Report for HW1: Search Engine

Zichao Li

2 de mayo de 2022

1. Software

1.1. Architecture

This assignment is finished by Python, including the data processor, index builder and index searcher. The data processor code finish most of utility functions to read the data (such as documents) into computer memory, or to convert the data to the required format(such as `[U+FB01]`tting 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 aims at creating inverted index for documents. And this program search the index with index search codes via searching algorithms.

1.2. Data Structures

We use a Python dictionary to contain each document or query because it contains key and value pairs and is implemented using a hash table so that we can easily store and quickly access them. We also use Python List to contain all the documents and queries for building of index next.

1.3. Tools or Libraries

The program is based on the python version of Apache Lucene which is pylucene. NLTK is used to tokenize text and remove stop words.

1.4. Strengths and weaknesses of your design

Strengths: The structure is easy to follow and add new algorithms.
Strengths: Difficult to add some deep learning methods that requires training.

1.5. Problems encountered

It is very difficult to build a pylucence environment because Lucence was not originally used for Python. It is very complex and has many dependencies.

2. My Designed Algorithm

I developed an algorithm which is MP-PRF-BM25 (BM25 with Multi-pass Pseudo Relevance Feedback). It does not use external data as relevant feedback and can be queried multiple times. BM25 is a provided function in the Apache Lucene.

Methods	MAP	recall_50	P_50	Times
Boolean	0.083	0.181	0.164	2.34
TF	0.046	0.123	0.110	3.45
TFIDF	0.079	0.178	0.153	3.56
RF	0.063	0.169	0.143	2.11
BM25	0.191	0.337	0.301	2.13
MP-PRF(1-pass)	0.213	0.348	0.321	3.12
MP-PRF(2-pass)	0.213	0.368	0.332	3.51

Before the final retrieval, we have several retrieval processes. The first process is to use BM25 to retrieve the number of K documents according to the original query, and then we use the designed term sorting algorithm to select the M terms with the highest score, and use these terms to expand the query. k_1 and m_1 need to be adjusted for best results. The designed term sorting algorithm is as follows: assuming that the document set retrieved for the first time is d_1, for each document in d_1, calculate the TF-IDF score s tfidftof of of each term t in d_1. After n passes and obtaining the final extended query, we use BM25 again for final retrieval to obtain the final result.

3. Experimental results

Table 1 shows all the experimental results. The results of Boolean, TF and TFIDF are surprisingly low, and BM25 and my method are much better than these methods. Relevance feedback using Rocchio algorithm can not improve the results, which may be due to insufficient feedback or the content of the gold document in the feedback is not completely consistent with that in the task. Our multi-pass pseudo relevance feedback algorithm is even much better than the traditional best performance model BM25. Intuitively, there are two reasons: 1. The algorithm is based on the BM25 model with good performance, so the document with good performance can be selected as pseudo feedback every time. 2. Each time we pass, we will adjust the parameters (the number of documents and the number of tokens extended to the query) to get better scores. Therefore, in this case, we can always ensure that it can give better results after multiple passes.

4. Learned from this assignment

1. The usage of Apache Lucene and pylucene. 2. The knowledge of many algorithms.

5. Github Link

https://github.com/zichaoli/272HW