Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

CSI 4107 Information Retrieval Systems

Winter 2023

ASSIGNMENT-1

Implement an Information Retrieval system based on the vector space model,
for a collection of documents

Course Coordinator: **Diana Inkpen**

Submitted by:

Group -03
Rakshita Mathur St#300215340
Fatimetou Fah   St#300101359

Table of Content

Task Division

Step1: Rakshita
Step2: Fatimetou
Step3: Rakshita & Fatimetou
Report writing: Rakshita and Fatimetou

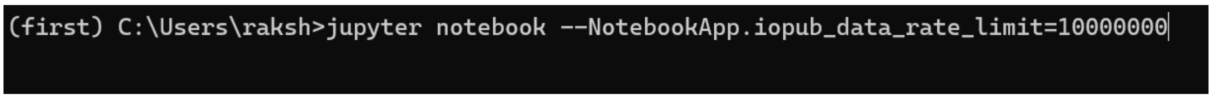
How to run the program

We used jupyter notebook for this assignment.
To run the program, we need to first install the packages that are imported. These are the instructions to run on the terminal/command line to have them installed:

- pip install re
- pip3 install Collections
- pip install nltk
- pip install tqdm

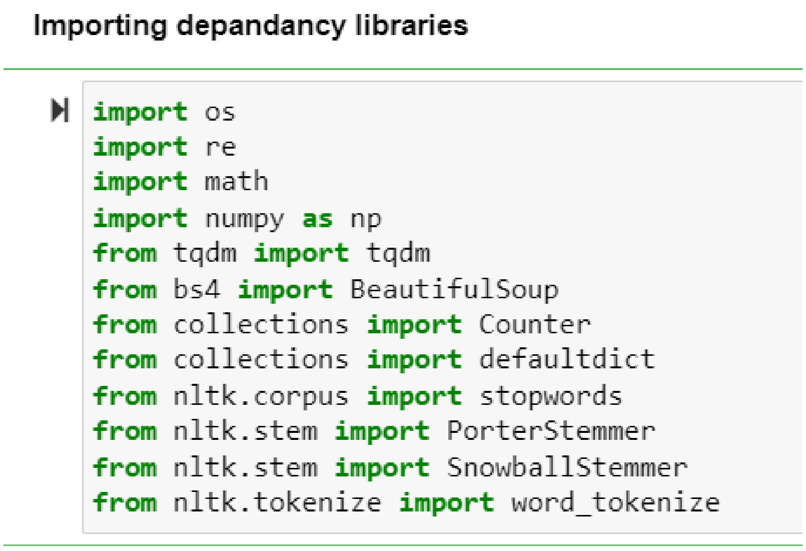
To be able to access the code, you need to download the .ipynb file to your desktop. Then, launch the *Jupyter Notebook App.* When you do so, the notebook will show all the files on your desktop, just open the ipynb file and the code will appear and click on the cell "Cell" and select "Run All".

Also need to run this command on the command line which will increase the limilt of juypter notebook to get all the outputs on the screen.



```
(first) C:\Users\raksh>jupyter notebook --NotebookApp.iopub_data_rate_limit=10000000
```

Package used



Importing dependancy libraries

```
import os
import re
import math
import numpy as np
from tqdm import tqdm
from bs4 import BeautifulSoup
from collections import Counter
from collections import defaultdict
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import SnowballStemmer
from nltk.tokenize import word_tokenize
```

Functionality of the program

In this assignment, we created a program using Jupyter notebook. Its functionality is to implement an information retrieval (IR) system for a group of documents based on the vector space model in order to output the results of the system on a set of 50 test queries. To do so, we had to 3 different tasks to do:

1. implement a preprocessing function for tokenization and stopword removal
2. Build an inverted index, with an entry for each word in the vocabulary
3. Use the inverted index to find the limited set of documents that contain at least one of the query words. Compute the cosine similarity scores between a query and each document.
4. Make a text file called results that'll contain the relevant documents and the queries.

Preprocessing the documents:

Text pre-processing and tokenization are being done by this code. The preprocessing function performs the following actions on a the texts of documents using the following steps:

1. uses a regular expression to eliminate any markup and non-text components.
2. utilises the word_tokenize function from the nltk toolkit to tokenize the text into individual words.
3. removes stop words like "the" and "and." The stopwords.words function from the nltk package is used to create a collection of stop words, which is then filtered through the tokenized words to remove any words that are in the stop word set.

4. Removes punctuation, numbers by checking if the token is alphabetic.

The PorterStemmer from the nltk package is used by the stem_tokens function to take a list of tokens and return a list of stemmed tokens.

The main loop then goes through all of the files in the "coll" directory, reading each one, pre-processing the text, stemming the tokens, and then saving the stemmed tokens for each file in the documents dictionary, where the file name serves as the key and the list of stemmed tokens serves as the value.

Preprocessing the test file:

To preprocess the query file(test_50.txt), we took the whole file and removed any tags, description from it and put it in a dictionnary. To do so, we imported the BeautifulSoup Python package from which we can extract the query number and query text from the space between the num and title tags.

Then, we removed stopwords, new line characters and punctuation using the same approach as the documents. We tokenized and stemmed the tokensand finally ptored the processed query.

Indexing

An inverted index, a type of data structure frequently used in information retrieval for text-based searches, is created using this code. The inverted index associates terms (such as words) with where they appear in a collection of documents.
By using defaultdict(list) to establish a dictionary whose keys that don't exist would by default return an empty list, the code initialises an empty dictionary called inverted index. The terms in this dictionary will serve as keys, and the values will be lists of (document ID, term frequency) pairs.

The code then iterates through each document kept in the documents dictionary, where the document ID serves as the key and a list of tokens as the value (e.g. words). To store the term frequencies in the current document, the code generates a dictionary term freq for each each document. It increases the term frequency in term freq by one for each token in the document. The (document ID, term frequency) pairs for each term are then added to the inverted index by the code. It adds the pair to the list of postings for each term in inverted index for each term and its frequency in term freq.

Finally, the inverted index is printed, with each term and its postings in the format term: [token1: (doc1, freq1), (doc2, freq2), ...].

Retrieval and Ranking

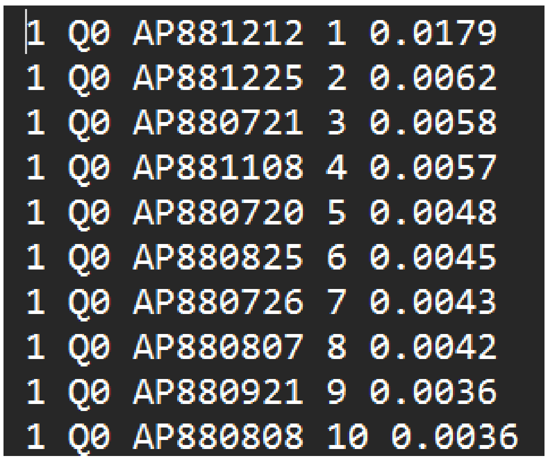
An information retrieval system that uses TF-IDF as its foundation is implemented in this code. Based on the cosine similarity between the TF-IDF representations of the query and the documents, the method accepts a set of documents and a set of queries and outputs a list of the top 10 documents for each question.
Here is a quick summary of what the code accomplishes:
Make a list of all the distinct terms used in each document, then calculate the document frequencies (df) for each term.
Calculate each term's inverse document frequency (idf).
For each document, calculate the TF-IDF representation.
Calculate the TF-IDF representation of each query in a loop, then calculate the cosine similarity between the query and each document.
The top 10 documents for each query should be added to a file called "Results.txt," ranked by cosine similarity.
The term importance in a document is gauged by the TF-IDF representation of that document. The following formula is used by the code to determine the TF-IDF representation for each document and each query:

(0.5 + 0.5 tfiq)idf = tf-idf

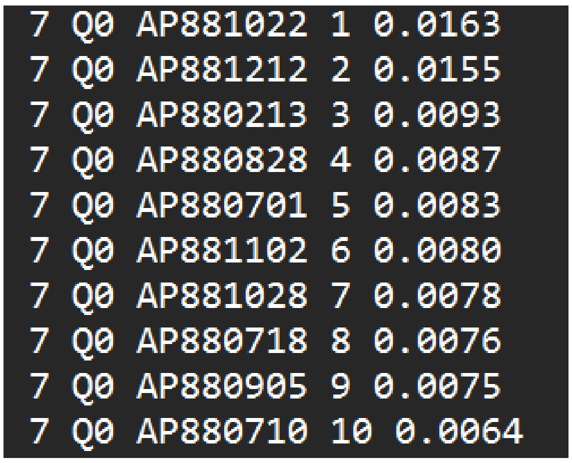
where idf is the inverse document frequency for the term and tf is the term frequency in the document or query. The dot product of two vectors divided by the product of their norms is used to calculate the cosine similarity between the two vectors.

The code keeps track of term frequencies in documents and queries using the Counter class from the collections module. In order to calculate the idf and cosine similarity, respectively, the code additionally uses the math module to compute logarithms and square roots.

Results



```
1 Q0 AP881212 1 0.0179
1 Q0 AP881225 2 0.0062
1 Q0 AP880721 3 0.0058
1 Q0 AP881108 4 0.0057
1 Q0 AP880720 5 0.0048
1 Q0 AP880825 6 0.0045
1 Q0 AP880726 7 0.0043
1 Q0 AP880807 8 0.0042
1 Q0 AP880921 9 0.0036
1 Q0 AP880808 10 0.0036
```



```
7 Q0 AP881022 1 0.0163
7 Q0 AP881212 2 0.0155
7 Q0 AP880213 3 0.0093
7 Q0 AP880828 4 0.0087
7 Q0 AP880701 5 0.0083
7 Q0 AP881102 6 0.0080
7 Q0 AP881028 7 0.0078
7 Q0 AP880718 8 0.0076
7 Q0 AP880905 9 0.0075
7 Q0 AP880710 10 0.0064
```

In the results.txt file we will see all the 50 query running with the top 10 file having the similarity in the decreasing order.

Contents of the submitted archive

The archive A1_300215340_300101359.zip contains this file, Assignment1.ipynb which is the complete code with the output after each steps, test_50.txt which is the test query provided, coll is the collection of documents with 322 files, and Results.txt which is the final file produced after running the last step of the code which has the 50 queries with the top 10 results file.

Références

Bhardwaj, P. (2022, January 6). Data Preprocessing Techniques - AlmaBetter. *Medium*. https://medium.com/almabetter/data-preprocessing-techniques-6b04d820fda2

freeCodeCamp.org. (2018, September 27). *Natural Language Processing (NLP) Tutorial with Python & NLTK*. YouTube. https://www.youtube.com/watch?v=X2vAabgKiuM

GeeksforGeeks. (2023, January 25). *Data Preprocessing in Data Mining*. https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/

Next Day Video. (2012, March 13). *Building A Python-Based Search Engine*. YouTube. https://www.youtube.com/watch?v=cY7pE7vX6MU

NLTK :: Natural Language Toolkit. (n.d.). https://www.nltk.org/

ProgrammingKnowledge. (2020, January 11). *Introduction To Natural Language Toolkit (NLTK) | NLTK Tutorial in Python*. YouTube. https://www.youtube.com/watch?v=WYge0KZBhe0