MIE 1513 Decision Support Systems
Assignment: Information Retrieval (IR)

This assignment allows students to apply Information Retrieval (IR) knowledge in practice with popular Python toolkits such as NLTK and Whoosh. This assignment also contains a competitive search engine optimization component to encourage students to explore the IR area.

- Programming language: Python (Google Colab Environment)
- Due Date: Posted in Syllabus

Marking scheme and requirements: Full marks will be given for (1) working, readable, reasonably efficient, documented code that achieves the assignment goals, (2) for providing appropriate answers to the questions in a Jupyter notebook (named `ir_assignment.ipynb`) committed to the student's assignment repository, and (3) attendance at the lab session, running your solution notebook for instructors, and providing clear and succinct answers in response to instructor questions regarding your solution.

Please note the plagiarism policy in the syllabus. If you borrow or modify any *multiline* snippets of code from the web, you are required to cite the URL in a comment above the code that you used. You do not need to cite tutorials or reference content that demonstrate how to use packages – you should certainly be making use of such content.

What/how to submit your work:

1. All your code should be included in a notebook named `ir_assignment.ipynb` that is provided in the cloned assignment repository.
2. Commit and push your work to your GitHub repository in order to submit it. Your last commit and push before the assignment deadline will be considered to be your submission. You can check your repository online to make sure that all required files have actually been committed and pushed to your repository.
3. A link to create a personal repository for this assignment is posted on Quercus.

Credit: This assignment derives from one originally prepared by Paul Thomas (CSIRO, Australia) for the Australian National University version of this course.

Notes that you should pay attention to

1. The same auto-grade restrictions apply for this assignment as they did for the Python warmup assignment. Please check the pinned posts about autograder on Piazza for more information.

2. During the code review, we will ask questions about your free text answers and your results in general as well as some details about your code and possible alternative approaches you might have taken. If you made both an on-time and late submission, you will have to tell us which version we should use for code review with a 30% deduction applying to the code review if you wish us to use your late submission. Note: you will receive a zero for the code review if you did not make an on-time or late submission – in this case you are considered to not have submitted your assignment.

3. Please do NOT use global variables in your functions... they are stripped out by the auto-grader. You can **only** use variables passed into functions or defined within classes.

4. The auto-grader will extract and use the following variables, DON'T change the their names:

   - self.topic_file
   - self.qrels_file
   - self.document_dir
   - self.file_list
   - self.index_sys
   - self.query_parser
   - self.searcher

5. DON'T change the names of functions and class

6. DON'T change the py_trec_eval function

7. DON'T change the class names including CustomFilter, IRSystem, IRQ2, IRQ3, IRQ4

8. DON'T change the CustomFilter class and DON'T create any new custom filter class that is used to define Whoosh schema.

This assignment has *7 points* in total and the point allocation is shown below:

- Auto-grading points (5 points):
  - Q2: 2 points
  - Q3: 1 point
  - Q4: 2 points
- Code review (2 points)

Points will be deducted from the auto-grading points for missing or incomplete text answers.

1 Before the Introductory lab

In the lab, you'll familiarize yourself with three pieces of software: Whoosh, a pure-Python search engineering library, NLTK, a natural language processing toolkit and pytrec_eval, an Information Retrieval evaluation tool for Python, based on the popular trec_eval, the standard software for evaluating search engines with test collections.

The Whoosh documentation can be found on its website at `https://whoosh.readthedocs.io/en/latest/index.html`

The NLTK documentation can be found on its website at `http://www.nltk.org/`

You should have an idea before coming to the lab that we are using Whoosh and NLTK libraries for IR tasks and py_trec_eval to evaluate the performance of IR.

2 In the Introductory lab

Please use the link to create the lab repository, and clone it and open the Jupyter Notebook in the Google Colab.

Start executing statements and follow the instructions in the lab. You will work with a dataset that contains example data for this lab. The example data we will be using is in the "lab-data" folder. This is a very small data set that contains three things:

- A set of documents (email messages), in the documents directory.
- A set of queries – also called "topics" – the file "air.topics".
- A set of judgments, saying which documents are relevant for each topic – the file "air.qrels".

The overall goal here is to search the documents for each query and produce some output. The output can then be compared with the judgments to say how good (or bad) Whoosh is for these tasks. In order to do so:

1. Whoosh needs to make an index of the set of documents, then
2. Whoosh needs to read queries (topics) from the set given, and
3. Whoosh needs to produce output for each query (topic). Then,
4. pytrec_eval can compare Whoosh's output with the judgments and say how good (or bad) the output is.

Once you understand how Whoosh works from running and understanding the provided code:

1. You'll need to index the correct document set.
2. Ideally, you'll need to read topics from a file (air.topics) instead of having them in the program directly.
3. Produce output in the format py_trec_eval expects. This is:

```
01 Q0 email09 0 1.23 myname
01 Q0 email06 1 1.08 myname
```

where “01” is the topic number (01 to 06); “Q0” is the literal string Q0, that is exactly those two characters[1]; “email*n*” is the name of the file you’re returning; “0” (or ”1” or some other number) is the rank of this result;“1.23” (or “1.08” or some other number) is the score of this result; and “myname” is some name for your software (it doesn’t matter what, just be consistent).

Once you’ve done this, index and rank the documents for any or all of the six test queries (e.g., using a default Whoosh index and ranking) and run *py_trec_eval* with the qrels file provided in *air.qrels*.

If py_trec_eval runs correctly and produces numbers which you think are sensible, you’re done with this part. You might want to look at the output, though, and get some understanding of what it means; later you will be asked to interpret this and to choose evaluation measures you prefer.

This is an example of what py_trec_eval may return

```
P_5       01      0.2000
gm_map    all     0.0141
```

where the first column, e.g. ”P_5” is the name of one of the measures. The second column e.g. ”01” is the id for the query/topic. If the value of this column is ”all”, this row is the average of the measure overall queries/topics. The last column is the score for this measure

3 Main Assignment (Assessed in Evaluation Lab)

In the Introductory lab section, you were asked to get Whoosh to index the documents from a very small test collection, run a few queries (“topics”) and produce output in the format expected by *py_trec_eval*. You were also asked to run *py_trec_eval*, to compare your output to the human relevance judgments (“qrels”), and to check you understand the output.

In this part of the assignment, you will run tests with a bigger collection of documents, and more queries. You will also need to improve on the baseline Whoosh configuration.

The data you need should be available in the government directory of the assignment repository. The test collection is about 4,000 documents from US Government web sites and the topics are 15 needs for government information. Both were part of the TREC conference in 2003.

Make sure you provide all the answers in the provided notebook:

- Text answers should be written in the dedicated markdown cells for each question.
- Code answers should be written in the dedicated code cells. Make sure you fill in the values of the result variables as instructed in each question.
- Make sure you perform a basic validation of your code as explained below.

[1] Once upon a time, this field meant something to *pytrec_eval*. It is not used for anything now but it’s still required.

Code Preparation

(a) Put all your imports, and path constants in the dedicated cell in the notebook

(b) Make sure all your paths are relative to `DATA_DIR` and do NOT hard-code absolute paths in your code.

Q1. Appropriate TREC measures

pytrec_eval can report dozens of measures: for example "p5" (precision, in the first five documents returned), "num_rel_ret" (the number of relevant documents retrieved overall queries), and "recip_rank" (the reciprocal rank of top relevant document: e.g., 0.25 if the first relevant document is the fourth in the ranking). You can get the key description of some key measures of py_trec_eval in this link:
`http://www.rafaelglater.com/en/post/learn-how-to-use-trec_eval-to-evaluate-your-information-retrieval-system`

(a) Which of py_trec_eval's measures might be appropriate for measuring search system performance for government web sites? [List the measure]

(b) Why do you think this measure is appropriate? [1 sentence]

Q2. Indexing and querying

Index the government documents, run the queries ("topics") through (vanilla) Whoosh as a baseline system, and run pytrec_eval to compare Whoosh's results with human judgments.

(a) Fill the *create_index*(), *add_files*(), *create_parser_searcher*() and *perform_search*() in class *IRQ*2 according to the input and output requirements of each functions. Don't change the variable name or function that already exist in the class.

(b) How well did the baseline Whoosh system do on your chosen measure? [Provide the number.]

(c) Are there any particular topics where it did very well, or very badly? [If so, list a few topic IDs for each]

(Note: *py_trec_eval*() will report measures for each query/topic separately as well as the averages. This will help you pinpoint good or bad cases.)

Q3. Improving performance

Look at where the baseline Whoosh system did well, or badly.

(a) What changes on the `analyzer (tokenizer, filters, stemmers)` do you think would improve Whoosh's performance on this test collection, and why? [List your findings, proposed solutions and reasoning]

- For the system you aim to improve, you need to (1) understand what documents were highly ranked, (2) what documents should have been highly ranked, and (3) explain false positives (irrelevant documents ranked highly) and false negatives (relevant documents not ranked highly) in order to directly inform your suggested improvements. Hence, please find one query and explain one false positive and one false negative case and explain each error and how this motivates your suggested modification. (Please note: it is highly unlikely for two students to choose the same query and same two false positive and false negative examples. Similarity in responses will be reported to the department for investigation as a possible plagiarism case.)

- Use *print_rel_name*() to show the ground true relevant files and the files your system return. Based on the result, you can open the false positive files and false negative files to do your analysis mentioned above

(b) Fill the *create_index*(), *add_files*(), *create_parser_searcher*() and *perform_search*() in class *IRQ*3 according to the input and output requirements of each functions. Don't change the variable name or function that already exist in the class.

(c) What modifications did you make and what were the improvements? Explain whether there were overall improvements (over some/all queries) in performance and also whether either the false negative or false positive cases from part (a) improved. [1-3 sentences, any single improvement over the baseline is sufficient for full credit, but nonetheless, you are encouraged to explore]

(d) Did your changes improve things overall? [yes/no]

(e) Did some queries get better while others got worse? [yes/no]

(f) What do you think this means for your idea: was it good? Why or why not? [1-3 sentences]

Q4. Search engine optimization

So far, we have explored different basic modules in Information Retrieval. Now, try alternative techniques (tokenizer, filters, stemmers, and scoring functions, or other techniques not covered in the lab.) to improve performance[2]. You should show multiple cells corresponding to multiple iterations of your improvement attempts. At the end, we want a clear markdown cell stating the following:

(a) What problems have you found lead to a low-performance model? Why did that happen?

(b) List all the modifications you have made, also provide with code in *IRQ*4 and reasoning for the changes.

(c) Is there a improvement of performance? How much the improvement is compared with Q3?

Code Validation

(a) Run the validation cells at the end of the notebook to make sure the variables are properly defined and their type is as required by the auto-grader.

(b) Make sure you address any `AssertionError`. The submitted notebook should not have any `AssertionError`.

(c) Make sure your notebook runs without generating exceptions, by restarting it and running all code cells. This can be done by Choosing Runtime −> Restart and Run all. You should see no exceptions.

[2] If you need to use external libraries that are not provided in lab, please ask for approval on piazza.