Wright State University
Department of Computer Science and Engineering

CS7800 Spring 2023

T. K. Prasad

Assignment 1 (Due: February 22) (10 pts)

The primary purpose of this assignment is to get familiar with the basic term generation, indexing, and query processing algorithms and use them to implement a simple information retrieval system. In this project, you will design and implement your own information retrieval system in **Python** using **scikit-learn** APIs. The project has two phases. In Phase I, you will build the indexing and search application. In Phase II, you will evaluate this search engine.

You will build your search engine in **Python** and test on the Cranfield Dataset. You can work individually or in a team of not more than three-persons. In the Cranfield Dataset, there are 225 queries (search terms), 1400 documents, and 1836 (evaluations) from 225 selected queries. When extracting the data from this dataset, make sure to only use the **body** (".W") for both documents (cran.all) and queries (query.txt). Documents and queries, in the Cranfield Dataset, are broken up into different distinct parts/ tags (.T)=tittle, (.A)= author, (.W)=body, (.I)= ID, Etc. A description of the corpus follows:

- **cran.all**: The Cranfield collection corpus file with many fields.
  - (.W) is the only field/tag that you can you use in this assignment.
- **query.text:** A set of 225 queries that can be used to test the retrieval performance of your system.
  - (.W) is the only field/tag that you can you use in this assignment.
- **qrels.text:** Query relevance judgements for each query. Each query is accompanied with a set of documents that are relevant in separate lines.
- **README.txt**: Fine grained description of each column in the dataset files

Phase I: Indexer and Query Processor

Scikit-learn is a free software machine learning library for the Python programming language. You will use the scikit-learn API to index the documents in Cranfield dataset to answer some standard queries. Here we describe possible modules you need to complete in Phase I.

Indexer:

The indexer will process all the documents in the corpus to create a searchable index/matrix. Note that documents in the Cranfield collection are contained in a single file and should be separated into multiple files before indexing. Therefore, you will need to prepare a corpus of documents (**training set**) and a set of queries (**test data)** for processing. If you are unfamiliar with the differences between fit(), transform(), and fit_transform(), then I would suggest you check out this article, read through the documentation, and do your own research. As choosing the wrong function for indexing or query processing can change your results, think carefully.

Before indexing, you'll need to perform text preprocessing on corpus of documents and the individual queries.

Text preprocessing is the first crucial step in a Natural Language Processing (NLP) system, with major ramifications on the final performance of the system. Therefore, to ensure efficiency and effectiveness of your system, you will need to use industry-standard text processing techniques, particularly tokenizing, case-folding and stop-word removal in this assignment. Do not deviate and use other text preprocessing procedures as your results will vary and you may lose points.

- **Tokenization** is the task of chopping a character sequence (text document) into meaningful units, called tokens, eliminating certain characters that may be irrelevant, such as punctuations.
- **Case-Folding** is a common strategy when normalizing a text document by reducing all letters (terms) to lowercase. This strategy allows us to match instances of *Fire* at the beginning of a text message to instances of *fire* appearing elsewhere. This technique is extremely important in the Cranfield Dataset because there is not always consistent usage of proper capitalization throughout the data set and we may be interested in just the root of the word to make sense of it.
- **Stop-Word Removal** is the process of dropping or removing common words. This process is important for implementing both Boolean and vector-space model as leaving the stop-words in will not give good results. Moreover, by removing stop-words, we are improving the performance of the information retrieval system. Make sure to use `sklearn.feature_extraction.text.ENGLISH_STOP_WORDS` for your list of stopwords.

The Querying process must undergo a similar treatment as document during indexing (so remember to use the same analyzers / text preprocessing pipeline for both indexing documents and processing queries). These steps are shared by both indexing and query processing.

For this assignment, you will use two different vectorizers in scikit-learn to create a vector space model.

- TFIDF vectorizer -- calculates TFIDF score for each document or query
  - Only Use `stop_words & lowercase`
  - tokenization happens by default
- Binary vectorizer -- 1 if term is present in document/query, 0 otherwise
  - Only Use `stop_words & lowercas`
  - tokenization happens by default

The Querying processor:
You will take a user query as input and apply the same sequence of pre-processing operations that was applied to the documents during indexing. Then you will need to calculate the similarity between queries and documents for both vector space model (TF-IDF and Binary) using two different similarity measures. The similarity measures you will need to use are cosine similarity a and Euclidean distance. After calculating the similarity, each element in the result matrix

represents similarity between one query and one document. Note: do not normalize document vector or query vectors to a unit length.

The cosine similarity can be calculated for a document and a query represented by their TF-IDF vectors d_i and q_j as follows:

$$\cos(d_i, d_j) = \frac{d_i \bullet q_j}{\|d_i\| \|q_j\|} = \frac{\sum_{k=1}^{n} d_{i_k} q_{j_k}}{\sqrt{\sum_{k=1}^{n} d_{i_k}^2} \sqrt{\sum_{k=1}^{n} q_{j_k}^2}}$$

where n is the total number of terms, and k represents the term being iterated over. The cosine similarity is 0 for orthogonal vectors, and 1 for identical vectors. Again, we are not asking you to manually code any of the two-similarity equations.

Phase II: Evaluation

- Get the indices (indexes) of the 10 most relevant documents for each query (225) for both vector space models (TFIDF and Binary) and both distance measures (cosine similarity and Euclidean distance). You should end up with two lists consisting of 225 queries where each query item contains 10 most relevant documents in each query. Note: utilizing a list of lists, dictionary of lists, or NumPy array, might be a good choice. The final shape of each of the data structures should be (225,10).
- Calculate the precision, recall and F-score from the lists of query relevant documents, retrieved in the previous step, against the list of relevant documents in qrels.text.
- Discuss how precision, recall, and F-score may vary or may not vary between different vector models and between different similarity measures used in this assignment.
- Discuss the effectiveness/ineffectualness of the two similarity measures and the two vector models used at retrieving the correct document.
- Discuss any other relevant insights about this assignment.
- Graphically display the Precision, Recall, and F-scores for the two similarity measures and two vector models using matplotlib.pyplot.bar. You should have a total of 12 graphs. Figure 1 demonstrates exactly how your graph should look.
- Finally, display in the terminal the mean and maximum Precision, Recall, and F-scores for the two similarity measures and the two vector models. Figure 2 demonstrates **exactly** how your terminal output should look. I repeat, do not deviate from this format at all as we will be using a grading script to ensure you got correct results.

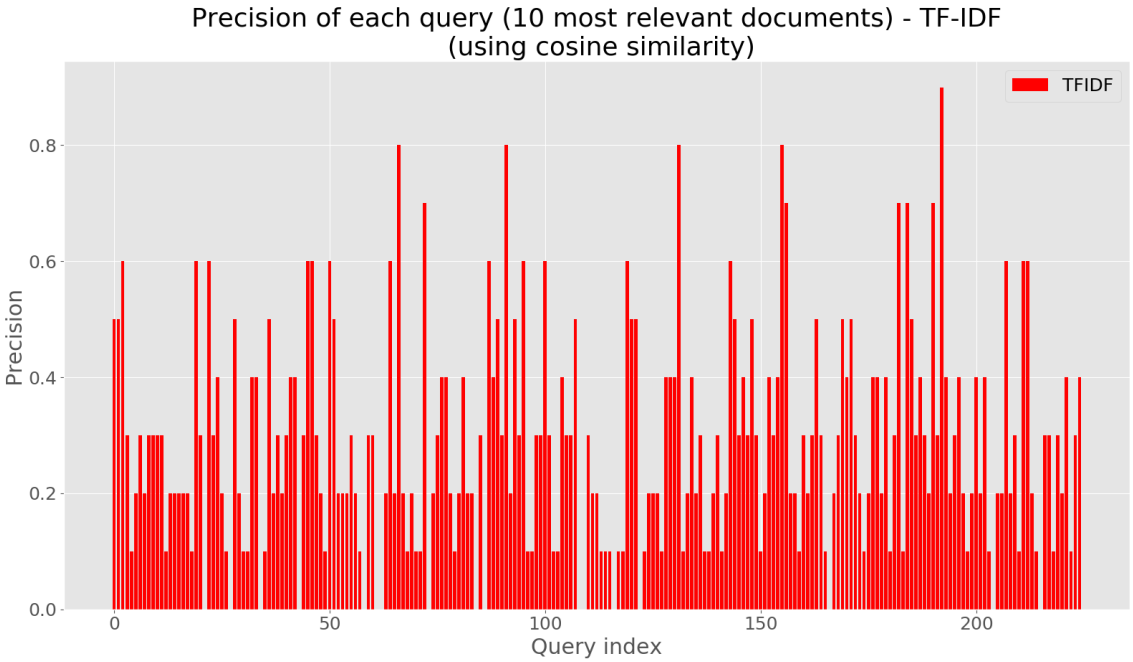

Figure 1: example of how you should graphically display your precision, recall, and f-score

```
{'Binary': {'f': {'cos': (mean, max),
                  'euc': (mean, max)},
            'p': {'cos': (mean, max),
                  'euc': (mean, max)},
            'r': {'cos': (mean, max),
                  'euc': (mean, max)}},
 'TFIDF': {'f': {'cos': (mean, max),
                 'euc': (mean, max)},
           'p': {'cos': (mean, max),
                 'euc': (mean, max)},
           'r': {'cos': (mean, max),
                 'euc': (mean, max)}}}
```

Figure 2: example of your exact format for the terminal output

Setup:

- Download the Cranfield dataset. It contains three files: cran.all - the document collection, query.text - the sample queries, qrels.text - the relevant query-document pairs. Check the README.txt for more details.

- Use Python 3 or higher
    - Do not use Jupiter notebook
- If it helps, you can use NumPy.
    - `import numpy as np`
- Install scikit-learn:
    - See: https://www.activestate.com/resources/quick-reads/how-to-install-scikit-learn/
- Name your well-documented Python script as *assignment1.py*.
- Students are required to use a Python virtual environment. For those unfamiliar with Python virtual environments here are two reference articles explaining how to use and activate:
    - See: https://uoa-eresearch.github.io/eresearch-cookbook/recipe/2014/11/26/python-virtual-env/
    - See: https://realpython.com/lessons/creating-virtual-environment/
- Once you finish this assignment and are ready to submit it to pilot, you need to create a *requirements.txt* file with all the libraries used in your development. Use the following command only after you've activated your Python environment. This is important to make sure that we can run your code on our systems.
    - `source venv/bin/activate`
    - `pip list --format=freeze > requirements.txt`
    - `deactivate`
    - See: https://openclassrooms.com/en/courses/6900846-set-up-a-python-environment/6990546-manage-virtual-environments-using-requirements-files
- Important note: do not use any other nonstandard Python library's.

Debug:

Please test/debug all your programs thoroughly. By designing tests, ask yourself questions like:

- Are the stop-words really removed?
- What is the number of terms in the dictionary? Do they make sense?
- Does your document-term matrix, have the right shape?
    - The shape should be close to (1400, 7180)
- How do you confirm that TF-IDF values are computed correctly?
- Do you also convert queries to terms?
- How do you confirm cosine similarity is computed correctly?
- How do you confirm Euclidean distance is computed correctly?
- Are your selected sample queries getting the same results as you expect for Boolean model processing?
- Are your selected sample queries getting the same results as you expect for vector model processing?

These are just a few questions I would suggest you ask yourself before submitting your assignment. I would expect you to perform more testing than what is suggested above.

Deliverables

TURN IN: Upload one tar archive **file** per team that contains the following files:

- **Code and accompanying documentation***:* Include well-documented source code for the entire project.
- **Evaluation information***:* Discuss, in a PDF file, the evaluation metrics to be used to quantify the quality of the search results, and determine the quality of results (e.g., precision, recall, f-score) for all the top 10 relevant document query results from the Cranfield collection and report the evaluation metrics for the entire test set. This document should also contain the screenshots of your graphs and terminal output. Additionally, all team member names, UIDs, and email addresses must be included in this document.
- **README.txt:** This document should briefly explain your application, how to launch the application, any external libraries used, what version **of Python 3 used**, all team members names and UIDs, and any other relevant information. The more information you provide in this document the easier it is for us to grade andgive partial credit if something does not work on our system.
- **Application execution**: Make sure that your Python program runs from the command-line. We will use the following command to execute your code:
    - *`python3 assignment1.py`*
    - To clarify, your *`assignment1.py`* must be self-contained, be able to run on someone else's computer, and able to be run from the command line with the exact syntax given above.
    - Do not use any absolute path for input files or any data files. All paths should be local and relative to your working directory.
    - We suggest each member of your team participate in the development and separately testing your application on multiple installations / computers to make sure everything works, and you have not hardcoded anything.
    - Any deviations from this will result in a 25% penalty.
    - Do **not use** Jupyter notebook, as your application must launch from the command-line with the above given syntax.
- **Input and output files**: All file(s) must be placed in your working directory, all generated output file(s) must also be placed inyour working directory, and no subdirectories.
- **requirements.txt:** All the libraries used in your virtual Python environment must be stated.
- **TAR archive**: Submit your application using the following command exactly as written to tar up your working directory:

  "*`tar -zcvf assignment1.tgz yourWorkDirectoryName/`*"
- **Upload** the archive *asg1.zip* onto *Pilot $>$ Dropbox $>$ Assignment 1*folder by **February 22**. (Only one submission per team.) Any member of the team

can be required to demo your program to us (if necessary) and be prepared to answer questions about its design, implementation, and comparative evaluation.

- **Environment setup requirements**: Any deviation from what was discussed in this section will result in a 25% reduction for each deviation in your total grade. Therefore, I strongly suggest you make sure you follow this document closely and ask questions on *discord* for clarification.

Grading Criteria

You must obtain a PASS on this assignment to PASS the course. At the minimum, your code should compile, process some queries, and return reasonable results. A passing grade is 60%.

Assignments are designed to help you learn the core concepts and are the primary course "homework". Corrupt files or other computer problems will not be considered a valid excuse to extend the deadline. It is your responsibility to regularly back-up your work. We strongly suggest that you save your work to multiple locations to aid in the recovery of corrupt files. If you have questions regarding the project, we (the GTA, the grader and the instructor) are there to help.

Assignments that are submitted late will incur a penalty of 25% reduction on total grade per day the assignment is late. The project must be turned in on Pilot as described in the project description to receive full credit. Assignments emailed to the GTA, or professor will receive an immediate 25% reduction in total grade because the whole purpose of Pilot is to streamline communication.

Cheating:

Please do not copy other team's work, do not copy other projects found on the Internet or use any outside resource without proper citation. In short, plagiarism will get you a zero on this assignment and potentially an F grade in the class. We are not obsessed with looking for cheating, but if we see something suspicious, we will investigate and then refer it to the Office of Judicial Affairs. This is more work for us and is embarrassing for everyone. Again, please don't; this has been a problem in the past. If the rules are unclear or you are unsure of how they apply, ask the instructor, GTA, or grader beforehand. The academic integrity policy as available online.

No deadline extension will be given.