Assignment 1 - Information Retrieval Systems

CSI 4107
Winter 2023
Due Date: Feb 14th, 2023

Course Coordinator: Diana Inkpen

Group 33
Mai Lyn Puittinen 300124762
Shreya Langhe - 300109724
Joanna Wang 300119385

Task Delegation

Name	Task
Mai Lyn Puittinen	Retrieval and Ranking, general debugging, final combination
Shreya Langhe	Indexing, general debugging, final combination
Joanna Wang	Preprocessing, general debugging, final combination

Program Functionalities

Preprogramming.java will read all the documents in the “docs” folder to a String. From there, the program will parse through the string and remove any punctuations and stop words. Finally, the program calls on the “tokenize” method that splits the string into an array that contains the unique words in the string (aka. the tokens in our vocabulary). These tokens are written into the tokens.txt file.

Indexing.java parses through the documents in the “docs” folder. For each file in “docs”, the program retrieves the relevant information, ie. the document number and text. The program compares the text to the tokens in the vocabulary and computes how frequently that token appears in the text (number of times it appears divided by the number of words in the text). This information is written to a HashMap for each token/document comparison (the inverted index). The total number of documents is also kept track of.

Retrieval.java takes in the HashMap of the inverted index and the total number of documents read. From here, based on the query, documents are scored in similarity and then ranked. The higher scored documents are ranked higher in a list entailing the query, document, rank and score.

Running Instructions

1. Open terminal to project directory
2. If Results.txt already exists in the project, delete the file
3. Compile javac Main.java
4. Run java Main
5. Sit and wait for the program to write the file Results.txt
6. Done

To change the documents being read, place/remove them in the docs folder. To change the queries being used, change the queries.txt file.

Data Structures and Algorithms

Preprocessing: uses Strings and String arrays to represent the documents and tokens. The documents are read into a String. From there, the <TEXT> and <HEAD> portion of the documents are extracted using patterns and matchers and concatenated into another String to be processed. The stop words are then removed from the string using a list of stop words from the ss.txt file and a pattern matcher as well. Punctuation is also removed in the same way as the stop words. The string is finally run through the "**tokenize**" method that takes a string and splits the words on blank spaces and then runs them through a stream that then uses the Java Stream distinct() method used to make sure all tokens are unique.

Indexing: uses embedded HashMaps (Hashmap as the value for a Hashmap) to keep track of each token and its frequency in whichever document it appears in.

The structure is:

```
HashMap<String, HashMap<String, Double>> FinalMap = new HashMap<>();
```

The algorithm loops through all the files in a folder. It takes each individual file and cleans it up using the methods from preprocessing. The document is then separated into its sub documents and compared as an ArrayList of strings to the also cleaned and string ArrayListed token words. If the token word exists in the sub document, the token word is either added to the Hashmap as a new entry or the key is updated with another document and its frequency of occurrence as an addition to the inner Hashmap.

Using Hashmaps made getting and adding information about each token much easier. With Hashmaps there is no need to loop through to search, and can be added on or deleted directly from a specified key.

Retrieval and Ranking:

The class receives a HashMap representation of the inverted index (as described in Indexing), and an int of the total number of documents.

Retrieval.java furthermore reads in the queries as a String Array (generated by the GetQueries class) , where each String is a query. The program loops through each query, where the terms are extracted by splitting the String, and then constructs a "queryMap", which is a HashMap including unique query term keys and their associated weight (tf-idf).

For each token in the vocabulary, the program retrieves the documents that contain the token based on the inverted index. It then calculates token weight and vector length by doing the following:

- ***idf*** is acquired by finding the $\log_2$ of total number of docs divided by the number of docs containing that token
- ***tf*** is the value from the inverted index for the document at a particular token
- ***tf-idf*** is ***tf*idf***

- ***doc_vl*** is the document's vector length calculated by taking ***tf*** and adding (***tf-idf***)^2

These vector lengths are then added to a HashMap. Then, the ***tf*** for each token in the query is given by (how many times the token occurs in the query) divided by (the total number of words in the query). The query ***tf-idf*** (calculated the same way as before) is added to a query HashMap. Vector length for the queries is calculated the same as above. For each word/token in the query and for each document that contains that token, the score is calculated by summing the cosine similarity score of the document. The cosine similarity score of the document is given by the (***tf-idf*** of the word in the doc multiplied by the ***tf-idf*** of the word in the query) divided by the (vector length of the doc multiplied by the vector length of the query).

After all the document scores are generated and saved in a HashMap, the documents are ranked. This is done by sorting the document scores in descending order.

GetQueries:

Similar to Preprocessing.java, but when reading the documents, data is separated into three String ArrayLists, one for each of the queries title, description, and text. Each element at index(+1) in the ArrayList corresponds to the query's number. The three arraylists merge each title, desc., and text for 1 query into a string, use Preprocess.removePunct and .removeStoopWords methods, and add that one combined String into an array where array[i] is equal to the title, desc, and text of query i+1.

Discussion: (Include the first 10 answers to queries 1 and 25. Discuss your results.)

Due to un-optimized aspects of our IR system, we were unable to successfully run all of the documents provided for the assignment. As such, the following results are ran against only 10 documents (AP880212 - AP880221). The results, against 10 documents,

```
1 Q0  AP880218-0080  1 0.1525331072880156 my_search
1 Q0  AP880218-0049  2 0.11675860369678329 my_search
1 Q0  AP880216-0112  3 0.07909393456595254 my_search
1 Q0  AP880220-0132  4 0.07586833778787817 my_search
1 Q0  AP880213-0052  5 0.06714793138332824 my_search
1 Q0  AP880219-0042  6 0.05132761536652572 my_search
1 Q0  AP880216-0089  7 0.05055548471625787 my_search
1 Q0  AP880216-0195  8 0.04714808655429979 my_search
1 Q0  AP880215-0167  9 0.045864370909769935 my_search
1 Q0  AP880212-0062  10 0.04404284203450683 my_search

25 Q0  AP880217-0158  1 0.4482485983095112 my_search
25 Q0  AP880213-0196  2 0.04494768133177808 my_search
25 Q0  AP880221-0084  3 0.04463166777938736 my_search
```

```
25 Q0  AP880217-0201  4 0.04413985937482383 my_search
25 Q0  AP880216-0114  5 0.04178812868341935 my_search
25 Q0  AP880217-0043  6 0.03933689715552633 my_search
25 Q0  AP880219-0013  7 0.03903184763544175 my_search
25 Q0  AP880220-0018  8 0.03626187443062553 my_search
25 Q0  AP880217-0034  9 0.03386054296815337 my_search
25 Q0  AP880217-0148  10 0.032644941559716256 my_search
```

Mean Average Precision (MAP): 0.0027

runid	all	my_search
num_q	all	50
num_ret	all	69725
num_rel	all	2099
num_rel_ret	all	41
map	all	0.0027
gm_map	all	0.0001
Rprec	all	0.0076

Since our system was unable to handle reading all the given files, the MAP score is very low since they are comparing our query results when parsing 10 document files against the query results when parsing all 322 document files.

Queries/Topic Discussion:

The runs where titles and descriptions of the queries return better results since the system is allotted more information to understand what general topic the query is looking for. For example, query 2 (“Accusations of Cheating by Contractors on U.S. Defense Projects”), when given the description, a document that describes the nature of wrongdoing is given a higher document score since it has more opportunity to match with words (like “illegality” or “committed”).

100 Sample Tokens:

witnesses
army
violence
cases
participated
president
leslie
manigat
boycotted
major
candidates
balloting
turnout
secret
ballot
cuban
prisoners
unruly
cell
transfer
officers
inmate
suffered
minor
injuries
fracas
authorities
ranting
raving
throwing
sheriff
aubrey
cole
inmates
trying
escape
housed
county
immigration
naturalization
service
separate
cells

jumped
jailers
opened
moved
hallway
fight
dispatcher
david
weaver
deputy
jack
myers
jailer
eddie
johnson
bruised
medical
treatment
assault
peace
custody
ins
department
jasper
police
help
subsequent
search
uncovered
homemade
weapons
fashioned
metal
electrical
switch
plates
walls
sharpened
wrapped
cloth
nationals
escaped
jail
october

apprehended
stage
director
yuri
lyubimov
soviet
union
greatest
theatrical
figure
says
misses
homeland
welcomes