Web Retrieval & Web Mining Programming HW1

By: Wu-Jun Pei (B06902029)

§ Vector Space Model

The vector space model I use in this programming homework is **Okapi BM25**, the model the professor suggests.

- Term Frequency for term t, document d

$$TF(t, d) = \frac{c(t, d) \cdot (k_1 + 1)}{c(t, d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{\text{avgDocLen}})}$$

  , where $c(t, d)$ is the frequency count for term t in document d.

- Inverse Document Frequency for term t

$$IDF(t) = \log \frac{N - n(t) + 0.5}{n(t) + 0.5}$$

  , where $n(t)$ is the number of documents containing t.

- Score for term t and document d

$$Score(t, d) = TF(t, d) \cdot IDF(t)$$

Following the suggestions on wikipedia, I set $b = 0.75$ and tried $k_1 \in \{1.2, 1.6, 2.0\}$. At last, I choose $k_1 = 2.0$ since it performs the best among the three.

About Implementation

- There's a big gap between the number of unigrams and the number of bigrams. I simply used bigrams as terms.
- Instead of calculate the document length using the document file, I collect the information from the pre-processed data `inverted-file`.

Reference

- Wikipedia - Okapi BM25

- Lecture slides

§ Relevance Feedback

After taking a look at `ans_train.csv` , I found that those answers is not really helpful to our testing set since the intersection between them is small. Thus, I simply implement **Pseudo Relevance Feedback**.

Implementation

Take the top K retrieved documents in the first round, calculate the centroid $\vec{C}_K$ and add it back to our query vectors.

$$\vec{q} = \alpha\vec{q}_0 + \beta\vec{C}_K$$

- Normalize both $\vec{q}_0$ and $\vec{C}_K$ before adding them together.
- Choose $K \in \{10, 15\}$, $\beta \in \{0.1, 0.2, 0.3\}$
- Try to do relevance feedback for several iterations.

§ Results of Experiments

Result Table

ID	Name	k_1	Relevance Feedback	Training MAP	Public MAP	Private MAP
1	`OkapiBM25-0xa0`	1.2	False	0.67366	0.76497	0.72510
2	`OkapiBM25-0xa1`	1.6	False	0.67401	0.76815	0.73523
3	**`OkapiBM25-0xa2`**	**2.0**	**False**	**0.67514**	**0.76994**	**0.74352**
4	`OkapiBM25-0xa3`	2.4	False	0.67273	0.77191	0.73959
5	`OkapiBM25-0xa2`	2.0	K = 10, ß = 0.1	0.66396	0.77130	0.74431
6	`OkapiBM25-0xa2`	2.0	K = 15, ß = 0.1	0.66396	0.77130	0.74431
7	**`OkapiBM25-0xa2`**	**2.0**	**K = 10, ß = 0.2**	**0.65258**	**0.77187**	**0.74574**
8	`OkapiBM25-0xa2`	2.0	K = 15, ß = 0.2	0.65258	0.77187	0.74574
9	`OkapiBM25-0xa2`	2.0	K = 10, ß = 0.3	0.63628	0.76276	0.74736
10	`OkapiBM25-0xa2`	2.0	K = 10, ß = 0.05 5 iterations	0.64840	0.76833	0.74401
11	`OkapiBM25-0xa2`	2.0	K = 10, ß = 0.05 10 iterations	0.60820	0.74202	0.65424

Some notes:

- The **bold and underline** submissions are the two I selected for final score.
- The training MAP is calculated based on `ans_train.csv` .

About Vector Space Model

1. Okapi BM25 is strong enough to ~~pass the strong baseline easily~~ beat the public strong baseline. However, it's a little bit far from (-0.02) the private strong baseline.
2. Tuning the hyperparameter k_1 will not make a huge improve. Choose $k_1 = 2.0$ might be a relatively better choice, compared to others.

About Relevance Feedback

1. The relevance feedback performs much worse (-0.02) on training set but gains a little improve (+0.002) on public/private testing set.
2. There is no difference between choosing $K = 10$ and $K = 15$. The reason might be the centroid between them are so similar that the modified query vector are also

similar.

3. As for β, choosing either 0.1 or 0.2 will lead to similar result. However, when $\beta = 0.3$, the result becomes terrible (`-0.015`) on training set.
4. I've also tried to increase the iterations to perform relevance feedback. However, the result turns out to be even worse (`0.05` for 10 iterations, compared to the 7-th one).
5. To conclude, I think increasing K, β or number of iterations blindly will make the output less precise since the query vector has been modified too much.

§ Discussion

In this homework, I learned

1. Implement a VSM model, I found that a simple Okapi BM25 is a not bad model for document retrieval.
2. Although the model is not bad, I found it hard to improve from tuning parameters and applying relevance feedback. To improve the performance, I come up with the following ideas:
   - Use more features.
     Besides "term frequency", what we already used. I think adding additional features will also help a lot. For example, the quality of the news resource.
   - Try dimension reduction.
     The large matrix TF is also a thorny problem. However, the computation complexity applying SVD might also be a big problem.
   - Change the retrieval model.
     I believe applying the state-of-the-art language model as the document encoder will also help. However, there will also be a computational overhead.