Task 2A (TF-IDF Vectorization):

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

To Run this file the command is:
python PAT2_1_ranker.py model_queries_1.pth queries_1.txt

Libraries Required for this task are:

- Standard python libraries
- from posixpath import join
- from queue import PriorityQueue

Python Version:
Python 3.9.6

We are using the output generated from the code of Task-1A (model_queries_1.pth) and Task-1B (queries_1.txt). Furthermore, inverted index also stores the term frequency along with the document id in posting list.

Implementation: Our goal is to find the similarity between query and document. We can use cosine similarity to find a similarity between two |V| dimensional vectors. Where |V| is the size of dictionary.
Thus, we have to construct |V| dimensional vector for query as well as document.
One way to construct |V| dimensional vector is to scan entire corpus for each query; however, this is not very efficient approach. This, process can be speed up by storing the term frequency in the inverted index along with the document id.
Furthermore, index of the term should be same in query vector as well as document vector. One way to implement this is to map term to term_id in a range(0,|V|-1) and use term_id instead of term from now onwards.

Term frequency for the term t is the number of times a term occurs in the document. The more the number of times the term occurs, the more is the relevance of the document. However, raw term frequency is not so useful. Thus, the term-frequency of a term can be calculated in one of the below mentioned way.
For Example:

- l means logarithm, therefore the weight of tf, in this case, is 1 + log(tf(t,d)).
- L means log average, therefore the weight of tf, in this case, is (1+log(tf))/(1+log(avg(tf))).
- a means augmented, therefore the weight of tf, in this case, is 0.5 + ((0.5*tf)/max(tf)).

Document Frequency for a term t is the number of documents that contain t. Frequent terms are less informative than rare terms. Furthermore, frequency of stop words is usually high but

they provide no useful information. Therefore, we can use idf (inverse document frequency) to handle this issue.

Different strategies for calculating Document Frequency(idf):
For Example:

- n (no) = 1
- t (idf) = log(N/df), N=#document in the corpus
- p (prob idf) = max {0, log((N-df)/df)}, N=#document in the corpus

Tf-idf weight of term t and document d is basically a product of tf weight and idf weight.
w(t,d) = TF(t,d) * IDF(t)
Normalization is used to normalize the length of vectors, we can use multiple things to normalize, such as:

- n (none) : w/1
- c (cosine) : $w(t,d)/(w_1^2 + w_2^2 \ldots\ldots\ldots + w_n^2)^{0.5}$

To compute term frequency and document frequency, we have used outputs of Task-1A.

Approach to reduce the Computation Time:
In the given dataset, there are approximately 90k documents and 50 queries. Thus, finding which terms are present in the document for each query by scanning inverted index is not efficient. Thus, we can create a dictionary to store the list of terms occur in a document in below mentioned structure.
{document id : list of (term_id,term_frequency) }

Furthermore, the unique terms in a single document and query are very less compare to total unique terms in entire corpus. Thus, it is not a wise idea to iterate over all terms in a dictionary to find dot product. As we know that, dot product is non-zero only for the terms which are present in query as well as document. We can leverage this fact to reduce the computation cost of cosine similarity drastically.

Schemes for Weighting and Normalizing:

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log\frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0,\log\frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Columns headed 'n' are acronyms for weight schemes.

We are supposed to find score using below three schemes:
lnc.ltc
Lnc.Lpc
anc.apc

Final Output of Task 2A:
It is stored in 3 files:

- File 1 ("PAT2_1_ranked_list_A.csv") contains the list of the top 50 highest scored documents for each query in the form (query_id,document_id). Here score is computed by using lnc.ltc scheme.
- File 2 ("PAT2_1_ranked_list_B.csv") contains the list of the top 50 highest scored documents for each query in the form (query_id,document_id). Here score is computed by using Lnc.Lpc scheme.
- File 3 ("PAT2_1_ranked_list_C.csv") contains the list of the top 50 highest scored documents for each query in the form(query_id,document_id). Here score is computed by using anc.apc scheme.

Task 2B (Evaluation)

To Run this file the command is:
python PAT2_1_evaluator.py ./Data/rankedRelevantDocList.xlsx PAT2_1_ranked_list_A.csv ./Data/raw_query.txt

Libraries Required for this task are:

- Standard python libraries
- pandas

- openpyxl

Python Version:
Python 3.9.6

We are using the output generated from the code of Task-2A (PAT2_1_ranked_list_A.csv, PAT2_1_ranked_list_B.csv, PAT2_1_ranked_list_C.csv) and given files (rankedRelevantDocList.xlsx, raw_query.txt).

Implementation:

- Our goal is to find the mean precision , average mean precision , ndcg and average ndcg for top 10 as well as top 20 documents.
- Initially a dictionary is created based on parseQueryDoc.xlxs file, containing query ids along with their relevant document list and respective relevance scores
- Next, another dictionary is created based on output file of PAT_1_ranker.py containing query ids along with their relevant document list
- Next, another dictionary is created based on raw_query.txt file containing query ids along with queries.
- For each query top 10 and top 20 documents from output file of PAT_1_ranker.py are compared against relevant document list in rankedRelevantDocList.xlsx corresponding to same query id.
- If the document is relevant it's considered in precision metric and its corresponding relevance score is used in ndcg metric computation.
- If the document is not relevant then it's not considered in precision metric and its corresponding score is set 0 in ndcg metric computation.

Final Output of Task 2B:
It is stored in 3 files:

- PAT2_1_metrics_A.csv : contains the metric average precision, ndcg value for top 10 as well as top 20 documents for each query. Moreover, it contains the average mean precision , average ndcg value for top 10 as well as top 20 documents for all queries. Input file is PAT2_1_ranked_list_A.csv.
- PAT2_1_metrics_B.csv : contains the metric average precision, ndcg value for top 10 as well as top 20 documents for each query. Moreover, it contains the average mean precision , average ndcg value for top 10 as well as top 20 documents for all queries. Input file is PAT2_1_ranked_list_B.csv.
- PAT2_1_metrics_C.csv : contains the metric average precision, ndcg value for top 10 as well as top 20 documents for each query. Moreover, it contains the average mean precision , average ndcg value for top 10 as well as top 20 documents for all queries. Input file is PAT2_1_ranked_list_C.csv.