Part A - Task 2

TF-IDF vectorization and Evaluation

[**Deadline:** 20th October 2021 11:59 PM]

You are required to build a tf-idf based ranked retrieval system to answer free text queries. You have to use *python* as libraries like *nltk* will make many things easier (stop word removal and lemmatization).

Dataset: https://drive.google.com/drive/folders/1PBdkyftTA86SM9R47rztTlib2ZlYdVVW?usp=sharing

Download the zipped corpus file “en_BDNews24.tgz", the query file “raw_query.txt” and the gold-standard file “rankedRelevantDocList” (download as “csv”) containing the ranked list of documents for each query. Unzip the corpus file and place the extracted folder named “en_BDNews24”, along with the other two files in a *Data* folder within your main code directory. You directory structure should look like:

```
[Main Code Directory]
|
|--------Data
|          |
|          |---------- en_BDNews24/ (Ensure that inside this folder, there are folders named 1,2…)
|          |
|          |---------- raw_query.txt
|          |
|          |---------- rankedRelevantDocList.csv
```

Task 2A (TF-IDF Vectorization)

For this assignment, you have to use the Inverted Index built in Part A - Task 1, i.e. ***model_queries_<GROUP_NO>.pth*** (pickle) file saved in the main code directory. To build a ranked retrieval model, you have to vectorize each query and each document available in the corpus.

- ➔ Consider all the terms (keys) in the inverted index as your vocabulary *V.* Obtain the **Document Frequency** for each term ***DF(t)*** as the size of the corresponding posting list in the inverted index.
- ➔ The **Term Frequency** *TF(t, d)* of term *t* in document *d* is defined as the number of times *t* occurs in *d.* **TF-IDF weight** W(t, d) of each term *t* is thus obtained as W(t, d) = TF(t, d) × IDF(t).
- ➔ Obtain the query and document texts as previously done in Part A - Task 1. Our goal now is to obtain |*V*|-dimensional TF-IDF vectors for each query and each document in the corpus.
- ➔ Represent each query *q* as [q] = [W(t, q) ∀ t in *V*]. Similarly, represent each document *d* in the corpus as [d] = [W(t, d) ∀ t in *V*], where *V* is the vocabulary defined above.
- ➔ Refer slide #45 of Lecture 5 and write codes for implementing the following three ddd.qqq schemes for **weighting and normalizing** the |V|-dimensional TF-IDF vectors:
  - ➢ **lnc.ltc**
  - ➢ **Lnc.Lpc**
  - ➢ **anc.apc**

➔ Rank all the documents in the corpus corresponding to each query using the **cosine similarity metric** as described in slide #40 of Lecture 5.

➔ For each of the schemes, store the query ids and their corresponding top 50 document names/ids in ranked order in a two-column **csv** file with a format similar to "rankedRelevantDocList.csv", i.e.

<query id> : <document id>

Save the following three files in your main code directory:

PAT2_<GROUP_NO>_ranked_list_A.csv for ***"lnc.ltc"***
PAT2_<GROUP_NO>_ranked_list_B.csv for ***"Lnc.Lpc"***
PAT2_<GROUP_NO>_ranked_list_C.csv for **"anc.apc"**

➔ Name your code file as: ***PAT2_<GROUP_NO>_ranker.py***

➔ Running the file : Your code should take the path to the dataset and inverted index file, i.e. ***model_queries_<GROUP_NO>.pth*** (obtained in Part A - Task 1) as input and it should run as:
$>> python PAT2_<GROUP_NO>_ranker.py <path to the en_BDNews24 folder> <path_to_*model_queries_<GROUP_NO>.pth*>

For example, for group 11 your code should run for the following command:
$>> python PAT2_11_ranker.py ./Data/en_BDNews24 model_queries_11.pth

Task 2B (Evaluation)

1. For each query, consider the top 20 ranked documents from the list obtained in the previous step.

2. For each query, calculate and report the following metrics with respect to the gold-standard ranked list of documents provided in "rankedRelevantDocList.csv":
   a. Average Precision (AP) @10.
   b. Average Precision (AP) @20.
   c. Normalized Discounted Cumulative Gain (NDCG) @10.
   d. Normalized Discounted Cumulative Gain (NDCG) @20.

3. Finally, calculate and report the Mean Average Precision (**mAP@10** and **mAP@20**) and average NDCG (**averNDCG@10** and **averNDCG@20**) by averaging over all the queries.

4. For each of the three ranked lists ***PAT2_<GROUP_NO>_ranked_list_<K>.csv*** (K in A,B,C) obtained in the previous step, create a separate file in the main code directory with name ***PAT2_<GROUP_NO>_metrics_<K>.txt*** and systematically save the values of the above mentioned evaluation metrics (both query-wise and average).
   .
5. Name your code file as: ***PAT2_<GROUP_NO>_evaluator.py***

6. Running the file: For each value of K (A/B/C), your code should take the path to the obtained ranked list and gold-standard ranked list as input and it should run in the following manner:
$>> python PAT2_<GROUP_NO>_evaluator.py <path_to_*gold_standard_ranked_list.csv*> <path_to_*PAT2_<GROUP_NO>_ranked_list_<K>.csv*>

For example, for group 11 and K=A, your code should run for the following command:
$>>python PAT2_11_evaluator.py ./Data/rankedRelevantDocList.csv PAT2_11_ranked_list_A.csv

Submission Instructions:
Submit the files:

PAT2_<GROUP_NO>_ranker.py
PAT2_<GROUP_NO>_evaluator.py
PAT2_<GROUP_NO>_metrics_A.csv
PAT2_<GROUP_NO>_metrics_B.csv
PAT2_<GROUP_NO>_metrics_C.csv
README.txt

in a zipped file named: **PAT2_<GROUP_NO>.zip**

Your README should contain any specific library requirements to run your code and the specific Python version you are using. Any other special information about your code or logic that you wish to convey should be in the README file. Also, mention your group number in the first line of your README.

IMPORTANT: PLEASE FOLLOW THE EXACT NAMING CONVENTION OF THE FILES AND THE SPECIFIC INSTRUCTIONS IN THE TASKS CAREFUL. ANY DEVIATION FROM IT WILL RESULT IN DEDUCTION OF MARKS.

Python library restrictions: You can use simple python libraries like nltk, numpy, os, sys, collections, timeit, etc. However, you cannot use libraries that calculate the vectors or the metrics *(like sklearn)*. If your code is found to use any of such libraries, you will be awarded with zero marks for this assignment without any evaluation.

Plagiarism Rules: If your code matches with another student's code, all those students whose codes match will be awarded with zero marks without any evaluation. Therefore, it is your responsibility to ensure you neither copy anyone's code nor anyone is able to copy your code.

Code error: If your code doesn't run or gives error while running, marks will be awarded based on the correctness of logic. If required, you might be called to meet the TAs and explain your code.