CSE272 Homework 1 Report

Changmao Li

1 Software

1.1 High-level software architecture

Figure 1 shows the high-level software architecture. The program is written in Python which includes the entry main code, data processor code, index builder code and index searcher code. The data processor code includes many utility functions to read the data (such as documents) into computer memory, or to convert the data to the required format(such as fitting for the usage of index builder), or to output results to the required evaluation format(such as converting results to the trec format). The index builder code is to create inverted index for documents using Apache Lucene which will formulate into a directory storage in the file system. The index searcher code is to search the index by searching algorithms. The entry main code is to start the program by giving the program arguments to set tasks or algorithms and other properties to run.

1.2 Major data structures

For documents and queries data which are read into computer memory, we use Python Dictionary to contain each document or query since it includes key and value pairs and uses hash table to implement so that we can easily store them and fast access them. Also, we use Python List to contain all the documents and queries for building of index next.

1.3 Any programming tools or libraries that you used

The program is based on the python version of Apache Lucene which is pylucene. To deal with text we also use NLTK to tokenize text and remove stopwords.

1.4 Strengths and weaknesses of your design

Strengths: 1. It is structural and has very clear work flow. 2. it is very easy to add new traditional searching algorithm in it. i.e. Just to add codes in Index Searcher.
Weaknesses: 1. It cannot deal with very large data since all current settings are making data read into the memory. 2. It is not easy to add recent machine learning or neural methods since these parts need to be trained.

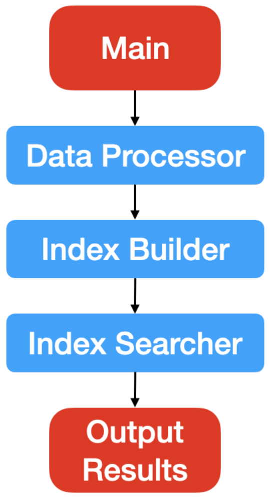

Figure 1: *High-level software architecture.*

1.5 Any problems that your system encountered

1. It is very hard to build the pylucence environment since the Lucence is originally not for Python, it is so complex and has many dependencies and after building it, I cannot even remember the specific steps to build it.
2. The IDE for python I used is pycharm which does not show the code hint for the pylucence so it was even very very hard to search the lucence documents and develop the algorithms based on it since we need to access the index and deal with them but lucence documents are in Java.

2 My new algorithm: Multi-pass Pseudo Relevance Feedback

I developed an algorithm which is **BM25 with Multi-pass Pseudo Relevance Feedback** which does not use the external data as Relevance Feedback and can have multiple times queries expansion. BM25 is a provided function in the Apache Lucene so here we mainly discuss about my implementation of the new proposed Multi-pass Pseudo Relevance Feedback.

Here we have several passes of retrieve before the final retrieve, the first pass is to retrieve the number of k documents based on the original query using BM25 and then we apply a designed term ranking algorithm to select the m highest scored terms and use these terms to expand the query. The k_1 and m_1 needs to be tuned to get the best results. The designed term ranking algorithm is the following: Assume the first retrieved documents collection is D_1, for each document d_1 in D_1, the TF-IDF score s^t_{tfidf} of each term t in d_1 is computed. Note that when computing the IDF we only consider the documents in D_1 instead of all documents. After computation the s^t_{tf-idf} of all terms in D_1, their scores are ranked and then we pick m_1 highest scored terms to expand the query. After expanding the query, we again use BM25 to do the second retrieve to retrieve the number of k_2 documents based on previous expanded query and as the pass one, we also used the designed term ranking algorithm to select the m_2 highest scored terms and use them to expanded query again. After n passes and obtaining the final expanded query, we use BM25 again to do the final retrieve to obtain the final results. If each pass we can make sure there are improvements then we can finally obtain a better and better results after tuning the k_i,m_i which i is the i th pass. In my experiments, we have 2 passes, for the first pass we set $k_1 = 5$ and m_1 = 2, for the second pass we set $k_2 = 9$ and $m_2 = 13$.

3 Experimental results

Table 1 shows all the experimental results. The results of Boolean, TF, TFIDF are surprisingly low while BM25 and my approach have very large improvement over these approaches. Interestingly, the relevance feedback using Rocchio algorithm cannot improve the results, that's possibly due to the number of feedback is not enough or the gold document content in the feedback cannot exactly consistent with the gold document content in the task. Our Multi-pass Pseudo Relevance Feedback algorithm even has large improvement over the traditional best performing model BM25 which intuitively has two reasons: 1. The algorithm is based on the good performing model BM25 so every time it can choose the good documents as pseudo feedback. 2. At each pass we tuned the parameters(the number of documents and the number of tokens extended to query) to get a better score, so in that case we can always make sure that it can give better results after several passes.

Besides above algorithms I also tried the directly use pretrained BERT embedding to compute similarity of the query and document but finally I cannot finish it because the number of documents is so large and the time will be taken hours or days to finish the selection which is unacceptable during an instant information retrieval system. I also tried to train a classification model from the given data to do learning to rank but the data has too imbalance distribution where approximately positive : negative is 1:5144, so it cannot be directly trained.

Approaches\Metrics	**MAP**	**P_50**	**recall_50**	**Time(s)**
Boolean	0.083	0.165	0.179	2.25
TF	0.047	0.104	0.126	3.66
TFIDF	0.077	0.161	0.179	2.42
RF	0.067	0.145	0.166	3.51
BM25	0.192	0.306	0.339	**2.23**
***MP-PRF*(1-pass)**	0.211	0.320	0.348	3.51
***MP-PRF*(2-pass)**	**0.221**	**0.335**	**0.368**	4.36

Table 1: Experiments Results. ***RF*** represents relevance feedback using Rocchio algorithm; ***MP-PRF*** represents Multi-pass Pseudo Relevance Feedback(My new algorithm)

4 Learned from this assignment

1. The usage of Apache Lucene and pylucene.
2. The implementation of many searching algorithms.
3. The usage of evaluation scripts(*trec_eval*) of the information retrieval.
3. The first experience to build a workable information retrieval system.

5 Software Github link

`https://github.com/arianakc/CSE272/tree/main/hw1`