SI 650 / EECS 549 F24 Homework 1
Text, Indexing, Ranking, and Evaluation

Due: See Canvas for official deadline

1 Homework Overview

This homework will have you developing a full search engine using vector space model techniques. You will go all the way from the document pre-processing to being able to interact and search with a UI. To accomplish this, you will implement four core pieces of an Information Retrieval system:

1. **Document Preprocessing**: First, you will transform each document in the collection into a list of concepts (e.g., words, phrases)
2. **Indexing**: Second, you will transform the processed documents into an inverted index data structure for easy retrieval
3. **Ranking**: Third, you will implement a ranking function that uses the inverted index to score how relevant is a document with respect to a query
4. **Evaluation**: Fourth, you will implement evaluation functions to measure how different ranking functions perform

To implement these four pieces, we have provided specific python files with methods that need to be implemented and descriptions of what is to be done. This document serves as high-level guidance on the tasks to be done and additional specifics of the implementations are provided in the code itself, which is probably where you'll be looking when you actually go to implement things. This document contains the list of tasks you'll need to complete and the point values of completing each task.

The homework is designed to be modular so that each piece can be run and tested separately. For example, you can run the document pre-processor by itself on an input document and verify that the contents have been correctly tokenized and extracted. Each of the four pieces uses files as input and output so you can inspect and analyze them. As a data scientist, testing and verifying your code is critical. We have provided very simple toy data to help get you started but these don't check for all the different tasks or cases you'll want.

There are two types of evaluations for this homework. First, this homework requires you to submit your code to an autograder that will test your implementations for each piece of the homework for correctness. You will be scored on a private set of test cases. Please utilize the public tests we provide in the starter code and design your own tests to ensure the robustness of your system. This accounts for 65 points of this homework. Second, you will submit answers to questions in the homework that will require you to think and write.

These are more conceptual questions, but you are required to do some basic plotting to gain some intuition and show your reasoning. These questions are aimed at helping you think more broadly about the different design trade-offs in implementing an IR system and what kinds of practical choices you might need to make.

This assignment has the following learning goals:

1. Develop an intuitive sense of what is "relevance" through making your own judgments
2. Understand how to do simple text operations like tokenizing with various packages.
3. Gain awareness of the performance costs of different implementations
4. Become familiar with the vector space model and how to implement ranking functions using an inverted index
5. Understand how IR systems are evaluated with MAP and NDCG and what impact ranking position has on a model's quality.
6. Gain new software development and debugging skills
7. Learn how to use unit tests
8. Build intuition on how IR systems are designed and how the different pieces fit together
9. Improve the ability to read software library documentation

2 Important Notes

Things to keep in mind when working on this assignment:

1. Please start early.
2. Read the whole instructions before getting started to get a sense of what all is involved
3. This document has higher-level instructions. The starter code we provide will also include more detailed information
4. You do not need to implement everything from each step before moving on. You can work in parallel; e.g., once you get a simple tokenizer running, you can begin working on an indexer.
5. Once you have at least one part of the first three parts running, you can launch the search engine locally and see how it performs. We've included a separate document on Canvas that details these steps. This can be a useful tool in debugging. Note: you can even launch the search engine without having implemented everything and it will still show results!
6. Debugging is essential for this assignment. You will need to verify that your code works at each step. We have included some "toy" inputs that you can start with but try to get a sense of what the inputs and outputs are for each method/class and test that your implementation is doing what you expect
7. Not all steps are spelled out in exact detail. This is by design. The assignment is intended to help you improve your software development skills—especially those that go from higher-level instructions to code, which is what you'll encounter in the workplace.
8. When working on the relevance labeling part (Part 5), we encourage you to break up this task into smaller time segments, e.g., label for 10 minutes then take a break.

The fresher you are, the better labels you will provide. You and your classmates will appreciate having good relevance scores to use in future assignments.

3 Document Preprocessing

The first step will be to turn the raw document strings into a sequence of terms. This process is often called *tokenization*. However, defining what exactly is a "token" is non-trivial. For example, in the sentence "I didn't get tickets to Taylor Swift, right?", what is a token? If we separate just by whitespace (spaces, newlines, tabs), we have tokens like "right?" where the punctuation is included. Perhaps we might also want to have "Taylor Swift" as a single term since those two tokens refer to a specific person. These options reflect different choices that will have trade-offs in terms of performance and tokenized-document quality. Unfortunately, there is no right choice. As an IR practitioner, you will need to examine your data and make an informed choice—possibly with respect to performance metrics—for how to preprocess your data. This part of the assignment will have you implement a few different options and compare the performance.

Your core implementation tasks are found in `document_preprocessor.py` in the homework code. In practice, you will implement three different tokenizers, summarized as follows:

1. Implement a document preprocessor that uses the `split` function to tokenize
2. Implement a document preprocessor that uses NLTK's RegexTokenizer. You should use the default regular expression in your code submission.
3. Implement document preprocessor that uses SpaCy to tokenize and extract Named Entities (people, places, etc.) as single terms

■ ~~**Problem 1.**~~ For each of the tokenizers, add support for recognizing specific multi-word expressions as single tokens from a list provided in the constructor. For example, if the list contains the phrase "Taylor Swift", if those two tokens appear in sequence, they should be merged into a single term. Your approach should match the longest sequence so that if both "Taylor Swift" and "Taylor Swift Eras Tour" appear in the list, the latter would be matched if it appears instead of just initially matching "Taylor Swift".

■ **Problem 2.** (5 points) Using all three document preprocessors, tokenize the first 1000 documents and record the time. Make a plot showing the time taken for each. Using the average speed per document, estimate how much time it would take to preprocess the entire corpus and write the estimates in a sentence you submit. Describe the trade-offs you observe in the tokenizers' accuracies and speed and which tokenizer you would use in practice.

4 Indexing

The second piece will have you turning the collection of documents into an inverted index data structure that can be used to efficiently retrieve relevant documents. Your task will be to implement two variations on an inverted index. These variations all share the same core structure, and much of the functionality will be nearly identical, but the two versions will give you a sense of what is being represented. To summarize (more details in code), you will implement:

1. Implement an inverted index that is all in memory
2. Implement an in-memory inverted index with positional information

Once you get these implemented, your tasks are as follows:

■ **~~Problem 3~~.** Add stop-word filtering as an optional configuration to the Indexer class (which creates the index). When a set of stop words is provided, these words should be excluded from the inverted index.

■ **~~Problem 4~~.** Add minimum word frequency filtering as an optional configuration to the Indexer class (which creates the index) where any word with fewer than k occurrences is not included in the inverted index.

■ **Problem 5.** (5 points) How efficient are these indices? Let's estimate both the time and space. Using the RegexPreprocessor, tokenize the first 10,000 documents. Then for each inverted index implementation, use the Indexer to create an index with the first 10, 100, 1000, and 10000 documents in the collection (what was just preprocessed).
Record (1) how long it took to index that many documents and (2) how much memory the index consumed. Record these sizes and timestamps. Make a plot for each, showing the number of documents on the x-axis and either time or memory on the y-axis. Tips: If sizing the index at each step is taking too long, you can size the index every 1000 steps.
Estimate how big the positional index might get in memory for the entire collection and how long it would take with that indexer. In a sentence, describe whether you think you could fit the positional index in memory on your own computer and why.

5 Ranking and Relevance

Once you have an inverted index, in the third stage, you'll implement multiple relevance functions to use in ranking documents for a query. Here, we've included several popular methods we've discussed in class. In addition, we'll ask you to implement your own ranking function. Here's the list of what relevance functions you'll implement:

1. inner product (unnormalized cosine similarity) of the query-doc term frequencies
2. TF-IDF: $\sum_{w_i \in q \cap d} \left[log(c_d(w_i) + 1) \cdot \left(log(\frac{|D|}{df_{w_i}^D}) + 1) \right) \right]$
3. Pivoted Normalization: $\sum_{w_i \in q \cap d} c_q(w_i) \frac{1+log(1+log(c_d(w_i)))}{1-b+b\frac{|d|}{avdl}} log(\frac{|D|+1}{df(w_i)})$
4. BM25: $\sum_{w_i \in q \cap d} log(\frac{N-df(w_i)+0.5}{df(w_i)+0.5}) \cdot \frac{(k_1+1) \cdot c_d(w_i)}{k_1(1-b+b\frac{|d|}{avdl})+c_d(w_i)} \cdot \frac{(k_3+1) \cdot c_q(w_i)}{k_3+c_q(w_i)}$
5. Your own idea! (see below)

The notation used above is summarized here. We will rely on the same notation in future homeworks.

1. An arbitrary word is w_i, where i refers to its index within the vocabulary V. A word can be a single token or a multi-word expression.
2. An individual document is d, which can be denoted as d_i to indicate an arbitrary document in the collection. The length of the document in words is indicated as $|d|$, which includes the stop words (before they are filtered).

3. The set of all documents in the collection is D and $|D|$ denotes the number of documents in the collection
4. $|q|$ refers to query length, $|d|$ refers to article/document length.
5. $c_d(w_i)$ is the count of how many times w_i appears in the main text of document d (not including any other metadata/title associated with the document)
6. $df_{w_i}^D$ is the number of documents in D that contain w_i in the body text (not title or metadata). $df_{w_i}^T$ is the analogous value for the number of titles that a word occurs in.
7. $avdl$ is the average document length, i.e., $\frac{1}{|D|}\sum_i |d_i|$

■ **Problem 6.** (8 points) Implement your own relevance score function and describe how you designed it with respect to the axioms we discussed in lectures/discussion sections. In a few sentences, describe your thought process. It's okay to have a badly-performing relevance function but we ask you to try to do this task to build some intuition.

■ ~~**Problem 7.**~~ Implement the ranker step that takes a query and a relevance function as input and returns the ordered list of the most relevant documents.

6 Evaluation

How good are all those relevance functions? In Part 4, you'll implement two evaluation metrics: MAP and NDCG. Both of these require having some ground truth about how relevant documents are for queries (e.g., those + and − symbols we see in the slides are labels for documents!). We've curated a set of query-document relevance scores for you to use in this part of the assignment.

■ ~~**Problem 8.**~~ Implement the MAP and NDCG metrics with a cut-off of 10.

■ **Problem 9.** (10 points) Score each of the ranking functions on the data we provide using the `BasicInvertedIndex` for the full document collection. Use the default hyperparameters in the code. To better understand the performance of the system, you should collect the MAP and NDCG scores for each query. We want you to create one table and one plot using these scores

- Table: Summarize the average performance for each ranker. You should have two rows (one for MAP and one for NDCG) and the columns should denote the rankers. The values should present the average score for that ranker using that particular evaluation metric.

- Plot: Create two plots, one for MAP and one for NDCG. Plot these scores on the y-axis and the relevance function on the x-axis. Use a violin plot to provide the distribution of scores for each relevance function. Use different hues for each metric. We recommend using Seaborn to make this easy. In 2-3 sentences, describe what you see and how similar you think the relevance functions are in terms of performance.

7 Writing Unit Tests

Building complex systems such as search engines is not an easy task. To create a well-functioning codebase, one of the first things one needs to do is create unit tests. In fact, we

provide you with a set of of unit tests to test your code. But our unit tests are only one of the many ways you can (and should) test your codebase. (Indeed, we have other tests that we will use to grade your submission.)

■ **~~Problem 10~~.** (2 points) (Writing Unit Tests) In this part of the homework, you will create your own unit test. We provided you with many unit tests but left one for you to complete. We want you to test whether your implementation is tokenizing a document with a multi-word expression containing punctuation correctly. Complete the relevant function (test_MWE_with_punc).

8 Rating Relevance

Last but not least, where do relevance scores come from? In Part 5, we will *create* our own relevance scores for a variety of queries. This will require you to inspect documents and use your own judgment for how relevant a document is with respect to a query. You should use a 5-point scale where a 5 indicates that the document is very relevant to the query (e.g., think of pages that appear in the first 10 links on Google). A score of 1 indicates the document is not at all relevant. Feel free to use the entire scale range.

We will use the relevance scores you produce to create the training and evaluation dataset in later assignments, so please be thoughtful with your labeling. Each of you is assigned 3 queries to assess relevance. Please annotate no less than ~ 50 documents for each query (there could be more than 50 documents for each query). You will find your assigned queries on Canvas, which is a zip file named in your uniqname.

■ **Problem 11.** (5 points) Label your assigned query-document papers as described above.

While this task can feel laborious, we intentionally included it to help you develop an intuition on what relevance means and how hard (and sometimes, *arbitrary!*) it can be to determine whether a document is relevant for a query. The intuition you build will help substantially later as you think about what new features or methods to use to improve the relevance and ranking parts of the search engine. Also note that the (query, document) pairs we assigned include some obvious answers (clear 5s or 1s). We will use these cases to assess whether you carried out the task faithfully.

9 Late Policy

You have three free late days for this course. All late days are managed through the Canvas. Any submission that is after the scheduled time on Canvas will use one late day. You do not need to ask permission for free late days. We intend you to use them *first* before asking for any other forms of extensions, unless you experience a serious and unexpected life event.

10 What to Submit

Autograder Submission Procedure Your implementation of all the files should be uploaded to the Homework 1 assignment in https://autograder.io. You should already be added

as a student to the course, but if you do not see the course there, please notify the instructional team immediately to be added. The autograder will tell you which files you have to upload.

1. Your code to the autograder
2. A PDF or Word document with your answers and plots to all questions (Problem 2, 5, 6, 9)
3. A file with your annotated relevance scores in the correct format. (Problem 11)

Academic Honesty Policy

Unless otherwise specified in an assignment, all submitted work must be your own, original work. Any excerpts, statements, or phrases from the work of others must be clearly identified as a quotation, and a proper citation provided. Re-using worked examples of significant portions of your own code from another class or code from other people from places like online tutorials, Kaggle examples, or Stack Overflow will be treated as plagiarism. **The use of ChatGPT or Co-Pilot for generating solutions is prohibited; any solution using these tools will receive an automatic zero, regardless of how much of the code these tools were used for.** The instructors reserve the right to have you verbally describe the implementation of any part of your submission to assess whether you wrote it. If you are in doubt on any part of the policy, please contact one of the instructors to check.

Any violation of the University's policies on Academic and Professional Integrity may result in serious penalties, which might range from failing an assignment, to failing a course, to being expelled from the program. Violations of academic and professional integrity will be reported to Student Affairs. Consequences impacting assignment or course grades are determined by the faculty instructor; additional sanctions may be imposed.

Changelog

- rev0 (9/11/2024): Initial Release
- rev1 (9/15/2024): Fixed the TF-IDF formula: from $\sum_{w_i \in q \cap d} \left[log(c_d(w_i) + 1) \cdot log\left(\frac{|D|}{df_{w_i}^D} + 1\right)\right]$ to $\sum_{w_i \in q \cap d} \left[log(c_d(w_i) + 1) \cdot \left(log(\frac{|D|}{df_{w_i}^D}) + 1)\right)\right]$