B06902125黃柏瑋 WM HW3 report

1. Methodology

1.1 Preprocessing

Throughout the whole experiment, we only use the *model *(including *file-list*, *inverted-file* and *vocab.all*) and *queries* provided in the original dataset for training and testing.

First of all, we read *file-list* to build a list containing file names and their lengths. To shorten our computing time, according to our observations, we divide the size of each file by 3.5 to approximate the file length, rather than diving into the file to count the exact file length.

Next, we focus on *inverted-file*. As we would like to use dot product similarity in our VSM, a dense vector/matrix that only contains query's terms and a sparse vector/matrix that gives 0 to terms not in the query/document will both lead to the same effect. However, the former saves time and memory for the construction of inefficient sparse matrix. Also, it can be easily adapted to different *model*.

The following is a short example:

	APP	FAT	COW	DOG	EGG
query	1	1	0	0	0
document 1	1	2	3	4	0
document 2	4	3	2	1	0

$$sparse: \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 3 & 2 \\ 4 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 7 \end{bmatrix}$$

Then we can truncate zeros in query vector and the corresponding part in document matrix.

$$dense: \begin{bmatrix} 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 7 \end{bmatrix}$$

Thus, without traversing the whole *inverted-file* to construct a huge spase matrix, we decide to record the offset for each term and its raw df only. Then for each query, we can find out the documents including query terms efficiently to construct a short query vector and a dense document matrix.

Eventually, we deal with files in *queries*. We use `xml.etree.ElementTree` as our tool to parse the xml file. Also, we convert the words into indices with *vocab.all* and cut into terms with *inverted-file*. We'll show different query sources of their performance in the experiment part.

1.2 TF-IDF

In the query vector and document matrix, tf-idf weights with Okapi BM25 are filled. Formulas are as follows:

$$TFIDF(d_j, t_i) = TF(d_j, t_i) \cdot IDF(t_i)$$

$$TF(d_j, t_i) = \frac{tf(d_j, t_i) \cdot (k_1 + 1)}{tf(d_j, t_i) + k_1 \cdot (1 - b + b \cdot \frac{|d_j|}{avgdl})} \quad where\ avgdl = avg\ doc\ length$$

$$IDF(t_i) = \log \frac{N - df(t_i) + 0.5}{df(t_i) + 0.5} \quad where\ N = \#\ of\ docs$$

Note that we view a query as document to calculate the weights. Also, the `b` is set as default 0.75 and the setting of `k` will be discussed later.

1.3 Similarity

Because out weights in query vector and document matrix are produced by Okapi BM25, which is a sort of normalization, we don't use cosine similarity but use simple dot product similarity instead. Afterwards, we'll show the comparison between the simple one and the cosine one.

1.4 Pseudo Rocchio Relevance Feedback

Since we don't know the exact revelent set for each query, we try to implement pseudo Rocchio relevance feedback.

$$\vec{q} = a \cdot \overrightarrow{q_0} + \frac{b}{|D_r|} \sum_{\forall \overrightarrow{d_j} \in D_r} \overrightarrow{d_j} - \frac{c}{|D_{nr}|} \sum_{\forall \overrightarrow{d_j} \in D_{nr}} \overrightarrow{d_j}$$

For related part, we pick `Rn` documents with highest similarity as relevant documents and `Rnr` documents with lowest similarity as irrelevant documents.

Hyperparameter `a` is set to 1, `Rnr` is set to 10, and c is set to 0.1. `b` and `Rn` will be discussed in the next part.

2. Experiments

The following experiments are implemented on `query-train.xml`. Scores below are MAP@100.

2.1 Query Sources

We denote `title` as `t`, `question` as `q`, `narrative` as `n` and `concepts` as `c`. We set the `k` in Okapi as 1.2 and use dot product similarity in VSM first.

	TQNC	TQC	TC	C
w/o rrf	0.8295	0.8214	0.8248	0.8386

2.2 Okapi BM25

The recommanded range for `k` is from 1.2 to 2.0. Thus, we pick 1.2, 1.6 and 2.0 for comparison.

K	1.2	1.6	2.0
w/o rrf	0.8386	0.8405	0.8350

2.3 Similarity

	DOT PRODUCT SIM	COSINE SIM
w/o rrf	0.8405	0.8238

2.4 Pseudo Rocchio Relevance Feedback

We first fix `b` to a common number 0.8 and set similarity threshold to pick a good `Rn`.

	1	5	10	15	20
w/ rrf	0.8332	0.8431	0.8375	0.8392	0.8357

Then we fix `Rn` to 5 and tune `b`.

	0.5	0.6	0.7	0.8
w/ rrf	0.8410	0.8437	0.8436	0.8431

Finally, we remove the similarity threshold for comparison.

3. Discussion

In the experiments above, we find out that using only concepts in queries. Maybe that's because there are too many noises in title, question and narrative, making noises in VSM . Also, `k` in Okapi BM25 equal 1.6 is relatively better than others.

In terms of Rocchio relevence feedback, `Rn` equal 5 gain more improvements, and `b` equal 0.6 gets closer to the peak. The reason why `Rn` has to be small may be that tf-idf is not powerful enough and pseudo relevance feedback can aggravate the poor estimation. If we want to improve significantly on pseudo Rocchio relevence feedback, maybe we have to change our weight strategies in VSM.