Web Search and Text Analysis
Assignment 1 Report

The given task is to develop a search engine. What I need to do is to find out the relevant documents in a given corpus (about 15,948 text files) for each query of 50 queries. The key problem is how to judgment whether two documents are relevant. The implementation is based on Vector Space Model(VSM). A ranked result list would be return for each query. P-R curved is used to evaluate the result. Since the fact that it is quite a tough job when it search in a large corpus meanwhile the waiting time of the users is poor. Not only the precision and recall should be cared about, but also the efficiency of time and storage should be considered.

The implementation and evaluation would be discussed in the following part and some thoughts and insight of this project can be concluded at the end of this report.

The implementation would be described in this part. According to the model of VSM and just simply follow the instruction on the specification, the given corpus should be analyzed. The plaintext is tokenized to tokens and the redundant information in the file like punctuation, stop words is stripped. Bag of words are extracted from the raw plain text. I observed that there are some long strings occurred in the corpus and it is meaningless to search result like website addresses. The length of the string are limited as all the queries we work on are short-length word. The token longer than that would be stripped. I reduce some words from the corpus I need to search for. The tokens, located file name and term frequency would be recorded in an inverted index. The inverted index is devised and stored in a nested dictionary like: { term: { 'documentID': term frequency}}. Once the term frequency is calculated and given, the IDF of the term can be calculated. The inverted index can be updated as: { term: { 'documentID': TF*IDF} }. The weight should be nomalised by divided the length of the files. { term: { 'documentID': TF*IDF / len(file)} } In terms of 50 queries, we regard it as 50 pseudo documents. Similarly, the inverted index for the queries can be produced. We got two inverted indexes: one for queries and one for corpus. The weight of one query to each file can be calculated: for the single word query, the relevant files are the files store in the posting list. For multiple word queries, all of the files that word located are regarded as relevant. If a query's tokens all occurred in one document frequently, it is obvious that the weight is high and probably be relevant. Observed that most tokens in the query only occurred one time, so the TF is 1. Each query would be regard as a pseudo document separately, so N = 1 ,IDF = 1. The weight of each token in query is 1. It makes easier for calculating the weight of query to document. For instance, if we want to find out the relevant files of query 851, we just need to facilitate the documents' inverted index to calculate the weight. Tokens in query 851['March', 'Penguin'] got two posting list in documents' inverted index: { 'March': {'doc1':0.4, 'doc2':0.6,'doc4': 0.8}} and {'Penguin':{'doc1':0.2, 'doc5':0.7}}. The query processing result for query 851 ['March', 'Penguin'] is {'doc1' : 0.6, 'doc2':0.6, 'doc4': 0.8, 'doc5':0.7}. The files of 'doc1', 'doc2', 'doc4' and 'doc5' can be said are relevant to the query 851. Each query in 50 queries would get a list of file names by its weight.

The effective of this method is not so good. The reason is that When I return a group of files that the system deemed is relevant to a multiword query, some files is actually not relevant to the query. The precision decreases. When user input "March of Penguin", he is probably search the information about the movie of "March of Penguin" like comments, released time etc. Even a document only contain one of query's word, the system would think that the document is relevant to the movie "March of Penguin" since the weight may be high. An article discussed about the season of March would be returned to users when users search for the movie "March of Penguin". Positional posting list can solve this problem in a good way. Positional posting list record the location of the word in a file. We can know whether a multiword query really occurred in a file. We can even know the tokens occurred in one sentence. For the longer query the positional posting list work well. However because every word's position required be parsed and recorded, it require a higher capacity of storing the inverted index and more time to parse the text. A small subset of the corpus was tried to implement by using positional posting list. Compared to the normal posting list, the precision is optimistic, the rate of precision improve 27%.

I use P-R curve to evaluate my system. 'K' is the number of result I used for evaluation. K also is the amount of result that system return to user. Let 'K' range from (10, 20, 30 … 100). A group of value of precision and recall at K were calculated and The P-R curve can be drawn by these values. Assumed K=1, it means system only return a result for the user and that result is probably be relevant. When we input a query on a search engine, the first result on the return page is what we are looking for. When 'K' grows up, the precision would fall down and the recall may increase. In other words, more search result I got, more relevant results I can have.

I do a query expansion for current queries. Some related words are added to the query like synonyms. More documents would be deemed as relevant documents include some irrelevant documents. For instance, if a query contains a token like ' apple', then the word 'fruit' would also be added to the query's token list. The documents contain the word 'fruit' would be included in the research result. As a result, the recall is probably increased however the precision would decrease. The normal P-R curve and the P-R curve with query expansion have a cross point on the graph or P-R curve. When K is small, the normal P-R curve performs better than the P-R curve with query expansion. When K grows up, the P-R curve with query expansion preform better than the normal P-R curve. The two strategies can be combined for satisfying users' information need. When the amount of the results that users require (K) is small enough, we can select the results that produced by the normal method. When Users want to look for more searching results, the method that applied query expansion can be used. Different model for judging relevant usually have different performance. For improving the effectiveness, both models can be applied to search the result and only the results performed well would be returned to users according to different value of K.

Vector Space Model was applied to develop a searching engine. Bag of word can be produced from the corpus. Inverted index of the corpus is produced and VSM model work on it. Sometimes we need to apply different models and method according to different searching requirement so that we can get the better performance.