Programming Assignment 1

R08922160
山下夏輝

1. my VSM

- Terms are from the given model files
  - Top 50,000 row term-frequency terms are used in the vector spaces for prediction.
  - The rest of terms is not used.
  - All of terms are bigram.
- Documents are from the given file-list and NTCIR dataset.
- The shape of the vector space is (46972, 50000).
  - 46972 documents.
  - 50000 terms.
- Weights of documents are the values of TF-IDF with Okapi BM25.
  - The parameters of Okapi BM25 is as follows.
    - b: 0.75
    - k: 1.6
- Weights of query are the values of TF-IDF.
  - TF is row-TF in each queries.

2. my Rocchio Feedback

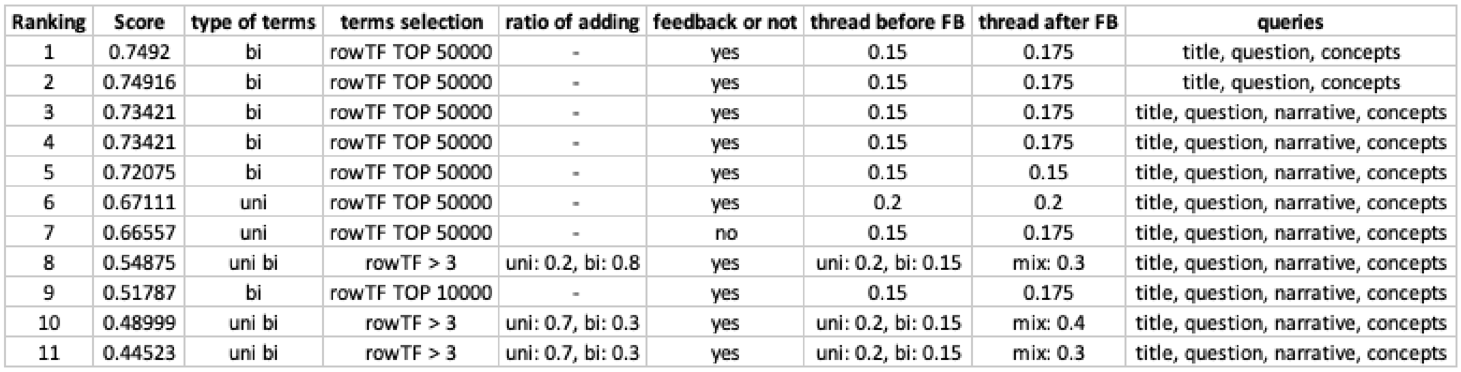
- The values of cosine similarity between TF-IDF of documents and of queries are calculated before retrieval.
- A thread is given when the relevant documents are retrieved.
  - The thread is for the values of cosine similarity between the vectors of documents and of queries.
    - The value of before Rocchio Feedback: 0.15
    - The value of before Rocchio Feedback: 0.175
  - The documents with the values more than thread are retrieved.
    - the number of retrieved document is different because of it
- The parameters of Rocchio Feedback are as follows.
  - alpha = 1

- beta = 0.75
- gamma = 0.15

3. Results of Experiments

The description for the columns of the following figures is as below

- Ranking: the numbers ordered by Ranking.
- Score: the public scores on kaggle.
- type of terms: the type of term used for retrieval.
- terms selection: the way to reduce terms used for retrieval in order to speed up.
- thread before FB: the thread for cosine similarity when the relevant documents are retrieved BEFORE Rocchio Feedback.
- feedback or not: do Rocchio feedback or not.
- ratio of adding: the rate of add cosine similarities between documents and queries with uni-gram vector-space and bi-gram vector-space before retrieving relevant document rangking.
- thread after FB: the thread for cosine similarity when the relevant documents are retrieved AFTER Rocchio Feedback.
- queries: the types of queries used for retrieval



Ranking	Score	type of terms	terms selection	ratio of adding	feedback or not	thread before FB	thread after FB	queries
1	0.7492	bi	rowTF TOP 50000	-	yes	0.15	0.175	title, question, concepts
2	0.74916	bi	rowTF TOP 50000	-	yes	0.15	0.175	title, question, concepts
3	0.73421	bi	rowTF TOP 50000	-	yes	0.15	0.175	title, question, narrative, concepts
4	0.73421	bi	rowTF TOP 50000	-	yes	0.15	0.175	title, question, narrative, concepts
5	0.72075	bi	rowTF TOP 50000	-	yes	0.15	0.15	title, question, narrative, concepts
6	0.67111	uni	rowTF TOP 50000	-	yes	0.2	0.2	title, question, narrative, concepts
7	0.66557	uni	rowTF TOP 50000	-	no	0.15	0.175	title, question, narrative, concepts
8	0.54875	uni bi	rowTF > 3	uni: 0.2, bi: 0.8	yes	uni: 0.2, bi: 0.15	mix: 0.3	title, question, narrative, concepts
9	0.51787	bi	rowTF TOP 10000	-	yes	0.15	0.175	title, question, narrative, concepts
10	0.48999	uni bi	rowTF > 3	uni: 0.7, bi: 0.3	yes	uni: 0.2, bi: 0.15	mix: 0.4	title, question, narrative, concepts
11	0.44523	uni bi	rowTF > 3	uni: 0.7, bi: 0.3	yes	uni: 0.2, bi: 0.15	mix: 0.3	title, question, narrative, concepts

3-1. MAP value under different parameters of VSM

- Type of terms

Look at the columns: type of terms and scores
Using bigram got the highest score. Using uni could get higher score than using mixed one with unigram and bigram.

About mixed one with unigram and bigram, "the ratio of adding" affects the performance. Giving more weight on bigram, the score would be higher.

From these scores and "the ratio of adding", you can know the bigram is the best among three of them.

- Thread for cosine similarity

Look at ranking 4 and 5 rows and thread after FB column.
The score improves when the thread adjusted, which means the number of the retrieved documents affects MAP score.

As a result of adjusting the thread, the score is over given strong base line.

- queries

Look at ranking 1,2 and 3 rows and queries column.
The score improves when the queries used for retrieval change, which means the selection of query type affects MAP score. In the case of ranking 3, "narrative" query type is not used. Therefore, narrative is not useful query type in this case, at least in my model.

3-2. Feedback vs. no Feedback

Look at ranking 3, 4 and 7 rows and the other column.
The score improves marvelously with Feedback. Approximately 0.06 points goes up. Apparently the vectors are adjusted forward good performance with Feedback.

3-3. Others

- Terms selection

In order to speed up, I reduce the number of terms in documents for retrieval. At the first time, I did not reduce it, so the memory overflowed. Secondly, I gave a condition that the terms which appear less than 3 times is not used. Then, the memory-overflow problem was solved but still take a lot of time. Thirdly, I cut off the less frequency terms and just used top 50000 term-frequency terms in all documents. Then the score is not affected and less time is taken to run my program.However, the score is worse when the term used for retrieval reduce up to top 10000 term-frequency terms.

Therefore, at least in my model, reducing therm in some degree is good for efficiency and not bad for MAP score.

- Speed up in Calculating cosine similarity

- Using scipy.scr_matrix, the efficiency of calculation of sparse vectors improved.
    - Originally, using numpy.
- Creating more numpy.array is faster than creating dictionally
    - Originally, I created the dictionally as dealing with inversed-file.

    -
- Using scipy.spatial.distance and sklearn.preprocessing.normarize, the efficiency of calculation of cosine similarity improved.
    - Originally, using cosine_similarity in sklearn, but it was not efficient.
    - Secondly, using distance in scipy and normarization with numpy. At the same time, normarization with math. but the efficiency did not improve. And I found the bottle neck of speed is normarization.
    - Thirdly, using distance in scipy and normarization with sklearn, the efficiency of calculation improved.
    - Finally, using distance and norm in scipy.

4. Discussion

4-1. What I learned

- Programming
    - using scipy for speed
    - creating numpy array, except that creating dictionally for search some values is necessary
- Retrieving with VSM
    - the more term is used, the much more time is need to run the program.
    - the more term is used, the much more attention for memory is needed.
    - the empirical adjustment(Rocchio Feedback) is useful.
    - the hyper-parameters is known empirically, because my model just over the base line with the hyper-parameter which everyone says "in many case works well"

4-2. What I want to know !!!

- Speed UP

I want to know how and what I can do for speed up!!!

As I mentioned above a part of it, I tried a lot of way to speed up: using scipy for sparse vectors, using numpy.array instead of dict, reducing the number of terms for retrieval, reducing the terms in query, reducing the number of candidate-documents for retrieval in advance with thread, using heap queue, using sklearn.

The first model I programmed took more than 2 hours to output the ranking-list. The last model which output 0.74920 MAP score took about 15 minutes. I succeed in saving 2 hours. However, my model is not able to output the result in 5 minutes.

So, what I do is reducing more the number of terms for retrieval. Originally, Top 50,000 terms in the row-TF rangking-list were used. In this model submitted, just 10,000 terms are used. Specifically, the 35000-45000th terms in the ranking list of the row-TF in all documents. In my experiments, the range of terms in 35000-45000th could get higher score than the other range.

In order to save time, I selected to sacrifice the score but I actually do not wanted to do like this. Although I have asked some classmates or elder lab-mates, they also do not know how to speed up more.

So, I would deeply appreciate if TA or Professor kindly explains or demonstrates how to program for speed up.

5. requirement

- scipy
- numpy
- sklearn
- pickle
- xml
- csv
- re
- getopt