CMPUT361 Winter 2024 Assignment 4 – Evaluation of Ranked Retrieval

© Denilson Barbosa

January 2024

Instructions

- All code must be written in Python3 and run in the Computing Science instructional lab machines using **only** the Python standard libraries and/or NLTK.
  - It is **your responsibility** to provide clear instructions for the TAs to execute your code **from the command line**.
- This programming assignment can be completed individually, in pairs, or in groups of three, under the **Consultation** model of collaboration as per the Computing Science Department Course Policies.
- All code and/or data *must be on the GitHub repository created by following the instructions on eClass*, following the folder structure for the assignment and the configuration file for automated testing.
- **DON'T'S**: Do not do any of the following. Violations will incur an automatic grade of zero and/or further sanctions:
  - Forget to upload your GitHub repository on eClass.
  - Forget to commit your latest code to the `main` branch of the repository before the deadline.
  - Share code, test collections, or answers.
  - Ask for code, test collections, or answers from others.
  - Use any non-standard library except for NLTK.

- Upload *binary* files or large text files to your GitHub repository.
- Add, remove, or rename any folder in your repository.
- Modify the signature of any of the functions provided.
- Move the functions provided to other files.
- Add any python files except for the ones required in the specs or another one called `utils.py` where you can write helper functions that you need and use in multiple places.

Learning Objectives

This assignment is intended for you to become familiar with:

1. Comparing stemming and lemmatization under different scoring metrics.

Overview

This assignment builds on your solution for Assignment 3 Winter 2024. You can (and **should**) reuse the code you submitted.

Your repository has the folders below.

```
--- collections  --> 'raw' collection files
 |- code         --> code for building the index and answering quer
 |- processed    --> output of your programs in 'code'
```

Collections

Each collection has exactly three files, as in all previous assignments. Your repository comes with one test collection, called `CISI_simplified` .

Code

You will work with the following starter `code` :

```
build_index.py  --> writes the inverted index
query.py         --> answers queries using the index
preprocessing.py --> tokenization and normalization functions
```

Only one change is required for `build_index.py` , if your submission was correct: it must write a lemmatized inverted index and a stemmed inverted index.

One change is required for `query.py` : it now should take one extra parameter, after the weighting scheme for the document vectors, to specify if lemmatization (`l`) or stemming (`s`) should be used.

Example usage specifying that lemmatization should be used:

```bash
% python3 ./code/query.py CISI_simplified ltn l 10 keyword
```

You must **modify** `preprocessing.py` to that it can do lemmatization or stemming as requested.

Query results

Like before, `query.py` must print a list of pairs `docID:score` in a single line, separated by a single `\t` character, corresponding to the `k` documents with highest score according to the parameters. The list must be sorted by decreasing score. Ties are broken by ID (smaller first). Scores must be rounded to three decimal places.

Evaluation program

You are asked to create a program called `evaluation.py` inside the `code` folder. You are free to design your code as you wish.

`evaluation.py` must take the following command line arguments (in order): collection name, weighting scheme for documents, the text normalization (`l` or `s`), the number of results to be returned for each query (`k`), a number of queries to be tested (`n`), and an evaluation metric (`mrr` or `map`).

The program should pick `n` queries at random from the collection and use the code in `query.py` to execute those queries and collect their answers. After running all queries, it should then compute the metric (MRR or MAP@k).

The output to `STDOUT` should be a single line with **all** of the input parameters (in order) and the numeric value of the MRR or MAP@k.

Here’s how you would calculate the MRR looking at the top-10 results of 100 random queries from `CISI_simplified` using `ltn` weights and lemmatization:

```bash
% python3 ./code/evaluation.py CISI_simplified ltn l 10 100 mrr
```

Evaluation report

You are asked to write a brief report with results of an experiment to find the combination of parameters that produces the best results overall. The report should have the following sections:

- **Introduction**: briefly describe the the different scoring parameters that affect the result of the query.
- **Metrics**: briefly describe each of the evaluation metrics.
- **Methodology**: explain your approach to find the best possible configuration. This should explain how you will use `evaluation.py` in a way that the TAs can reproduce your experiment. *You are free to write an extra Python program for this task if needed*. If you do, explain how to use it in the documentation. Note that your code will be judged according to the criteria described below, but its will count towards your report.
- **Results**: provide tables with the more important results that support your conclusion. Do not just list all results. You are asked to prioritize quality versus quantity.
- **Conclusion**: write a conclusion from your experiment. Try to answer questions like "is there a weighting scheme that is always best regardless of text normalization?" or "is there a normalization that is always better regardless of weighting scheme?" and "are there trends relating `k` and the evaluation metric?". Refer to specific rows in tables to support your conclusion.

Your report **must** be written in markdown, in a file called `REPORT.md` at the root of the repository. Reports in any other format or reports that do not use sections and table markup will be considered in violation of the specification and will not be graded.

Documentation Requirements

Write all documentation in the files specified below. Do not add or modify any other files for documentation purposes. Documentation written in the incorrect file will not be graded.

`README.md` at the root of the repo

Include the following:

- CCID(s) and name(s) of students who worked on the assignment.
- Instructions for using your programs from the command line on an instructional machine in the CS department, including all parameters.
  - TAs and instructors will copy and paste your instructions and test them.
- A list of all sources (e.g., StackOverflow) consulted to answer the assignment.

Grading and rubric

Weights:

- Code: 40%:
  - change(s) to `query.py` : 5%
  - change(s) to `build_index.py` : 5%
  - `evaluation.py` : 40%
- Report: 50%.

Grade	Meaning
100	Excellent – meets or exceeds **all of the criteria**
75	Good – meets most of the criteria
50	Acceptable – meets some of the criteria
25	Minimal – meets at least one of the criteria
0	Missing **OR** submission violates specifications

Grading criteria for code

- Organization: code is organized into meaningful and well-defined functions.
- Correctness: code correctly executes what is asked.
- Clarity: variable and function names are meaningful for the material; comments are clear, short, and informative.
- Efficiency: the time complexity of the code matches the asymptotic cost of the optimal solution described in any of the textbooks.

Grading criteria for the report

- Format: report file must be markdown, organized into sections and tables using proper markdown syntax.
- Organization: the report is organized into meaningful sections.
- The conclusions in the report match the data provided in the tables.
- Writing: the text is free of typos and grammatical errors.

< PREVIOUS

Assignment 3