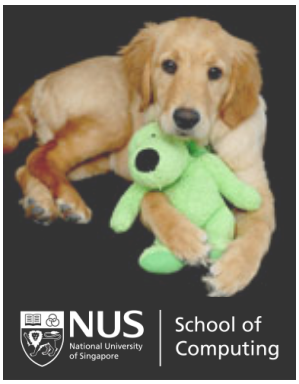

Information Retrieval

NUS SoC, **AY 2019/20**, Semester II, Fridays 12:00-14:00 @ LT15

Last updated: Friday, March 6, 2020 - Information updated for AY 2019/20 Sem II

Deadline for submission: **26 Mar 2020, 10pm SGT**

Homework #3 » Vector Space Model

In Homework 3, you will be implementing ranked retrieval described in Lectures 7 and 8.

Commonalities with Homework #2

The indexing and query commands will use an identical input format to Homework 2, so that you need not modify any of your code to deal with command line processing. To recap:

Indexing: `$ python index.py -i directory-of-documents -d dictionary-file -p postings-file`

Searching: `$ python search.py -d dictionary-file -p postings-file -q file-of-queries -o output-file-of-results`

We will also again use the Reuters training data set provided by NLTK. Depending on where you specified the directory for NLTK data whenyou first installed the NLTK data (recall that the installation is triggered by nltk.download()), this data set is located under a path like: `.../nltk_data/corpora/reuters/training/` .

As in Homework 2, your search process must not keep the postings file in memory in answering queries, but instead use a file cursor to randomly access parts of the postings file. However, there is no such restriction for the indexer (you are not asked to implement BSBI or SPIMI). Also, you should

continue to employ Porter stemming in your submission, both for document indexing as well as query processing.

Vector Space Model

To implement the VSM, you *may* choose to implement (you can can do it differently) your dictionary and postings lists in the following format. The only difference between this format and that in Figure 1.4 in the textbook, is that you encode term frequencies in the postings for the purpose of computing tf×idf. The tuple in each posting represents (doc ID, term freq).

term	doc freq (df)	→	postings lists
ambitious	5	→	(1, 5)→ (7,2) → (21, 7) → ...
...	...		...

In addition to the standard dictionary and postings file, you will need to store information at indexing time about the document length, in order to do document normalization. In the lecture notes and textbook this is referred to as `Length[N]` . You may store this information with the postings, dictionary or as a separate file.

In the searching step, you will need to rank documents by cosine similarity based on *tf×idf*. In terms of SMART notation of *ddd.qqq*, you will need to implement the *lnc.ltc* ranking scheme (i.e., log tf and idf with cosine normalization for **queries** ~~documents~~, and log tf, cosine normalization but no idf for documents. Compute cosine similarity between the query and each document, with the weights follow the tf×idf calculation, where *term freq = 1 + log(tf)* and inverse document frequency *idf = log(N/df)* (for queries). That is,

*tf-idf = (1 + log(tf)) * log(N/df).*

It's suggested that you use log base 10 for your logarithm calculations (i.e., `math.log(x, 10)` , but be careful of cases like `math.log(0, 10)`). The queries we provide are now free text queries, i.e., you don't need to use query operators like AND, OR, NOT and parentheses, and there will be no phrasal queries. These free text queries are similar to those you type in a web search engine's search bar.

Your searcher should output a list of up to 10 most relevant (less if there are fewer than ten documents that have matching stems to the query) docIDs in response to the query. These documents need to be ordered by relevance, with the first document being most relevant. For those with marked with the same relevance, further sort them by the increasing order of the docIDs. Output these documents as a space delimited list as in Homework #2. Be sure not to include extra whitespace before and after any document IDs. For example, if document IDs 2, 1024 and 512 are only 3 relevant documents, in that order your searcher should output these document IDs one one line for the corresponding query:

```
2 1024 512
```

What to turn in?

You are required to submit `index.py`, `search.py`, `dictionary.txt`, and `postings.txt`. Please do not include the Reuters data.

Essay questions:

You are also encouraged to think about the following essay questions as you work on the assignment. These questions are meant to enhance your understanding of the lecture materials and the assignment. You are NOT required to submit your answers but you are welcome to discuss your answers with us. Note that these are open-ended questions and do not have gold standard answers.

1. In this assignment, we didn't ask you to support phrasal queries, which is a feature that is typically supported in web search engines. Describe how you would support phrasal search in conjunction with the VSM model. A sketch of the algorithm is sufficient. (For those of you who like a challenge, please go ahead and implement this feature in your submission but clearly demarcate it in your code and allow this feature to be turned on or off using the command line switch "-x" (where "-x" means to turn on the extended processing of phrasal queries). We will give a small bonus to submissions that achieve this functionality correctly).
2. Describe how your search engine reacts to long documents and long queries as compared to short documents and queries. Is the normalization you use sufficient to address the problems (see Section 6.4.4 for a hint)? In your judgement, is the *ltc.lnc* scheme (n.b., not the ranking scheme you were asked to implement) sufficient for retrieving documents from the Reuters-21578 collection?
3. Do you think zone or field parametric indices would be useful for practical search in the Reuters collection? Note: the Reuters collection does have metadata for each article but the quality of the metadata is not uniform, nor are the metadata classifications uniformly applied (some documents have it, some don't). *Hint: for the next Homework #4*, we **will** be using field metadata, so if you want to base Homework #4 on your Homework #3, you're welcomed to start support of this early (although no extra credit will be given if it's right).

Submission Formatting

You are allowed to do this assignment individually or as a team of two. There will be no difference in grading criteria if you do the assignment as a team or individually. For the submission information below, simply replace any mention of a student number with the two student numbers concatenated with a separating dash (e.g., A000000X-A000001Y).

For us to grade this assignment in a timely manner, we need you to adhere strictly to the following submission guidelines. They will help me grade the assignment in an appropriate manner. You will be penalized if you do not follow these instructions. Your student number in all of the following statements should not have any spaces and any letters should be in CAPITALS. You are to turn in the following files:

- A plain text documentation file `README.txt`: this is a text only file that describes any information you want me to know about your submission. You should give an overview of your system and

describe the important algorithms/steps in your system. However, you should not include any identifiable information about your assignment (your name, phone number, etc.) except your student number and email (we need the email to contact you about your grade, please use your `[u|a|g|e]*******@u.nus.edu` address, not your email alias). This is to help you get an objective grade in your assignment, as we won't associate student numbers with student names.
- All source code. We will be reading your code, so please do us a favor and format it nicely.
- (Optional) A plain text file `ESSAY.txt` that contains your answers to the essay questions (if you choose to attempt the questions). Again, do not disclose any identifiable information in your essay answers.

These files will need to be suitably zipped in a single file called `<student number>.zip`. Please use a zip archive and not tar.gz, bzip, rar or cab files. Make sure when the archive unzips that all of the necessary files are found in a directory called <student number>.

A package of skeleton files have been released in LumiNUS to help you prepare the submission. You may use the homework checker script in the same package to check whether your zip archive fulfills the criteria above. Upload the resulting zip file to the LumiNUS by the due date: 26 Mar 2020, 10pm SGT. There will absolutely be no extensions to the deadline of this assignment. Read the late policy if you're not sure about grade penalties for lateness.

Grading Guidelines

The grading criteria for the assignment is tentatively:

- 50% Correctness of your code
- 20% Documentation
  - 5% For following the submission instructions and formatting your documentation accordingly.
  - 5% For code level documentation.
  - 10% For your high level documentation, in your README document.
- 30% Evaluation: we will test the accuracy and querying speed of your searching program
- 0% Essay questions
- Note: Nominal bonus marks (+1) might be given to submissions that excel in the above-mentioned components.

Disclaimer: percentage weights may vary without prior notice.

Hints

- Indexing a large number of documents may take a while. We suggest that you just test and develop your system on a subset that will make it quick to do experimentation. For example just use the first 10 or 100 documents in your development and debugging. Also, since indexing all 7,000+ documents may take a while, please ensure you save enough time to actually do the queries.

- In the same as spirit as above, make sure you have a working solution to one part before modifying it to do another part. Save your incremental, working progress in another (directory/file/source control system) before starting the next part. In particular, if you work with just a few documents first, you may want to do the tasks in this order:
    1. In-memory indexing of a few documents, testing your query handling portion.
    2. Implementing the stemming, tokenization methods described.
    3. Implementing postings disk-based indexing. Here you can separate the search and indexing parts into two files as suggested.
- Working in a group can be helpful but can be counter-productive too. I have observed that group work tends to make homework submissions grades slightly higher than single submissions but also more average (it's harder to have an outstanding grade with a group). If you think you work better in a group of two, please do. Make sure to clearly partition the work and demarcate the API between your modules.
- You'll need to use the python (lower-)level file input/output commands, `seek()`, `tell()` and `read()`. Another Python module to look at is `linecache`. Please look through the documentation or web pages for these.
- How do you check the correctness of your results? Use your peers! Suggest a few queries that you ran your system on, and post them and the results that you get to the LumiNUS forum (make sure you use the right topic heading).
- While you don't need to print scores out for the official runs that your system will be graded on, you may find it useful to include such information in your debugging output when developing your solution.
- Be aware of any differences in data type sizes on sunfire vs. your development machine. To ensure the portability of your code, check and record the size of the data types in a variable and seek/read accordingly. We will not be responsible if your code works fine on your development machine, but not on sunfire.
- Similarly, bulletproof your input and output. Make sure directories (e.g., arguments to `-i`) are correctly interpreted (add trailing slash if needed). Check that your output is in the correct format (docIDs separated by single spaces, no quotations, no tabs).
- The Reuters corpus has some particularities (as does practically all corpora). There are some character escapes that occur in the corpus (`&lt;` for `<`) but you can safely ignore them; you do not have to process these in any special way for this assignment.
- A consequence of only needing the top 10 documents is that you can speed up search by using a max-heap data structure to process the document list. You could set the max-heap to contain only 100 documents (fewer, if you'd like) and at the end of processing, just report the max 10. Note that the max-heap approach speeds ranking relatively more in document collections that are significantly larger than Reuters-21578 (our collection is a bit too small to have this give much efficiency gain. To think about: why?)

Designed with Twitter Bootstrap.

Back to top