert a query `genera* e*treme value` to a boolean query `<genera* and e*treme and value>`

4 What should you submit?

Files to be submitted.

1. A zip file with only code, no indexes
2. A report.pdf: Reports written in Latex will get bonus points. Report should document all findings from the experiments.

Checklist for zip file.

1. **main.py**: The file that you updated.
2. Other supporting code files.
3. **requirements.txt**: You may include extra libraries for implementation. To be able to run your submission, all your dependencies should be written to a 'requirements.txt' file and submitted. To create your requirements.txt file, run the following command after you have finished implementation:

```
pip freeze > requirements.txt
```

5 Grading Principles

Demos will be conducted for evaluation. Scores will be awarded based on:

1. Whether or not all files are included in the submission (30% penalty is any of the files are missing or if indexes are included)
2. Whether or not your code runs into error (30% penalty)
3. If your code/report is found to be plagiarized (100% penalty, won't qualify for bonus)
4. If the report was generated using a tool like ChatGPT (100% penalty for report, won't qualify for bonus)
5. Report structure and content (breakup of points is provided)
6. *i*) Bonus (10%) for optimizing the data structures using your own or any method discussed in class, *ii*) bonus (5%) for reports written in latex. **A student can earn a maximum of 30 points including the bonus.**
7. Implementation correctness and viva