Group 1:
Krinal Patel (21CS60R39)
Sarvesh Gupta (21CS60R53)
Arkapravo Ghosh (21CS60R64)
Anima Prasad (21CS60R66)

Task B1 (Relevance Feedback):

To Run this file the command is:
python3 PB_1_rocchio.py model_queries_1.pth queries_1.txt PAT2_1_ranked_list_A.csv ./Data/rankedRelevantDocList.xlsx

OR

python PB_1_rocchio.py model_queries_1.pth queries_1.txt PAT2_1_ranked_list_A.csv ./Data/rankedRelevantDocList.xlsx

Libraries Required for this task are:

- import **pickle**
- import **math**
- import **time**
- import **sys**
- import pandas
- from math import **log2**, **log10**

Python Version:
Python 3.9.6

We are using the output generated from the code of Task-1A (model_queries_1.pth), Task-1B (queries_1.txt), and Task-2A (PAT_2_1_ranked_list_A.csv). Along with this, a given file rankedRelevantDocList.xlsx is also used.

Implementation:
In this part, we are using relevance feedback to modify the queries, with an expectation of better recall. We will consider the output of Part A - Task 2, that is the file PAT2_1_ranked_list_A.csv.
For each query, we consider the 20 documents from the list. And now we have implemented 2 schemes for expanding the query:

- **Relevance Feedback (RF):** In this document is considered relevant if it has a gold standard relevance judgement score is 2, and the remaining documents as non-relevant.
- **Pseudo Relevance Feedback (PsRF):** In this top 10 documents of the list are considered as relevant, and the non-relevant documents set is considered NULL.

We are modifying the query vector using Rocchio's Algorithm:

$$q_m = \alpha q_0 + \beta \frac{1}{|D_R|} \sum_{d_j \in D_R} d_j - \gamma \frac{1}{|D_{NR}|} \sum_{d_j \in D_{NR}} d_j$$

D_R = Set of Relevant Docs

D_{NR} = Set of Non-Relevant Docs
q_m = Modified Query Vector
q_o = Original Query Vector
$\alpha, \beta, and\ \gamma$ = weights or Hyperparameters

Now we compute mAP@20 and NDGC@20, with these modified queries for both the schemes mentioned above. For the computation, we consider the following values of $\alpha, \beta, and\ \gamma$:

$$\alpha = 1, \beta = 1, and\ \gamma = 0.5$$
$$\alpha = 0.5, \beta = 0.5, and\ \gamma = 0.5$$
$$\alpha = 1, \beta = 0.5, and\ \gamma = 0$$

Optimization performed:
Log and power functions are expensive in terms of computation. So, if for each query we compute tf-idf values for all the documents on the go it will take a really long time. So, to improve the run time, pre-computation of the tf-idf vector for documents and queries is required. Furthermore, it is also expensive to create |V| dimensional array explicitly for finding lnc.ltc for each query- document pair. This can be avoided by finding common terms in query and document and performing multiplication operations for those terms only. This method works, as multiplication will be zero if either of one is zero. These optimizations have reduced run time of a code drastically.

Final Output of Task B1:
The output of this code is in 2 CSV files, one for RF and the other for PsRF.

- The file Name of RF metrics is **PB_1_rocchio_RF_metrics.csv**
- The file Name of PsRF metrics is **PB_1_rocchio_PsRF_metrics.csv**

Comparison and Observation:
Values of mAP and NDCG for the weighting scheme lnc.ltc used Part A - Task 2:

mAP@20	**NDCG@20**
0.61	0.65

The output of 2 schemes discussed above (stored in output file too):

RF

α	β	γ	**mAP@20**	**NDCG@20**
1	1	0.5	0.7339	0.7846
0.5	0.5	0.5	0.6516	0.7248
1	0.5	0	0.7370	0.7932

PsRF

α	β	γ	**mAP@20**	**NDCG@20**
1	1	0.5	0.7000	0.7396
0.5	0.5	0.5	0.5252	0.6090
1	0.5	0	0.5457	0.6125

RF is giving better results in all cases because sometimes the query contains synonyms words but not the exact words. Thus, using Rocchio center is shifted towards document center also. Thus, it returns documents that are similar to relevant documents.

$\alpha = 1, \beta = 0.5, and\ \gamma = 0$ give the highest increase, its reason can be that the IR system only needs positive feedback. It returns a document similar to a positive feedback document. Too much negative feedback may result in a poor score, because irrelevant and relevant documents may contain the same words. Thus, it is good to give a higher value of alpha and beta than gamma.

Task B2 (Identifying words from pseudo relevant documents that are closer to the query):

To Run this file the command is:
python PB_1_important_words.py ./Data/en_BDNews24 model_queries_1.pth PAT2_1_ranked_list_A.csv

Libraries Required for this task are:

- import pickle
- from math import log10, sqrt
- import sys
- import pandas as pd

Python Version:
Python 3.9.6

We are using the output generated from the code of Task-1A (model_queries_1.pth) and Task-2A (PAT_2_1_ranked_list_A.csv). Along with this, given corpus ./Data/en_BDNews24 is also used.

Implementation:

- Our goal is to identify top 5 words from pseudo relevant documents that are closer to the query.
- Initially model_queries_1.pth file is loaded to read the inverted index. From the inverted index, we map the token with corresponding indices and also the reverse mapping is stored to retrieve the tokens for a given index.
- Now, from inverted index file, we recreate the corpus and also get the term frequencies.
- Next, PAT_2_1_ranked_list_A.csv file is read, then top 10 documents for each query is retrieved.
- Next, for each query, the tf-idf normalized vector is created using lnc for all 10 documents.
- Next, the td-idf vector of above 10 documents are averaged.
- The tf-idf vector corresponding to each query is stored in a list and then it is sorted in non-increasing order based on the values/weight of token.
- The top 5 tokens based on highest weight/values are written in the output file.

Final Output of Task B2:
The output of this code is in one CSV file.

- **PB_1_important_words.csv :** One row for each query, and consist of 6 columns. The first column is the query id and next 5 columns for top 5 words with largest values.