COL-764 Information Retrieval and Web Search
Assignment -2

1. Rocchio Methods -

Preprocessing data -

- Tokenizing of word using – nltk word_tokenizer
- Using Porter Stemmer from nltk
- Also removing stopwords using list from nltk.corpus

Choice of SUM(d_R) and SUM(d_NR)-

- For sum of relevent documents for a query I took the top 100 queries which we got. (lets call it Sum{d_R}).
- For the sum of Non Relevent documents I took the documents given in 40*100 queries (i.e docs retrieved for all queries let it be = Sum{d}) – relevent docs for that query.
  => Sum{d_NR} = Sum{d} – Sum{d_R}
- Because of this choice we don't need to process the entire data which will take a huge amount of time just for calculating the sum of non relevent docs.

Reading the data (Common to rm1 and rm2)-

Using Method - r*ead_metadata()*
Initially I am reading the complete metadata.csv file and storing the paths for pdf and pmc files along with title and abstract.

Later while generating the vocabulary and vector of all 40*100 documents and 40 queries, using method *contents_for_cord_id()* I am reading the pdf json for documents and pcm json (if there is no data in pdf json) and using them to make the vocabulary (using method – *process_ranked_docs()*)

Performing Reranking-

I am generating the new query using query, sum of relevent docs and sum of non relrevent docs, with parameters (alpha, beta and gamma)

The *reranking()* method executes for every query and then calculate the similarity, between new query and documents (cosine similarity)

For calculating similarity Tf-Idf represenattion is used, tf = (frequency of that terem in doc), and idf = log(N/f) (where - f = Number of docs having that term, N = total number of docs)
So a vector of document will have ith term as = tf_i*idf_i

This result is written in the output file provided.

Results-

The parameters alpha and beta are calculatd using grid search method and the method *train_hp()* has the implementation (this does not run when you execute the code, as the path for relevence docs is not provided)(but its possible to uncommend it and use it)

Using the values --
alpha = 1
beta = 0.03
gamma = 0.01

Values got -
MAP = 0.061
nDCG values -
nDGC @ 5 : 22.774771
nDGC @ 10: 22.351907
nDGC @ 50: 25.515809

While tuning --
For values -
alphas = [1, 0.7, 0.5]
betas = [0.03, 0.3, 0.5, 0.7]

Map scores	Beta = 0.03	Beta = 0.3	Beta = 0.5	Beta = 0.7
Alpha = 1	0.061	0.0589	0.0588	0.0587
Alpha = 0.7	0.0602	0.0588	0.0587	0.0586
Alpha = 0.5	0.0596	0.0587	0.0586	0.0586

2. Language Models Reranking -

Preprocessing data -

- Tokenizing of word using – nltk word_tokenizer
- Using Porter Stemmer from nltk
- Also removing stopwords using list from nltk.corpus

Reading the data (additional tweaks)-

All of the bookkeeping is done using the method *process_ranked_docs()* like sizes of documents, top docs to use as models, docs (dictionary of vector doc representations), etc.

Selecting Models -

Since selecting models is basically selecting documents for each query so using *models_size* variable I am selecting documents with rank <= *models_size*, which gives only top (m) docs for model representation

Currently choosing top 15 docs for representing models for each queries

For RM1 and RM2 -

Pw_dj() Method is for calculating P(w| Dj) or P(w|Mj) using Dirichlet Smoothing

$$\hat{P}(t|M) = \frac{f_{t,d} + \mu \hat{P}_C(t)}{|d_j| + \mu}$$

Approx Timings --
RM1 took around **2hrs** to complete processing in my local system
RM2 took around **3hrs** to complete processing in my local system

Although I have written the code to decrease the time of computation by **picking random values (10K)** and selecting top x words for expanding the term but it decreases the evaluation values (**map – 0.0551, nDGC @ 5 : 21.118493, nDGC @ 10 : 19.02152)** ans took only about **15-20 mins** for rm1 and rm2 each. (Values are for RM1)
But not using this.

Only for RM1-

Class *rm1* handles everything for rm1.

P(w|R) = P(w|q1,q2,...) is calculated using method *P_w_q()*

Expansion Terms for a query -
Using method *expansion_term()* the expansions terms are calculated.

- Going through all the words in vocabulary and finding P(w|R) for all words
- Selecting top x terms with max probability, this is done using variable *top_x_terms*, giving our expanded query
- Calculating the similarity score (using kl divergence) for all the documents using the expanded query. This step is optimized as for Kl divergence we need to calculate sum over vocabulary but since if that term not exist in expanded query kl_div for it will be zero (i.e if foe w – P(w|R) = 0 (then dont calculate P(w|D))

$$\sum_{w} P(w|R) \log P(w|D)$$

So instead of going through all vocabulary terms I am only processing *top_x_terms*.

For all Queries -
The same processs is repeated for all the queries and the result is saved in the output and extensions files provided along with run command.

<u>Only for RM2-</u>

Class *rm2* handles everything for rm2-

P(w) – This is calculated using the method *P_word()*,i.e for all models P(w|Mi) and P(Mi) s are Equally Likely so P(Mi) = 1/total models

P(qi|W) – This is calculated using method *P_qi_given_word()*, This is by computing sum over all the models P(Mi|w)*P(qi|Mi)

Also
P(Mi|w) = P(w|Mi)*P(Mi)/P(w)

$$P(q_i|w) = \sum_{M_i \in \mathcal{M}} P(M_i|w) P(q_i|M_i)$$

P(w,q1,q2,...)/P(w|R)- This is calculated using P(qi|W) for all terms in query and miltiplying by P(w), This is implemented using method *P_word_givn_R()*

$$P(w, q_1 \dots q_k) = P(w) \prod_{i=1}^{k} P(q_i|w)$$

Expansion Term for query-

- This is calculated using method *expansion_term()*, I calculate P(w|R) for all the words in vocabulary.
- Then selecting the top x (using 10 right now) terms
- Then I used the similar way to calculate the similarity score using kl divergence (fasster way as mentioned for rm1)

For All Queries-

The same processs is repeated for all the queries and the result is saved in the output and extensions files provided along with run command.

Results-

The trec_eval was not showing values for nDCG so make a script for nDCG (*eval_rms.py*) using sklearn. (this will not be executed if you run the normal code as no prelevent path there)

The evaluation is done using trec_eval (bundled along the submittion) and the method *evaluate()* has the implementation of this (this does not run when you execute the code, as the path for relevence docs is not provided), although you can unconnent and provide path of relrevence doc to print the result on console

RM1--

At **mu = 0.5**

We can see that these is some improvement in **map** value **0.0650,** as compared to original

The main difference is visible in Precisions as initial levels as we see a major improvements

Mu Values	Map	nDCG@5	nDCG@10	nDCG@50
0.1	0.0643	25.143396	24.625631	25.825309
0.5	0.0650	25.646965	24.712289	26.1519327

RM2 -

At **mu = 0.1**

We can see that thers is some improvement in **map** value **0.0604** as compared to original

Although there is more difference in nDCG values at initial levels, therefore because of slightly more (nDCG@5) mu = 0.1 seems a better option

Mu values	Map	nDCG@5	nDCG@10	nDCG@50
0.05	0.0604	21.9420838	21.7574259	23.9818007
0.1	0.0604	22.0164267	21.7903726	24.0477083

0.5	0.0604	21.94621	21.8202528	24.1011951

Result Dump for RM1-

For Complete data and with mu = 0.5

runid	all	Harsh_rm1
num_q	all	40
num_ret	all	4000
num_rel	all	22724
num_rel_ret	all	1841
map	**all**	**0.0650**
gm_map	all	0.0311
Rprec	all	0.0856
bpref	all	0.0834
recip_rank	all	0.8002
iprec_at_recall_0.00	all	0.8293
iprec_at_recall_0.10	all	0.2063
iprec_at_recall_0.20	all	0.0179
iprec_at_recall_0.30	all	0.0000
iprec_at_recall_0.40	all	0.0000
iprec_at_recall_0.50	all	0.0000
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.7150**
P_10	**all**	**0.6800**
P_15	**all**	**0.6600**
P_20	all	0.6512
P_30	all	0.6250
P_100	all	0.4602
P_200	all	0.2301
P_500	all	0.0920
P_1000	all	0.0460

Result Dump for MR2 -

with mu = 0.1

runid	all	Harsh_rm2
num_q	all	40
num_ret	all	4000
num_rel	all	22724

num_rel_ret	all	1841
map	**all**	**0.0604**
gm_map	all	0.0287
Rprec	all	0.0856
bpref	all	0.0829
recip_rank	all	0.7408
iprec_at_recall_0.00	all	0.7959
iprec_at_recall_0.10	all	0.1982
iprec_at_recall_0.20	all	0.0175
iprec_at_recall_0.30	all	0.0000
iprec_at_recall_0.40	all	0.0000
iprec_at_recall_0.50	all	0.0000
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.6350**
P_10	**all**	**0.6025**
P_15	**all**	**0.6217**
P_20	all	0.6012
P_30	all	0.5817
P_100	all	0.4602
P_200	all	0.2301
P_500	all	0.0920
P_1000	all	0.0460

----------END----------