Information Retrieval
Assignment-2

Akash Rawat (MT21005)
Shubham Rana (MT21092)

Question 1

For this question, we have used the following python libraries:
- Pandas, Numpy.
- NLTK library for pre-processing the data.

Pre-Processing:

1. Read data files from the directory using os.listdir() function.
2. Opened all the files and read the content inside them and stored them in a dataframe.
3. Converted all the words in lower case.
4. Removed all the special characters and punctuation marks.
5. Removed all the stopwords from the corpus using NLTK stopwords.
6. Performed tokenization on the document corpus.

METHODOLOGY:

A. Jaccard Coefficient

- We have unique tokens for each of the queries and the word corpus, we got them using the pre-processing steps as discussed above. After that, we have made the sets for both the tokens of query and word corpus.
- We then fetched the union and the intersection between the set of query tokens and the set of document tokens for each file using the intersection and union operation of python sets.
- After getting the union and intersection we calculated The Jaccard score for a particular document by dividing the length of intersection between the query and document corpus by the length of the union between query and document corpus.
- We then got the top five documents with the highest Jaccard coefficient by sorting the above-obtained scores.

Result:

```
Enter your queryevery two hours until symptoms subside
The list of top-5 document names retrieved:

1st_aid.txt: Jaccard Coefficient: 0.03205128205128205
prooftec.txt: Jaccard Coefficient: 0.02531645569620253
poli.tics: Jaccard Coefficient: 0.019230769230769232
women.jok: Jaccard Coefficient: 0.01775147928994083
japice.bev: Jaccard Coefficient: 0.017543859649122806
```

B. TF-IDF Matrix:

- We created a dictionary for each document, which contain the terms that are present in that document and the count of the words present in that document.
- We then calculated the IDF score for each unique word in the corpus.

IDF score= log of the total number of files / length of the posting list corresponding to that word.

TF-IDF matrix for every Five scoring schemes:-

❖ **BINARY:** For the TF-IDF matrix ROW = Number of docs and COLUMN = Number of unique words in the word corpus.
TF value is 1 if the token is present in the document else 0.
TF-IDF score = tf * idf

❖ **RAW COUNT:** For the TF-IDF matrix ROW = Number of docs and COLUMN = Number of unique words in the word corpus.
TF value is stored in the dictionary created as shown above.
TF-IDF score = tf * idf

❖ **TERM FREQUENCY:** For the TF-IDF matrix ROW = Number of docs and COLUMN = Number of unique words in the word corpus.
TF value = Word count stored in the dictionary created as shown above divided by the total number of words in that document.
TF-IDF score = tf * idf

❖ **LOG NORMALIZATION:** For the TF-IDF matrix ROW = Number of docs and COLUMN = Number of unique words in the word corpus.

TF value = LOG(Word count stored in the dictionary created as shown above divided by the total number of words in that document)
TF-IDF score = tf * idf

❖ **DOUBLE NORMALIZATION:** For the TF-IDF matrix ROW = Number of docs and COLUMN = Number of unique words in the word corpus.
TF value = 0.5+0.5*(f(t,d)/ max(f(t',d))
TF-IDF score = tf * idf

Result:

```
Enter your queryreputable health care practitioner.
Using Binary Weighting Scheme
1: 1st_aid.txt    score: 14.425372944053915
2: pro-fact.hum    score: 9.472190224705745
3: arthriti.txt    score: 9.184508152253965
4: critic.txt    score: 9.184508152253965
5: acronyms.txt    score: 9.184508152253965
```

```
Enter your queryevery two hours until symptoms subside
Using Raw Count Weighting Scheme
1: candy.txt    score: 137.72814734400978
2: mlverb.hum    score: 83.95704580203962
3: humor9.txt    score: 82.3818437455539
4: practica.txt    score: 75.5356307031516
5: vegan.rcp    score: 48.49705678655475
```

```
Enter your queryevery two hours until symptoms subside
Using Term Frequency Weighting Scheme
1: 1st_aid.txt    score: 0.10386255955130275
2: firstaid.inf    score: 0.043729758109026545
3: docspeak.txt    score: 0.03640206552678465
4: doc-says.txt    score: 0.036247819486416914
5: woods.txt    score: 0.025635485211593888
```

```
Enter your queryevery two hours until symptoms subside
Using Log Normalization Weighting Scheme
1: 1st_aid.txt    score: 12.501054577733143
2: coffee.faq    score: 11.869446617089666
3: candy.txt    score: 11.318917688531545
4: bad    score: 10.092789534032725
5: humor9.txt    score: 9.876313495314193
```

```
Enter your queryevery two hours until symptoms subside
Using Double Normalization Weighting Scheme
1: 1st_aid.txt    score: 8.493802818823262
2: bad    score: 4.236190450089574
3: firstaid.inf    score: 4.033035083196812
4: quux_p.oem    score: 3.7953407551448715
5: lozeuser.hum    score: 3.7590589982242655
```

Pros and cons of using each scoring scheme:

1. JACCARD COEFFICIENT

Pros:-

- The sets need not be of the same size.
- The Jaccard similarity score always lies between 0 and 1.

Cons:-

- Term frequency is not considered while calculating the similarity.
- We can not decide which terms come more frequent and which is less frequent.
- The length normalization is not considered.

2. TF-IDF

a) BINARY

Pros: Fast computation of tf-idf matrix as it is easy to compute.
Cons: The term frequency in a document is not considered.

b) RAW COUNT

Pros: Better than the Binary scheme as term frequency is also considered in a document.
Cons: Relevance of a word is not increased if the word occurs more frequently. Example stopwords.

c) TERM FREQUENCY

Pros: It is better than the Raw Count because we are normalizing the occurrence of words with the total number of words in the document.

Cons: If a word is very frequent in a document (Stopword) then the TF score will dominate
and hence further normalization (through using a log) is required.

d) LOG NORMALIZATION

Pros: Here we are normalizing by taking the log so the problem with the above method is solved.

Cons: We are not normalizing each term by dividing it by the document length.

e) DOUBLE NORMALIZATION

Pros:

● The smoothing factor of 0.5 added to the term frequency.

● The tf weights are also normalized.

Cons:

● An outlier term in a document that has a very large number of occurrences but is not representative of the content of the document.

Question 2

Part 1:

Load the dataset using pandas and extracted all the rows with qid:4.

Part 2:

To calculate Max DCG we sort the relevance of all the terms and then calculate the DCG value using the formula:

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

To calculate the number of files that could be made, we calculate the factorial of each relevant class and then multiply them to get the total number of ways.

```
Maximum Discounted Cumulative Gain (MDCG) is: 20.989750804831445

Discounted Cumulative Gain (MDCG) is: 12.550247459532576

Number of possible files that could get MAXDCG are: 19893497375938370599826047614905329896936840170566570588205180312704857992695193482412686565431050240000
```

Part 3:

To calculate the nDCG the formula is: $nDCG = DCG \,/\, Max\ DCG$

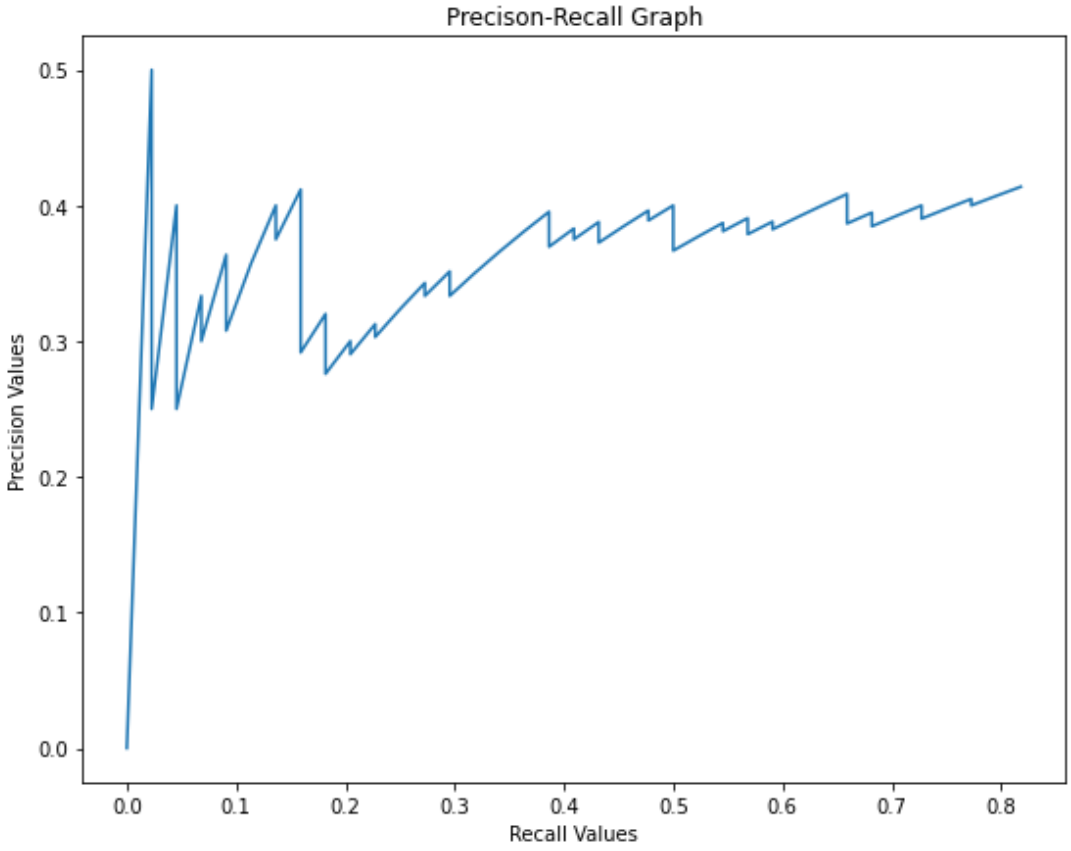
For first 50 dataset, we calculate nDCG for first 50 rows of dataset.

```
Maximum Discounted Cumulative Gain (MDCG) for whole dataset is: 0.5979226516897831

Discounted Cumulative Gain (DCG) for first 50 rows is: 0.5253808413557646
```

Part 4:

We rank the URL based on the 75th feature. The higher the value of this feature, the more relevant the URL is. All nonnegative relevance is set to 1 and then while iterating over the dataset we calculate precision@k and recall@k.



Question 3

For this question, we have used the following python libraries:

- Pandas, Numpy.
- NLTK library for pre-processing the data.

Pre-Processing:

1. Read data files from the directory using os.listdir() function.

2. Opened all the files and read the content inside them and stored them in a dataframe.
3. Converted all the words in lo*\8+
5wer case.
4.0. Removed all the special characters and punctuation marks.
5. Removed all the stopwords from the corpus using NLTK stopwords.
6. Performed tokenization on the document corpus.

METHODOLOGY:

TF-ICF matrix is created similar to tf-idf matrix as shown above. We have made a dictionary in which for each unique word, we have calculated the number of times that word is coming in each of the classes(TF). And using this dictionary only we can calculate the icf value of that particular token.

For implementing top-k features, we have calculated the sum of tf-icf values for each unique token for each class. Then sort them according to this sum. Then we fetched the top-k features for each class and then did the union of these features obtained to get the required top tokens.

The Naive Bayes algorithm is implemented from the scratch.

Part 6:

```
Enter the value of k for selecting the top-k features20
```

50:50 split:

	precision	recall	f1-score	support
1	1.00	1.00	1.00	525
2	1.00	1.00	1.00	506
3	0.97	0.66	0.79	699
4	0.80	0.95	0.87	401
5	0.72	1.00	0.83	369
accuracy			0.90	2500
macro avg	0.90	0.92	0.90	2500
weighted avg	0.92	0.90	0.89	2500

70:30 split:

	precision	recall	f1-score	support
1	1.00	1.00	1.00	291
2	1.00	1.00	1.00	301
3	0.99	1.00	1.00	287
4	1.00	0.99	1.00	293
5	0.99	1.00	1.00	328
accuracy			1.00	1500
macro avg	1.00	1.00	1.00	1500
weighted avg	1.00	1.00	1.00	1500

80:20 split:

	precision	recall	f1-score	support
1	1.00	0.99	1.00	195
2	1.00	1.00	1.00	210
3	1.00	1.00	1.00	198
4	1.00	1.00	1.00	178
5	0.99	1.00	1.00	219
accuracy			1.00	1000
macro avg	1.00	1.00	1.00	1000
weighted avg	1.00	1.00	1.00	1000

We can see that as the size of training data increases or decreases, the Naive Bayes algorithm accuracy remains almost the same. The reason is that Naive Bayes works well even in less training data because it converges easily in less training data also.