University of Ottawa
School of electrical Engineering and Computer Science (EECS)

CSI4107 Information Retrieval and the Internet

Winter 2023

Information Retrieval System

Group 14
Emily Lu, 300114727
Feyi, 300120992
Ismael, 300069969

Presented to:
Professor Diana Inkpen

Submission Date: February 14, 2023

Table of Contents

Roles and Responsibilities

Team Member	Responsibilities
Ismael	**Deliverable Document:** Detailed note about the functionality of step 1 in the code. **Code:** Responsible for cleaning and preparing the text data for indexing and retrieving, which is step 1.
Feyi	**Deliverable Document:** Detailed note about the functionality of step 2 in the code. **Code:** Responsible for creating an index of the documents in the collection to make retrieval more efficient, which is step 2.
Emily	**Deliverable Document:** Detailed note about the functionality of step 3 in the code. **Code:** Responsible for retrieving and ranking the relevant documents for each query, which is step 3.

Functionality

Step 1 – Preprocessing:

Preprocess is a function defined in our code that accepts a single document as input. The function extracts the text from the input document in between the TEXT and TEXT tags using regular expressions. The nltk.word tokenize() function is then used to tokenize the extracted text. The non-alphabetic tokens are eliminated, and the remaining tokens are all changed to lowercase. The code then reads the file "stopwords.txt" in read-only mode, splits the file's lines to produce a set of custom stopwords, and then deletes tokens from the list of tokens that are in the set of custom stopwords. The function returns the list of stemmed tokens after creating an instance of the Porter Stemmer class and applying it to all remaining tokens to stem them.

Step 2 – Indexing:

Parameters

The function **buildIndex** handles building up the entire inverted index from the corpus of preprocessed documents. It takes the variable *preprocessed* which is an array where each entry is an individual preprocessed document and the variable *doc_number_tokens* which is an array that maps the Document IDs to the Document names.

Data Structures

The main data structures we use for holding information in the index are Dictionaries (Hash Tables). The benefits of python dictionaries are that they allow for quick access and utilizing a specific library allow for nested dictionary entries. The variable mainIndex will hold the inverted Index, where the keys are terms and the values follow this format (example uses index[term]:
mainIndex[term][0]= IDF Value for the term
mainIndex[1….x] = An array containing the important values for each relevant document to the term, formatted like: [Document Name, TF-IDF Score, TF Score, Number of Occurrence in This Specific Document]

Steps

- Place all terms from every document into a set so it filters for unique terms only
- For each document check for the number of occurrences of each unique term, if the term is present update mainIndex[term] with the Document Name, TF Value, and Number of occurrences.
- Iterate through all terms and calculate the IDF value then for each relevant document,
- Iterate through mainIndex and multiple the calculated IDF to the TF value for each relevant term in a document and append it to the document details array for the relevant documents

Returns

This function returns the mainIndex and weightsArray

Step 3 – Retrieval and Ranking:

Extracting the Queries

The first step was to extract all the queries. We decided to use both query title and query description. However, we could easily edit the function so that it retrieves only the query titles.
queriesArray(fileLocation) is a function that takes a file location (the queries.txt file) and returns an array

of queries with their query number. This function also removes punctuation and common expressions like "The document will".

Function output: [[query number, query], [query number, query], …]

Retrieval and Ranking – Dictionaries and Arrays

The next step was to use the inverted index from the Step 2 to find the set of documents that contain at least one of the query words. Get the query and document weights for the cosine similarity computations. Then we would be able to rank the relevance of each document. retrieval_and_ranking(query, invertedIndex, weightsArray) is a function that takes a query, an inverted index and an array as input. The invertedIndex is the one created in step 2 and the array is an array containing the term weights. In this function, the related documents are determined and stored in a dictionary, the query and document weights are calculated and also stored in dictionaries / arrays, the cosine similarity scores are evaluated using a helper function (cosine_sim), and finally returns a dictionary of documents and their score, sorted in descending order.

Function output: {dict name : score, dict name : score, ...}

Steps for Retrieval and Ranking Function

- Loop through each word in the query to the word frequency as well as the related documents
- Get the maximum frequency in the query
- For each word in the query word frequency dictionary, calculate the query word weight and add it to a query weight dictionary
- Iterate through each relevant document – get the document word weight for each word in the query and append it to a docWeights array. After the doc weights are calculated for a document, compute the cosine similarity score (using the cosine_sim helper function), and add it to a cosimScores dictionary
- Sort the dictionary by values in descending order and return the final dictionary

Writing Results into a File

In a for loop, we run the ranking_and_retrieval function for all 50 queries in the main of the file. Then write the first 1000 results into the Results file.

Instructions

Place the python file in a folder which should contain the following items:
Folder with "content/AP_collection/coll/" which contains all the documents in the corpus
stopwords.txt, which contains all stopwords
queries.txt , which contains all 50 queries

Please ensure you have NLTK downloaded and imported to use the module punkt

Then in your IDE of choice run the python program, it will produce two documents VocabResults.txt, which contains a sample of 100 vocab words and the size of the entire vocabulary. Along with Results.txt which contains the result output of the querying.

MAP Scores Using Trec_eval

runid all tag
num_q all 50
num_ret all 49337
num_rel all 2099
num_rel_ret all 59
map all 0.0006
gm_map all 0.0001
Rprec all 0.0029
bpref all 0.0329
recip_rank all 0.0124
iprec_at_recall_0.00 all 0.0129
iprec_at_recall_0.10 all 0.0010
iprec_at_recall_0.20 all 0.0000
iprec_at_recall_0.30 all 0.0000
iprec_at_recall_0.40 all 0.0000
iprec_at_recall_0.50 all 0.0000
iprec_at_recall_0.60 all 0.0000
iprec_at_recall_0.70 all 0.0000
iprec_at_recall_0.80 all 0.0000
iprec_at_recall_0.90 all 0.0000
iprec_at_recall_1.00 all 0.0000
P_5 all 0.0040
P_10 all 0.0020
P_15 all 0.0027
P_20 all 0.0030
P_30 all 0.0020
P_100 all 0.0024
P_200 all 0.0026
P_500 all 0.0017
P_1000 all 0.0012

First 10 Answers for Queries 1-25 Discussion

A Text file containing the top 10 answer for queries 1-25 will be attached to the submission.

After analyzing our results, we drew the conclusion that the results were not accurately relevant to the query. The cosine similarity scores were within the range of 0 and 1, leading us to believe that our results were correct. However, following a deeper analysis, we realized that the high-ranking documents do not relate much to the given query. A reason for this issue may be that some documents in each file have more than one text section. Refer to Image 1 for an example. When we began the assignment, we made the assumption that each document only had one text section. Our logic was that the first DOCNO matched the first TEXT section, the second DOCNO matched the second TEXT section, etc. Unfortunately, since there may be 2+ text sections per document, we are retrieving the correct text section but assigning it to the wrong document.

```
<DOC>
<DOCNO> AP880221-0007 </DOCNO>
<FILEID>AP-NR-02-21-88 0010EST</FILEID>
<1ST_LINE>u a AM-PlaneCrashes 5thLd-Writethru a0666 02-21 1108</1ST_LINE>
<2ND_LINE>AM-Plane Crashes, 5th Ld - Writethru, a0666,1136</2ND_LINE>
<HEAD>Plane Crashes Kill At Least 21</HEAD>
<TEXT>
   Eds: SUBS 2nd graf to UPDATE with two more plane-crash deaths
reported in California. INSERTS 2 grafs after 21st graf pvs,
`Airport fire...', with detail on California crash, picking up 22nd
graf, `In El...'. Adds one graf on end with names of victims being
withheld pending notification of relatives.
</TEXT>
<BYLINE>By JOHN FLESHER</BYLINE>
<BYLINE>Associated Press Writer</BYLINE>
<DATELINE>MORRISVILLE, N.C. (AP) </DATELINE>
<TEXT>
   A commuter plane that plowed through a
stand of trees just after taking off, killing 12, was only 3 years
old and had just been inspected, the company president said
Saturday.
   At least nine other people died in plane crashes Friday and
Saturday in New Jersey, Texas, California and Connecticut. One man
was missing and presumed dead.
```

Image 1: An example of a document (AP880221-0007) with more than one text section.

Vocabulary Questions

Vocabulary Size: 107646

Sample 100 Vocabulary Words
precend

rajab

monsoon

bodnar

demari

gomers

moppet

gerney

karamanit

kracov

applebi

marylin

adib

thieu

ohler

kilberg

scanner

philadelpia

infam

arrar

wymor

terciari

grandpar

suround

ariv

mubani

playa

rins

holleuf

tigri

psv

quitter

hendel

exchang

regurgit

presuad

busch

romana

southampton

notif

palest

velayatiati

brumidi

sinaloa

unfortu

auxiiari

stormwat

luxemburg

tansitor

undershirt

nment

tyrel

felmer

accuss

backbon

arkansan

redek

ladenburg

goeppingen

samu

nowa

dismemb

daviana

simpler

ayupala

colada

vliet

humco

szep

asiat

songer

unlc

morand

fulmar

macomb

ferrington

motlana

debuskey

cantwel

gurjen

hocherman

twesm

unforget

pruegner

hoptial

sharrom

marit

schoenbaum

reinjur

barnhart

vasconcelos

larrick

roettger

bya

insati

castellano

blazaki

kaushida

azl

fuzhou

Titles Vs Descriptions Discussion

In terms of performance and accuracy for generating the documents relevant to a query using description is superior then only the title. Our chosen ranking approach depends on the similarity between the document and query, with the weights using the TF-IDF approach. The document title can give a lot of information about keywords of the document and its general context, however extra textual detail is necessary to determine the intentions and what the true context of the text is. Extracting and processing the description information is extremely useful for determining the true nature of the text.

From our two different runs we noticed that the rankings for relevant documents were much more correct when we added the query description. The reason for this is because we extract the keywords from a given query, and the gather a list of relevant documents which contain at least one of the query terms. With a longer query – the title with the description – it contains more descriptive words. As expected, there are more relevant documents with the query, but the documents that are truly relevant to the query will have a higher cosine similarity score, placing it at the top of the rankings.