CS172: Information Retrieval

UC Riverside

Prof. Mariam Salloum Winter 2019

Problem Set 2

You are encouraged to discuss the assignment with your classmates, but eventually each student should sit-down and work on their own code.

1 Description

We will extend the previous assignment to use the inverted index (which is used to store document postings) for ranked document retrieval. This assignment extends PS1 as follows: 1) requires support for complex queries (more than 1 term) using the Vector Space Model (using Cosine Similarity) 2) utilize stemming (as well as stopwords) 3) evaluates various retrieval methods based on speed as well as Precision/Recall.

1.1 Part 1 - Document Indexing

Utilize your PS1 to parse documents and create an inverted index of postings. Utilize standard data structures in Java (HashMaps,ArrayList, etc.) or Python (Dictionary, List, JsonFormat).

This time around, in addition to removing stop words, also apply stemming. We will be using a library to apply stemming. You are free to use any library to accomplish this, but i recommend you use Python (`https://pythonprogramming.net/stemming-nltk-tutorial/`) or Java (`https://github.com/stanfordnlp/CoreNLP` where you want to specifically look at `https://github.com/stanfordnlp/CoreNLP/blob/master/src/edu/stanford/nlp/process/Stemmer.java`).

1.2 Corpus Description

The corpus files provided **(ap89_collection)** are in a standard format used by TREC (Text Retrieval Conference). Each file contains multiple documents. The format is similar to XML, but standard XML and HTML parsers will not work correctly. Instead, read the file one line at a time with the following rules:

1. Each document begins with a line containing $< DOC >$ and ends with a line containing $< /DOC >$.
2. The first several lines of a document's record contain various metadata. You should read the $< DOCNO >$ field and use it as the ID of the document.
3. The document contents are between lines containing $< TEXT >$ and $< /TEXT >$.
4. All other file contents can be ignored.

1.3 Part 2 - Query Execution

Write a program to run the queries in the file **results_file.txt**, included in the data folder. You should run all queries (omitting the leading number) using the retrieval model implemented in Part1, and output the top 100 results for each query to an output file. If a particular query has fewer than 100 documents with a nonzero matching score, then just list whichever documents have nonzero scores.

You should write precisely one output file per retrieval model. Each line of an output file should specify one retrieved document, in the following format:

$< query - number >$ **Q0** $< docno > < rank > < score >$ **Exp**

Where:

- $< query - number >$ is the number preceding the query in the query list
- $< docno >$ is the document number, from the $< DOCNO >$ field (which we asked you to index)
- $< rank >$ is the document rank: an integer from 1-100
- $< score >$ is the retrieval models matching score for the document
- Q0 and Exp are entered literally (because we will use a TREC evaluation code, so the output has to match exactly)

Your program will run queries against the retrieval model implemented Part1. Given a query, you will be retrieving information such as TF and DF scores from the index and implementing our own document ranking. It will be helpful if you write a method which takes a term as a parameter and retrieves the postings for that term from the inverted index. You can then easily reuse this method to implement the retrieval models.

You will use the Vector Space Model to compute the document relevance scores. You will use Cosine Similarity to compute the score between a Query and a given document.

1.4 Part 3 - Evaluation

The **trec_eval** is a tool used to evaluate rankings, either documents or any other information that is sorted by relevance. The evaluation is based on two files: The first, known as "qrels" (query relevances) lists the relevance judgments for each query (this file is given to you). The second contains the rankings of documents returned by your RI system (in this case its the **results_file**). Download **trec_eval** and use it to evaluate your results.

To perform an evaluation, run:

trec_eval [-q] qrel_file results_file

The -q option shows a summary average evaluation across all queries, followed by individual evaluation results for each query; without the -q option, you will see only the summary average. The trec_eval program provides a wealth of statistics about how well the uploaded file did for those queries, including average precision, precision at various recall cut-offs, and so on.

Create a document showing the following metrics:

- The uninterpolated mean average precision numbers for the retrieval model.
- Precision at 10 and precision at 30 documents for the retrieval model.

1.5 README

Include a README file that describes the design of your code and detailed instructions for running your program. We will not grade programs that don't compile, and won't attempt to run your program if the instructions are not provided.

2 Submission

You will be submitting your via Github. Your submission should include all your source files, a README files, and the output data (for the provided data files). The README file should include a short description of what is included in the submission and how to run your program.

NOTE: its your responsibility to verify that your code was checked-in to the Githup repo correctly and ontime.