Web Retrieval and Mining

Report for Hw1

資工二 B00902064 宋昊恩

1. Implementation:

Initially, I hesitated whether to use `C++` or not. Since that I am not familiar with `Java`, `Python` and other kinds of languages, the only language I can use is `C++`. However, it is quite difficult for `C++` to parse `XML` and Chinese character in `UTF-8`. I soon encountered a very big challenge after choosing the `C++` as my main language. The `XML` parser plug-in for `C++` is very difficult to understand and use. After studying the document of `Xerces C++ XML Parser` for more than four hours. I decided to parse the `XML` document by my own self.

The second challenge I met was that I cannot find a proper parser for Chinese character in `UTF-8`. I browsed a lot of documents from `google` and still cannot find a easy-to-use one. Finally, I chose to deal with the problem by myself, too.

My grading method is quite different from usual, but I still get about `.7339` precision of `query-5`. I will state my method as following.

My first thought is to use all vocabularies as vector and build up vectors for queries and documents. However, I got the information from the email from `TAs` that if I loaded all the content of file `inverted-index`, the memory might be inadequate. Then, I came up with a new idea that I only concerned the “key words” which I decided, and not caring about all the others. In this case, my vector's dimension is just the size of these “key words” which are much smaller than the prior one.

That is to say, I will first determine the word which I concerned. And then, build up the query vector with the occurrence of the very word in the “whole” query file as element value. Next, I use these “key words” to collect the documents which contains these words, then, build up the vectors just the same as query vector.

To fulfilled and accelerated the above-said evaluation method, I needed to first analyze the file `inverted-index`, and get the start position for every single term then store them in the file `vocab.revised`. Note that, each line of the file consisted of three integer. The first one and the second one indicate a single bi-term, which may be `unigram` or `bigram`, and the third one record the distance from the beginning of the file `inverted-index` of this term in btye.

Second, I will build up a file called `file.revised`. Each line of it contains one string stands for the filename, and three integers represented the kinds of terms, the total terms and the maximum times of a term's appearance.

After all the tools I need to use is prepared, I began to parse the query file. I first parsed the arguments of the file, loaded all the data that I have prepared. Then, I put every word into a vector. After that, I used these terms to find out all the files that I need to concern and then build up vectors for documents. Finally, I did some calculation and evaluation and print the first one hundred fitted filename to the target file.

The module I used for `tf` and `idf` is very different from usual, either. The final `tf` I used for document is like `4 + log(tf)`, and the `idf` is `(5 + log(NumberOfTerms/ KindOfTerms)) * 100`. For the module for query file, I used the most normal one, `tf * idf`. It might seem unreasonable to use this very weird module, but its performance was very well in the test for `query-5`.

I perform the query with feedback by combining the top `K` related files' vectors to the original query's one with a weight value constant `W`.

The feedback performance, however, was not very good in the begin. If I concern only the first `10` files as related file, and let the weighted value be `1.0`, the precision rate will reduced to `0.6954`. This situation might be led by the over-fitting for `query-5`.

Nevertheless, after I tuned the constant for many times, I gradually got a conclusion. The more of the weight value W I used, the worse result I will get. The best constant value I have gotten now is to let `K=25, W=0.05` and the result for `ranked-5` is `0.7435`

2. Some testing result:

1) `TF=4+log(tf), IDF=(5+log(NumberOfTerms/KindOfTerms))*100`

a. Without feedback:	`0.7339952104914644`
b. With feedback, `K=10, W=0.1`:	`0.7352723573864086`
c. With feedback, `K=20, W=0.1`:	`0.7422294480700293`
d. With feedback, `K=25, W=0.05`:	`0.74347000443139`

2) `TF=(4+log(tf)), IDF=200+log(FileNum/FilesRelatedTermNum)`

a. Without feedback:	`0.7339952104914644`
b. With feedback, `K=10, W=0.1`:	`0.6969689860229119`

3) `TF=(1+log(tf)), IDF=log(FileNum/FilesRelatedTermNum)`

a. Without feedback:	`0.49661999747901253`
b. With feedback, `K=10, W=0.1`:	`0.5138530056856206`

3. Reference:

The whole program is implemented by my own self. I took a lot of advice from my classmates, and learn a lot from the documents on `google` website.