Coursework1 (S2311720)

1. Preprocessing

In this coursework. I implemented tokenization, removestopwords, and stemming functions to do the preprocessing. For the tokenization function, I used regex formula to get only words and numbers in the given text; for the removestopwords function, I first imported stopwords from the file provided by the coursework, then I put each stop words in the list and compare with the tokenized text and finally get tokens without stopwords; for the stemming function, I used nltk.stem package and import PorterStemmer to do the stemming for each words. For the convenience purpose, I used function called preprocess which contains all the process above and return a list of preprocessed terms.

2. Inverted Index

For the inverted index part, I first imported xml.etree to parse the xml file provided in the coursework, I used the find() method to find D0CN0 which contains the article number, find HEADLINE and TEXT and then combine them together because these two are all about the main context in the article. I stored inverted index into a dictionary of dictionary which the key is terms, and the value is dictionary with article number as they and the corresponding position as the value. I used this method because it is clearer for me when writing that information in the file.

3. Search Functions

For the search functions, I wrote a function called searching to deal with all different search function. In this searching function, I have a special function called special_pre to do the preprocessing for the input queries because the stop words list contains words like AND, OR, NOT, I must make this new function to deal with the stop words in queries. For checking if the given queries have AND OR, NOT or not, I create three variable called sig_and, sig_or, and sig_not to signal if the query have each of these three words. Then I check all different situations that will happen corresponding to these three words that are A and B, A or B, not A, not A or B, not A and B, A and not B, A or not B, and change the value of three signal above and put each term into different variable for later use. First, if all these three variables are false which means the query only contains terms. Then I separate them into three situations which are 1 term, 2 terms, and 3 terms. For the 1 term, I have a function called single_term to find the term’ s document index numbers. For the 2 and 3 terms, I have a function called multi_terms, which contains the distance of two terms; the 2 terms query (Phrase search)have distance 1, I only need to check if the two terms’ appear distance positions in the document is one, if it is, I will append them in the result list and return; for the 3 terms query(Proximity search), the first term is the distance number, so different from the 2 terms part, I need to check if the within distance of two terms. If at least one of the logical signals appears, then separate them into different lists, then I have a function called booleanS to deal with different combinations of three logical signals and return the result, writing the result into file called results.boolean.txt. For ranked IR based on TFIDF, I loop through each query term to find the df, then use the searching() method to apply on each term. The result is a list of document indexes, then I looped through it to find the tf in each document. For each term in the document, we will have a score by applying the formula provided in the coursework, and limited the result to the top 150,

writing the result into the file called results.ranked.txt.

4. Brief Commentary

The whole system is very functional, from the preprocessing part to the inverted index to the use of searching functions, and all these three parts in the system is corresponding to each other for example, in the preprocessing part, you can decide if you want to contains numbers or not, or if you want to remove some special stop words from it, all these changes in the preprocessing part will finally cause the different inverted index as well as the searching function results. Things I have learned through this course work: at first, I wrote all parts together in one function, and this makes the debug process extremely difficult, then I reconstruct the code by following the process above and separate them into different parts to debug more easier; also, I become more familiar with data structure because the term and document index and corresponding positions in the document is stored in a nested dictionary.

5. Challenges

I have several challenges through this coursework. At first, I want to store terms, document indexes and corresponding positions in separate lists, but I finally found it is difficult to loop each list and it needs several for loops to achieve that, then I changed my data structure to dic of dic, although it may not be the prefect one, it makes the process of coding more clearly. Also, the different combinations of logical operators are also quite challenge, at first, I want to come up with a structure that can automatically handle different combination, but while coding, I found it is extremely difficult to do that, so I decided to write every case of combination.

6. Potential Improvement

I think there will be a better way to store terms, document index and corresponding positions because a nested use of dictionaries could cause the processing process very slow, other data structure may can be applied like trees or hash tables to improve the efficiency. Also, I have some nested for loops in the code, which may cause the processing time to become longer because the running time become n square, I may can find some algorithms to alternate the nested for loops.

7. Challenges (For Extra Marks)

With stop words, the size of index becoming larger, the running time is shorter compared to without stop words one. The Boolean result does not change with or without stop words, that may because Boolean only shows whether the term contains in the certain document number, and the stop words will not affect that. The Ranked result score (with stop words) will stay the same if the query does not contain any stop words, and if the query contains stop words, the score will become higher with stop words, that may because stop words are also contained in the inverted index, and therefore those stop words will have high score and the overall score of that query will increase.