Text technology for data science Report

Author: Senhao WANG Student Number: s1905759 Date:19/10/2019

SECTION I Introduction

The system is a basic search engine for specified documents. Given by a file that contains many documents, the system should read the file, select different search ways for each line and output a result file for the search file. The system is required to use Boolean search, proximity search and phrase search. Except that, ranked information retrieval based on TFIDF is also required. The system should generate a rank file and calculate the scoring for every term.

The report mainly includes three parts: preprocessing, search and conclusion. The preprocessing part will describe the details of every step during preprocessing. The second part introduce the different methods for three search and the library used in development. The last past will come up the conclusion that the optimization of the system, the innovations of improving performance and the evaluation.

SECTION II Preprocessing

In this section, the author introduce the details of every step in preprocessing. At the beginning , the system is required to read content from xml file. Developer can import $xml.dom.minidom$ library to read the xml file and use the functions of $parse()$ to get the whole file. By using $getElementsByTagName()$, developer can get the document's id list and headline list and text list.

According to requirements, developer need to merge headline into document content. Developer can start a loop and iterate every document and add the headline into the text list.

2.1Tokenisation

In tokenization, developer need to remove all punctuations in documents. The developer can define a regular expression to solve this problem. Considering that one document is consist of punctuations, words and numbers, developer can define a regular expression $[\hat{}a-zA-Z0-9]+$ which can match all characters except letters from a or A to z or Z and numbers from 0 to 9.

After removing punctuations, developer need to change all upper words into lower words. In python, there is a function $Lower()$ provided. Developer can write a loop and iterate words in document. For every word , use this function to lower.

2.2 Stopping

After tokenization, developer need to remove all stop words from document. The English common stop words txt has been provided. Developer can read this file by using $with\ open$ clause and define a list to contain stop words. Then define a loop to iterate every word in document and verify whether this word is in stop words list. If the iteration is in stop words list, remove this word from document. At the same time, developer can define another list named words to contain all preprocessed words in document to prepare for the next steps.

2.3 Stemming

In this step, developer import a library named $NLTK$ to stem document. The $NLTK$ provides a function $PorterStemmer().stem()$ to stem words. Developer can do the stemming in the last steps' loop. For every word that has not been removed in stopping, developer can stem the word and add it into document again. Then developer can append the word into the words list that was defined in the last step. Developer can write this words list into a json file for getting a better search. Python provides a function $json.dump(list, file)$ to write a list into json file.

2.4 Summary

During the preprocessing, the developer should pack the whole process as a function. The required parameters are xml filename and the element tag name that user wants to get from xml file. The function should return the preprocessed xml content and document id list.

SECTION III Search

3.1 Boolean Search

In Boolean search module, developer need to generate a matrix to store the Boolean value of every term in different documents. Developer can import a library named $pandas$ to generate matrix. The $pandas$ provides a new data struct named $dataframe$ to store matrix. Developer can create a $dataframe$. Rows are document id and columns are single character in term. The author started a loop and iterate every word in every document and store Boolean value in matrix.

After generating the matrix, developer need to solve the problem about how to read the term calculate the matrix word by word. The author created a stack class and used stack to store the term. By using the stack, the author can differ the operator "AND " or "OR" or " NOT". When the stack pops the operator, developer can calculate the matrix and store the temporary result in a new column. When the stack pops all the elements, the new column

stores final result. Developer can iterate the new column and store all the id in a list which value is equal to “1”.

3.2 Proximity/Phrase Search

Firstly, developer need to generate an inverted list. Developer need to iterate all the documents and create a dictionary for every word. The key is word and value is another dictionary to store document id and position. In this loop, developer store all the words position information into dictionary. Then developer can write the dictionary into json file by using $json.dump()$ in order to make search faster.

Secondly, when user use proximity or phrase search, developer can split the term and find the position list for every character, Then iterate both list and find the public document id that they contain. Developer can calculate the absolute value of two positions in public document. If the absolute value is equal to the distance, developer can store the public document id in a list. When finish the loop and find all the public document id, the list is the final result.

3.3 Ranked IR based on TFIDF

In this part, developer can use the inverted list generated in proximity search. Firstly, developer can split term and count the value of DF and TF for every single character by reading the inverted list.

By using the TFIDF formula, developer can calculate the rank for every document. Then define a dictionary to store the document id and rank score. After that, developer use the $sorted()$ function and use the $lambda$ to sort the dictionary from top score to bottom score. According to course work requirement, developer needs to write the list into one txt file.

SECTION IV Conclusion

4.1 Optimization

Problem 1: The system will take too much time and resources in Boolean search.
Optimization: Instead of using matrix to calculate search result, developer could operate set based on inverted list by using intersection set, union set and difference set. Developers do not need to create a matrix file and fill the value for the huge dimensions matrix by using this way. Developer can just save the useful documents' id and save a lot of time and resources.

Problem 2: It is hard to differentiate which search way should we use for an input string.
Optimization: Developer can use the stack to solve this problem. In python, there is no packaged stack data class provided. The author writes a class to complete the function of

stack. By using the stack, developer could store every character of string. When stack pop up '#', developers could use phrase search. When stack pop up 'AND' or 'OR' or 'NOT', make it use Boolean search.

4.2 Idea

4.2.1 Developer could make the system faster by using some parallel techniques. For example, developers could use the function of $multiprocessing.Pool()$ in python.

4.2.2 When the collection of documents are changed slightly, it is unnecessary to recreate the inverted file. What should developers do is to detect which part has changed and add/delete relevant parts and id.

4.2.3 Developers could immigrate the system into Linux OS to build and detect it automatically. Developers can write a script to detect whether there are some changed in collections of documents and whether there is some input string which can cause risks. In Linux, developers could use crontab command to make this script run automatically everyday.

4.3 System Assessment

The search system can satisfy the demand of boolean search, proximity search and phrase search. After generating all the relevant index files, the whole time for one search will be less than 30 seconds. Beside search, the system can generate a ranked file by using IFTDF. The system is strict for input string's format and we do not write an error file to deal with the possible bugs by the input string.

Overall, the system is satisfied with the requirement of course work basically.