# README

## TASK B1

PB_16_rocchio.py takes the preprocessed query file: query_16.txt as an additional **5th** command-line argument as we need the original query vectors to calculate the new modified query vectors

**Run it as:**

*python PB_16_rocchio.py ./Data/en_BDNews24/ ./model_queries_16.pth ./Data/rankedRelevantDocList.csv PAT2_16_ranked_list_A.csv queries_16.txt*

**Algorithm for Rocchio Calculation:**

- We first calculate the tf-idf vectors of documents and queries using lnc.ltc scheme using the same code from *PAT2_16_ranker.py*
- Now we obtain ranked of queries from *PAT2_16_ranked_list_A.csv*.
- The function ***modify_query_vecs*** takes in the calculated query vectors and document vectors and returns modified query vectors.
    - For each query in gold standard file, it separates the docs into two sets: relevant and non-relevant according to relevance feedback from gold standard file if the mode is RF. If the mode is PsRF, it just takes set of relevant docs as top 10 docs from ranked_list_A.
    - It then calls function ***rocchio_formula*** which takes in the two sets and query vector. It returns modified query vector.
- The function ***evaluate*** generates modified query vectors using ***modify_query_vecs*** and generates new ranked results using the modified query vectors. Using these results it calculates mAP@20 and NDCG@20.
- We call the function ***evaluate*** for different values alpha, beta, gamma and modes as PsRF to calculate all the values.

NOTE: Calculating modified query vectors is fairly fast, as the function ***modify_query_vecs*** calculates all the modified query vectors as suggested in the task. The slow step of the code is recalculating the rank of documents using modified queries i.e. the compare function which calculates similarity between query and document vectors. This is because the modified query vectors are no longer sparse as query vectors of 20 documents are added in each modified query vectors, increasing their components in the worst case by 20*(vocabulary of each document), so we printed a nice progress bar to indicate the process

of comparing modified query vectors to document vectors to generated the new ranked list. The code will take approximately 7-10 mins to run.

## TASK B2

**Run it as:**

*python3 PB_16_important_words.py ./Data/en_BDNews24/ ./model_queries_16.pth PAT2_16_ranked_list_A.csv*

**Algorithm for finding important words:**

- We first calculate the tf-idf vectors of documents and queries using lnc.ltc scheme using the same code from *PAT2_16_ranker.py*
- Now we obtain ranked of queries from *PAT2_16_ranked_list_A.csv*.
- Then for each query number, we take the first 10 documents as relevant documents.
- We take an average of tf-idf vectors of these 10 documents.
- Then, sort the averaged tf-idf vector in decreasing order of tf-idf value.
- Extract words corresponding to the first 5 tf-idf values.
- Save these 5 words into `PB_16_important_words`

**NOTE:** We observed that for many queries **"said"** comes in the top 5 words. This is due to the reason the "said" is not considered a stopword in nltk.stopwords library.

## Libraries and version:

- Python 3.9.7
- We used csv, math, pickle, sys, and collections