Text Processing Report– Assignment (IR)

Qingyu Yang[1]

[1] Ucard No. 001722400   Registration No. 190256746

1. Assignment Purpose and Aim

This assignemnt requires to build and complete the reserved python class *Retrieve* by finishing three models named Binary, Term Frequency(TF), Term Frequency Iverse Document Frequency(TFIDF) from the text processing.

For the other classes, like *CommandLine*, *IndexLoader* etc. wich have been provided to transform the data structure and operate the file of store rusults input and output. The generall programme structure has been clarified and the consummator should follow the definition of different scheme to analyse the data from different sourece files which have been included in the assignment materials folder.

Otherwise, the efficiency of programme and method are also required to be minimal and marked as a part of the grades.

2. Instruction of assignment

I have built the *shell* file named *script.sh* to redirect all the screen printing results into one single file *show.txt*.

3. Different Schemes Analysis

3.1 Binary

As the hint in assignemnt guidebook given in binary scheme, we don't need to mention about the size of the query vector which is constant across the comparisions which can be dropped without affecting how the documents are ranked. The key to solve problem is to extract the similarity information from the data-structure (*IndexLoader)* and compare with the query vector. For dissimilar occurance of terms in query and documents, we only count 1 in each vector. The procedure of calculating the binary similarity:

1)Extracte the terms in each query and utilize the terms to return to the *IndexLoader.index* and obtain all (*doc_id:count*) from the *Retrieve. getCandidate(self)*.

2)Create a new list to store all the repetitive *doc_id*, count the number of occurance and this result is the molecule of the model.(∑*qidi*)

3)Obtain the length of the each document(*doc_id*),we can calculate it by the sum of all *count*'s square.(*Retrieve. doc_length(self, termWeighting == binary)*)

4) *Retrieve.Binary_model* figures out all the similarity for each document as the formula(1.1) and finally sort its' values in descending order and return the result list *topscore*.

$$\text{Score} = \left(\frac{\sum q_i d_i}{\sqrt{\sum q_i} \times \sqrt{\sum d_i}}\right) \qquad (1.1)$$

3.2 Term Frequency (TF)

As for the definition of term frequency, we should not only focus on the occurance and number of terms in documents, also consider the number of terms showed in each query.For the length of document, we can use the same method which is meantioned in Binary, but this time we cumulus the sum of count's square to reprensent the length. The procedure of calculating the TF similarity:

1) Calculate the length of the documents by the default method, which cumulus the sum of counts' square.

2)Traverse each the query to get the terms and find the terms corresponding basic data set in the *IndexLoader.index*. Create a new dictionary variable to store *doc_id:{term:count}* to facilitate later calculations.(*Retrieve. getCandidate*)

3)Obtain the set of terms in each query, and search for the *tr_candidate_all* to calculate the cosine score according to the formula(1.2) and get the result for different document.

$$\text{Score}=\frac{\text{Term occurance numer(Q)}\times\text{Term occurance numer(D)}}{\text{Doc\_Length}} \quad (1.2)$$

3.3 Term Frequency, Inverse Document Frequency

As for the definition of TFIDF, we should primarily calculate the IDF value for each term and it's dictionary values as the formula(1.3):

$$\text{IDF}=\log\left(\frac{Number\ of\ all\ the\ documents}{Number\ of\ documents\ which\ contain\ the\ term\ in\ Q}\right) \quad (1.3)$$

The IDFs value are stored in a newly created dictionary and the procedure of calculating the similarity is generally same as the TF. But we should multiply the IDF value for each term. It can decrease the weight of terms which are universal and increase the weight of terms which are rare.As a result, the documents which have so much rare words can be ranked equally.The formula can been seen below.(1.4)

$$\text{Score} = \left(\frac{\text{TON(Q)}\times\text{TON(D)}\times\text{IDF(T)}}{\text{Doc\_Length}}\right) \qquad (1.4)$$

3. Result

Weighting Scheme	No Stoplist No Stemming	No Stoplist With Stemming	With Stoplist No Stemming	With Stoplist With Stemming
Binary	P = 0.07	P = 0.09	P=0.13	P=0.16
	R = 0.06	R = 0.07	R=0.11	R=0.13
	F = 0.06	F = 0.08	F=0.12	F=0.14
Time	0.31	0.34	0.12	0.15
TF	P=0.08	P=0.11	P=0.17	P=0.19
	R=0.06	R=0.09	R=0.13	R=0.15
	F=0.07	F=0.10	F=0.15	F=0.17
Time	0.30	0.33	0.13	0.14
TFIDF	P=0.18	P=0.23	P=0.21	P=0.27
	R=0.14	R=0.18	R=0.17	R=0.21
	F=0.16	F=0.20	F=0.18	F=0.24
Time	0.56	0.49	0.33	0.29

4. Evaluation

From the result form we can see that for the same scheme the time reduce as changing the source file (-s -p command). The reason for that is that the nuber of index(source file size) is reduced and the weught of uncommon term increase because of different algorithm.Among these three different scheme, we can see that the time comsumes twice from Binary to TFIDF because of increasing the algorithm complexity and elements for caculating the similarity betwenn documents and queries.

For the differnet configuration, we can see that for the Binary and TF scheme the effect of using stop words(S) is better than the stemming(P). In contrast, the effect of stop words in TFIDF is worse than using stemming words.

TFIDF scheme has the best precision among these three schemes with the same stop words and stemming configured, which has a little more time consuming. And comparing with the Binary and TF, the improvement is very obvious. For the Binary and TF scheme, the precision seems remain at the same level, 0.07 at Binary and 0.07 at 0.08 without stop words and stemming.