# Wright State University

Department of Computer Science and Engineering

CS7800 Summer 2024

T. K. Prasad

**Assignment 1 (Due: June 19) (10 pts)**

## Objective

This assignment provides an exciting opportunity to delve into the realm of information retrieval system (IRS). The primary objective is to introduce you to key concepts such as the API, term generation, indexing, and query processing algorithms. You will apply these concepts to develop, design and implement a basic IRS in Python using *scikit-learn APIs*. The project is structured into two distinct phases:

- Phase I: Building the indexing and search application.
- Phase II: Assessing the performance of your search engine.

You will be tasked with building a search engine in Python and evaluating it using the Cranfield Dataset. You may work individually or in teams of up to three members. The Cranfield Dataset includes 225 queries (search terms), 1,400 documents, and 1,836 evaluations based on these queries. For this project, ensure you use only the body (.W) sections for both documents (cran.all) and queries (query.txt). The Cranfield Dataset organizes documents and queries into several distinct parts delimited by tags, such as (.T) for title, (.A) for author, (.W) for body, and (.I) for ID.

Here is a brief description of the dataset components:

- **cran.all:** The Cranfield collection corpus file containing various fields, with only the (.W) field/tag being usable in this assignment.
- **query.txt:** A collection of 225 queries that can be utilized to assess the retrieval performance of your system, with the (.W) field/tag being the relevant one for this assignment.
- **qrels.txt:** Query relevance judgments for each query, with each query accompanied by a set of relevant documents for it listed one per line.
- **README.txt**: A detailed explanation of each column in the dataset files is provided in this file.

### Phase I: Indexer and Query Processor

In Phase I, we will leverage the scikit-learn library, a powerful open-source machine learning tool for Python, to create an efficient Indexer and Query Processor. Our primary objective is to process the Cranfield Dataset and provide answers to standard queries. Below, we outline the key modules required for Phase I:

## Vectorization and Indexer Module:

The Indexer module is integral to transforming the Cranfield dataset's documents into a searchable index/matrix. Considering the Cranfield collection's unique format — storing documents in a single file — it is imperative to first separate these documents into multiple files for effective indexing. This process is facilitated by employing vectorization techniques using scikit-learn. The following steps outline the critical components of the Indexer module:

1. **Data Preparation:** Your initial task is to prepare two sets: a corpus of documents (the training set) and a set of queries (the test data). To accomplish this, you will leverage two distinct vector space models by employing different vectorization techniques within the scikit-learn framework.
    - **NOTE:** Understanding the distinction between `fit()`, `transform()`, and `fit_transform()` is crucial as the appropriate application of these functions is vital for accurate indexing and query processing. It is recommended that you explore additional resources, including documentation, to gain a comprehensive understanding before proceeding the assignment.
2. **Text Preprocessing:** Text preprocessing is a fundamental step in Natural Language Processing (NLP) systems. To ensure the efficiency and effectiveness of your system, adhere to industry-standard text processing techniques, including:
    - **Tokenization:** Divide a character sequence (text document) into meaningful units called tokens, while removing irrelevant characters like punctuation.
    - **Case-Folding:** Normalize text documents by converting all letters to lowercase. This approach ensures that instances of words with varying capitalization are treated consistently.
    - **Stop-Word Removal:** Eliminate common words, known as *stop-words*, from the text. It is *required* to use `sklearn.feature_extraction.text.ENGLISH_STOP_WORDS` for your list of stop-words.

In this section of your assignment, you'll face a dual challenge. First, you need to proficiently utilize the default Binary Vectorizer. Second, your task involves replacing the standard TF-IDF calculation in the library with a custom implementation that creatively modifies the traditional TF and IDF calculation. For both vectorizers, it's essential to maintain consistency and accuracy in your data handling and analysis by adhering to text preprocessing standards. The following outline offers precise instructions for your implementation:

**Part 1: Binary Vectorizer**

- **Usage**: Employ the Binary Vectorizer, which assigns '1' if a term is present in a document/query and '0' otherwise.
- **Configuration**: Ensure to enable `stop_words` and `lowercase` options. Tokenization is handled by default.
- **Reference**: Consult the scikit-learn documentation for more details and implementation guidelines.

**Part 2: New Custom Implementation for Term Frequency-Inverse Document Frequency (TF-IDF):**

Normally, the term frequency (TF) is calculated as the raw count of a term in a document divided by the total number of terms in the document. However, we are tasking you with implementing your own custom term frequency that seamlessly integrates with CountVectorizer. Your custom TF-IDF vectorizer should inherit from the appropriate scikit-learn base components to ensure it

does not break the normal behavior of `CountVectorizer`. Your job is to implement this custom TF and IDF within the class, maintaining compatibility with the existing functionalities.

**Part 2.1: TF:**

Our new custom implementation for term frequency will use log normalization with a document length penalty. This approach helps balance the influence of document length and reduces the impact of terms with exceptionally high frequencies, leading to fairer and more effective comparisons between documents. The new definition of term frequency can be expressed as:

$$TF_{modified(t,d)} = \frac{1 + log(count(t,d))}{1 + log(length(d))}$$

where `count(t,d)` is the raw count of term `t` in document `d`, and `length(d)` is the total number of terms in document `d`.

**Reasons for the New Version:**

1. **Mitigates the Impact of Document Length**: This approach helps to normalize term frequency by considering the length of the document. It prevents longer documents from having an unfair advantage simply because they contain more terms. By dividing by the log of document length, the term frequency is scaled down for longer documents, making the comparison between documents of different lengths fairer.
2. **Reduces the Effect of Outliers**: Log normalization dampens the effect of very high term frequencies. Terms that appear very frequently in a document do not disproportionately influence the term frequency score. This helps in reducing the skewness in the data caused by outliers (terms with exceptionally high counts).
3. **Encourages Creative Thinking**: We want students to think creatively and avoid copying from online resources. One reason we've implemented this term frequency approach is to encourage you to code and solve the problem in an original way. In the real world, you will often need to modify how you transform your data into a numerical format to ensure your machine learning algorithms work successfully. This exercise will help you develop those crucial skills.

**Example:** Consider a collection of documents with varying lengths. In a standard term frequency calculation, a term appearing 10 times in a short document would be considered more important than the same term appearing 20 times in a much longer document as the term in the longer document might not be as significant if the document is very lengthy.

- Document A (Short): 10 occurrences of term "machine" in 100 words.

$$TF_{modified(machine,Doc\ A)} = \frac{1 + log(10)}{1 + log(100)} = \frac{2}{3}$$

- Document B (Long): 20 occurrences of term "machine" in 1000 words.

$$TF_{modified(machine,Doc\ B)} = \frac{1 + log(20)}{1 + log(1000)} = \frac{2.3}{4}$$

Even though the term "machine" appears more frequently in Document B, its normalized frequency is lower due to the document's length, ensuring that Document A's relevance is not overshadowed simply due to its shorter length: $TF_{modified(machine,Doc\ A)} > TF_{modified(machine,Doc\ B)}$

**Crucial tip [LAPLACE CORRECTION]:** When implementing the log normalization for term frequency, it's crucial to ensure that the log function is never applied to zero. This can occur if a term frequency count is zero or if the document length is zero. To handle this, you must check to see if either value is zero and, if it is, replace it with one. This way, we will never

have a log of zero; instead, we will have a log of one, which is zero. Ensure this check is done to avoid any mathematical errors during the process.

**Part 2.2 IDF:**

Traditionally, the IDF component uses a logarithmic scale. However, you will implement a variant where the IDF is the reciprocal of the document frequency. This new definition of IDF must be self-contained in the custom TF-IDF vectorizer that interfaces seamlessly with `CountVectorizer`. Your vectorizer should compute the new term frequencies (TF) as discussed above and calculate the Inverse Document Frequency (IDF) as the reciprocal of the document frequency (without using the logarithmic scale).

**Implementation Details:**

1. **Custom IDF Calculation**: You will need to modify the IDF component to be the simple reciprocal of the document frequency. This means for each term, calculate IDF as:

$$IDF_{modified(t,d)} = \frac{1}{DF(t)}$$

   where `DF(t)` is the document frequency, i.e., the number of documents containing the term `t`.
2. **Handling Zero Document Frequency**: If the document frequency of a term is zero, which would lead to an undefined value in the calculation of IDF, set the IDF value of that term to one to avoid division by zero.

**Final Example Usage:** Below is an example of how your final custom TF-IDF transformer should work. You must follow this exact layout to avoid losing points. Given a set of documents, you will instantiate your custom TF-IDF transformer class, apply the `fit_transform` method to the corpus, and generate the correct TF-IDF matrix.

```
# Sample documents
documents = [
    "the quick brown fox jumps over the lazy dog",
    "the quick brown fox",
    "lazy dogs run fast",
    "quick brown foxes leap over lazy dogs in the park"
]

# Apply the custom TF-IDF transformation
custom_tfidf = CustomTFIDFVectorizer()
tfidf_matrix = custom_tfidf.fit_transform(documents)
```

## The Querying processor:

All queries will undergo the same series of preprocessing operations that were previously employed on the documents during the document indexing process. After this preprocessing step, we will assess the similarity between queries and documents using two distinct similarity metrics within the two vector space models (TFIDF and Binary). Note that Binary VSM in the APIs and Boolean Model we discussed in class are the same in that both of them represent data (term-

document relationship) using zeros and ones, indicating the presence or absence of terms. The selected similarity metrics are Cosine Similarity and Euclidean Distance. It is imperative to exclusively employ the following similarity measures for this task:

- sklearn.metrics.pairwise.cosine_similarity
- sklearn.metrics.pairwise_distances.metric='euclidean'

After calculating the similarity, each element in the result matrix represents similarity between one query and one document. Do not normalize document vector or query vectors to a unit length.

**Similarity Descriptions:**

- 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 index being iterated over. The cosine similarity is 0 for orthogonal vectors, and 1 for directionally aligned (includes identical) vectors.

- Euclidean distance is a measure of the "straight-line" distance between two points in n-dimensional space. It is a commonly used metric in various fields, including text analysis, to determine the similarity between two vectors. For a document and a query represented by their TF-IDF vectors $d_i$ and $q_j$, the Euclidean distance can be calculated as follows:

$$Euclidean(d_i, q_j) = \sqrt{\sum_{k=1}^{n} (d_{i,k} - q_{j,k})^2}$$

  where n is the total number of terms, and k represents the term being iterated over. In this formula:

  - $d_{i,k}$ is the TF-IDF weight of term $k$ in document *i.*
  - $q_j, k$ is the TF-IDF weight of term $k$ in query *j.*

  Unlike cosine similarity, which measures the angle between vectors, Euclidean distance provides a direct measure of the absolute difference. The Euclidean distance is not bounded and depends on the magnitude of the differences in term weights. A smaller Euclidean distance indicates greater similarity between the vectors, while a larger distance indicates more dissimilarity.

## 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 (Euclidean distance and Cosine similarity). You should end up with **four** lists, consisting of 225 queries each, where each query item contains the 10 most relevant documents for each model-measure configuration. 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 or limitations of the two similarity measures and the two vector models used in retrieving the relevant documents.
- Discuss any other insights you may have gained that we did not foresee.
- Graphically display the Precision, Recall, and F-scores for the two similarity measures and two vector models using matplotlib.pyplot.bar. Your task is to generate a total of 12 graphs, each saved as `'{metric}_{vectorizer_type}_{distance_type}'.png` files to your working directory. Do not display these graphs while your application is running; they should be saved in the background. 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 not only shows exactly how your terminal output should appear, but it also provides the exact results that you should obtain. Make sure your output matches this exactly as we will be using a grading script that checks for this.

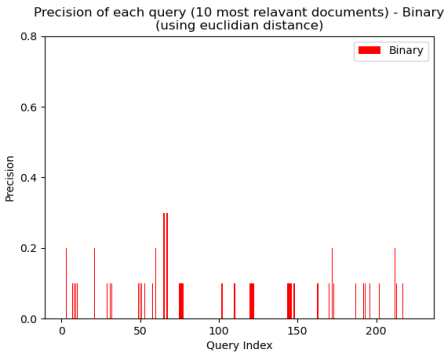


**Figure 1:** Example of graphic display of precision, recall, and F-score.

```
(venv) PS C:\Users\natcr\Desktop\College\CS 7800 Information Retrieva\ProjectsUpdate\a1_2024 - Summer\Assignment1\assignment1_code> .\venv\Scripts\activate
(venv) PS C:\Users\natcr\Desktop\College\CS 7800 Information Retrieva\ProjectsUpdate\a1_2024 - Summer\Assignment1\assignment1_code> python .\assignment1.py
{'Binary': {'f': {'cos': (0.2684196452691116, 0.7058823529411764),
                  'euc': (0.0215344699034236, 0.2666666666666666)},
            'p': {'cos': (0.2435555555555555, 0.7),
                  'euc': (0.01955555555555556, 0.3)},
            'r': {'cos': (0.3533881875277164, 1.0),
                  'euc': (0.02845181978515312, 0.5)}},
 'TFIDF': {'f': {'cos': (0.17126607278736708, 0.7368421052631577),
                 'euc': (0.01840728976034244, 0.32)},
           'p': {'cos': (0.15244444444444444, 0.7),
                 'euc': (0.017777777777777778, 0.4)},
           'r': {'cos': (0.2356335453009705, 1.0),
                 'euc': (0.02274170845017078, 0.3333333333333333)}}}
```

**Figure 2:** example of your exact format and exact final results for the terminal output

## Setup:

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

- Use Python 3 or higher
  - Do not use Jupyter notebook
- You may need 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.
  - `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 libraries.

## 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?
- 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 the Binary Vectorizer?
- 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 solution. I would expect you to perform more rigorous testing than what is suggested above.

## Deliverables

**TURN IN**: Upload one tar archive file or Zip 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**: Ensure that your Python program is executable from the command-line. To execute your code, use the following command:
    - *python3 assignment1.py*
    - It is essential that your assignment1.py is self-contained, capable of running on any computer, and can be executed from the command line using the exact syntax provided above.
    - Your application should automatically save the 12 graphs generated to the working directory and should not display them during execution.
    - Avoid using absolute paths for input files or any data files; all file paths should be relative to your working directory.
    - We recommend that each team member actively participates in the development process and separately tests the application on multiple installations or computers to ensure compatibility and to avoid hardcoding any specific configurations.
    - Please note that using Jupyter Notebook is not allowed, as your application must launch from the command-line using the specified syntax.
    - Any deviations from these guidelines will result in a 30% penalty.
- **Input and output files**: All file(s) must be placed in your working directory, all generated output file(s) must also be placed in your working directory, and no subdirectories.
- **requirements.txt:** All the libraries used in your virtual Python environment must be stated.
- **Uploading to Pilot:** To submit your assignment, follow these steps:
    1. Before compressing your assignment, make sure to remove the environment (venv) folder. Major points will be deducted if you do not remove this folder.
    2. Compress your assignment into an archive file named "assignment1.zip". Only archive your files in a zip format.

3. Upload the archive file to Pilot > Dropbox > Assignment 1 folder no later than June 21st.
4. Ensure that your team submits only one file and includes the names and UIDs of all team members in the Dropbox.

If required, be prepared to demo your program to the instructors and answer questions related to its design, implementation, and comparative evaluation.

- **Environment setup requirements**: Any deviation from what was discussed in this section will result in a 30% reduction for each deviation in your total grade. Therefore, we strongly suggest you make sure you follow this document closely and ask questions on ***discord*** *channel for the course CS7800* for clarification.
- **Caution:** Points will be deducted if you do not follow the naming conventions and use standard file extensions, or if your program does not run in the terminal/command line. **Use/monitor *discord* to seek clarifications and for updates.** If there are several small changes due to these discussions, we may even email an updated assignment consolidating all the changes.
- Below, we provide an illustration of your folder structure in its initial state, showing all the existing files.

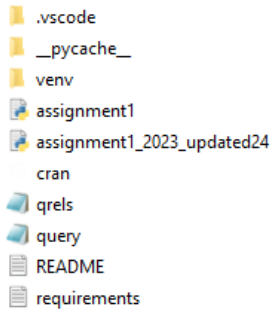


**Figure 3:** folder structure initial state.

## Grading Criteria

At the minimum, your code should compile, process the dataset, and return reasonable results.

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.

**Due to generous deadline, late submission will not be accepted. So, plan to submit a version early and then revise as needed as we will grade only the last version and will ignore earlier versions.** 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 and scan your submission for plagiarism.

**Cheating:**

Please do not copy other team's work, do not copy other projects found on the Internet, do not blindly use AI assistance (ChatGPT/Copilot), 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.

*Full disclosure*: Plagiarism detection tool was deployed last semester to detect violations, offenders penalized, and reported to the administration. A partial original solution is better than borrowed complete solution for your ultimate grade. All the team members are equally responsible for the turned-in work.

No deadline extension will be given.