Information Retrieval

COMP 479 Project 3 Report

Lin Ling (40153877)

A report submitted in partial fulfilment of the requirements of Comp479.

Concordia University

INDEX

1. Subproject I: Upgrade Project 2 Subproject 1 system

a. Source Code: upgrade_indexer.py

util.py

1.1 Compare timing of this SPIMI inspired procedure with the naïve indexer

(for 10000 term-docID parings)

To get the 10000 term-docID or token-docID, first use a function to find the first 10000 token will from document 1 to the middle of document 79. So, picking these data out and putting in extra directory named <*test_time_data*> for the comparing time between naïve indexer and upgraded indexer.

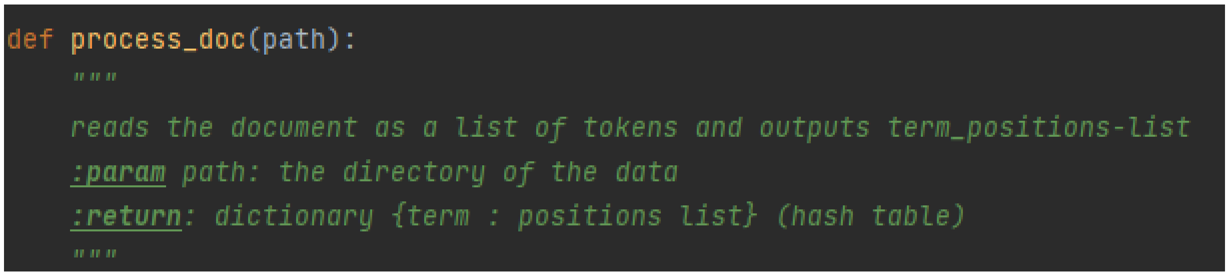
The results of the comparing for 10000 term-docID parings are as follows:

- Naïve indexer: 0.018971 seconds
- Upgrade indexer: 0.009973 seconds

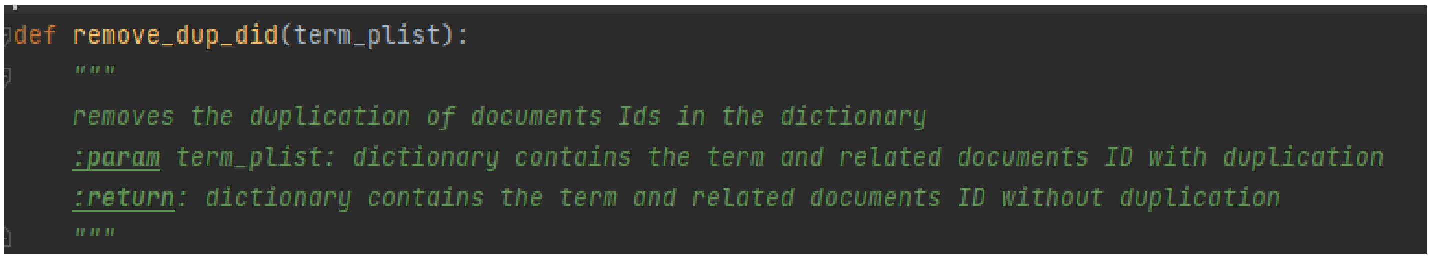
For larger corpus, the upgrade indexer supposed to be performed even better. As it generated the posting list with storing the term simultaneously, and without sort if setting several blocks for generating the upgrade indexer which inspired by SPIMI should be saved more times here.

1.2 compile an inverted index for Reuters21578 without using any compression techniques

For the upgrade indexer inspired by SPIMI, the procedure is reading documents file by file and stored the term associated with posting list in dictionary (hash table in python) for all the data by <*process_doc*> method and remove the duplicate docIDs in posting list and sorted it by <*remove_dup_sort_did*> method.



```
def process_doc(path):
    """
    reads the document as a list of tokens and outputs term_positions-list
    :param path: the directory of the data
    :return: dictionary {term : positions list} (hash table)
    """
```



```
def remove_dup_did(term_plist):
    """
    removes the duplication of documents Ids in the dictionary
    :param term_plist: dictionary contains the term and related documents ID with duplication
    :return: dictionary contains the term and related documents ID without duplication
    """
```

2. Subproject II: Convert indexer into a probabilistic search engine

2.1 Design four test queries

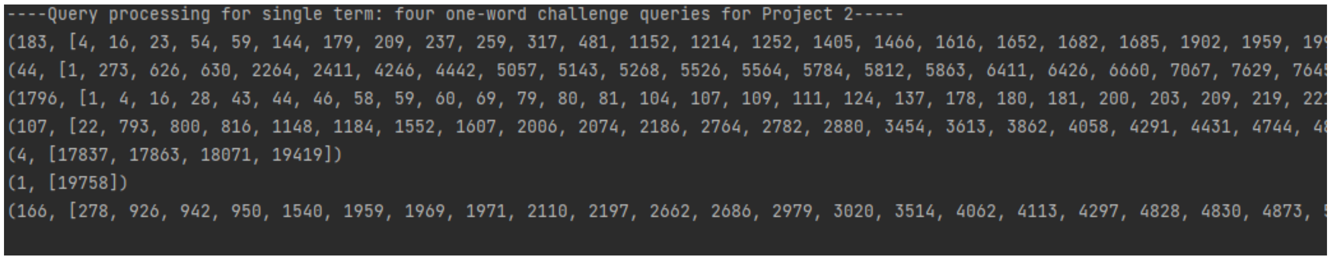
Source Code: probabilistic_search_engine.py

util.py

2.1.1 a single keyword query, to compare with Project 2

As finding of the comparing the results of a single keyword query, there are totally the same performance. The reason is although the generate process is different, the indexer which generated at last is the same.

The result of upgraded indexer:



```
----Query processing for single term: four one-word challenge queries for Project 2-----
(183, [4, 16, 23, 54, 59, 144, 179, 209, 237, 259, 317, 481, 1152, 1214, 1252, 1405, 1466, 1616, 1652, 1682, 1685, 1902, 1959, 19
(44, [1, 273, 626, 630, 2264, 2411, 4246, 4442, 5057, 5143, 5268, 5526, 5564, 5784, 5812, 5863, 6411, 6426, 6660, 7067, 7629, 7645
(1796, [1, 4, 16, 28, 43, 44, 46, 58, 59, 60, 69, 79, 80, 81, 104, 107, 109, 111, 124, 137, 178, 180, 181, 200, 203, 209, 219, 22
(107, [22, 793, 800, 816, 1148, 1184, 1552, 1607, 2006, 2074, 2186, 2764, 2782, 2880, 3454, 3613, 3862, 4058, 4291, 4431, 4744, 48
(4, [17837, 17863, 18071, 19419])
(1, [19758])
(166, [278, 926, 942, 950, 1540, 1959, 1969, 1971, 2110, 2197, 2662, 2686, 2979, 3020, 3514, 4062, 4113, 4297, 4828, 4830, 4873, 5
```

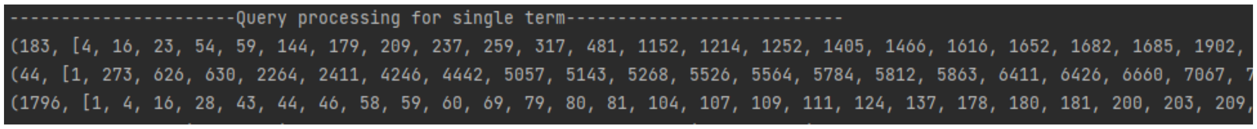
The result of naïve indexer in Project 2:

Three sample queries:

a. “outlook”: frequency :183

b. “zone”: frequency:44

c. “week”: frequency:1796



```
---------------------Query processing for single term--------------------------
(183, [4, 16, 23, 54, 59, 144, 179, 209, 237, 259, 317, 481, 1152, 1214, 1252, 1405, 1466, 1616, 1652, 1682, 1685, 1902,
(44, [1, 273, 626, 630, 2264, 2411, 4246, 4442, 5057, 5143, 5268, 5526, 5564, 5784, 5812, 5863, 6411, 6426, 6660, 7067, 7
(1796, [1, 4, 16, 28, 43, 44, 46, 58, 59, 60, 69, 79, 80, 81, 104, 107, 109, 111, 124, 137, 178, 180, 181, 200, 203, 209,
```

Given challenge queries results:

a. “copper”: frequency :107
b. “Samjens”: frequency:4
c. “Carmark”: frequency:1
d. “Bundesbank”: frequency:166:

2.1.2 a query consisting of several keywords for BM25

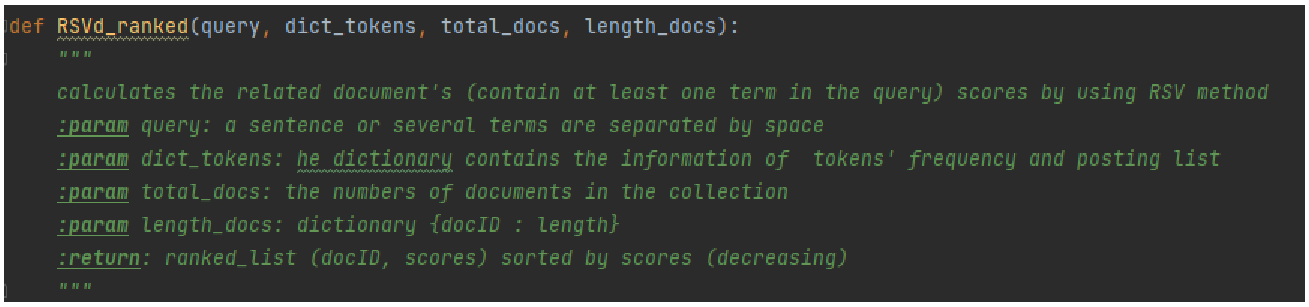
The formula used for BM25 is (11.33 formula in notebook) which is just setting similar weighting for query term if the query is long. This is appropriate if the queries are paragraph long information needs, but not unnecessary for short queries. So, I set k3 to 0 at this step (which will perform same as 11.32 formula in notebook), and I will keep the 11.33 formula for future need.

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{\mathrm{df}_t}\right] \cdot \frac{(k_1+1)\mathrm{tf}_{td}}{k_1((1-b)+b \times (L_d/L_{\mathrm{ave}}))+\mathrm{tf}_{td}} \tag{11.32}$$

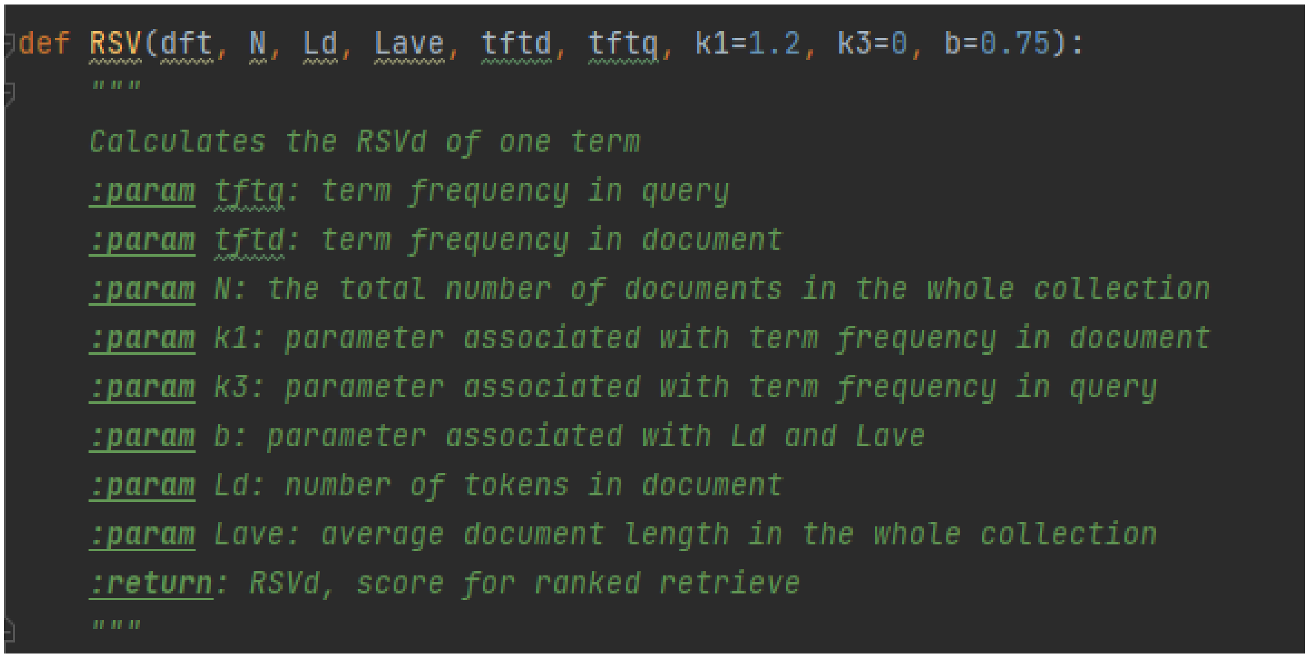
$$RSV_d = \sum_{t \in q} \left[\log \frac{N}{\mathrm{df}_t}\right] \cdot \frac{(k_1+1)\mathrm{tf}_{td}}{k_1((1-b)+b \times (L_d/L_{\mathrm{ave}}))+\mathrm{tf}_{td}} \cdot \frac{(k_3+1)\mathrm{tf}_{tq}}{k_3+\mathrm{tf}_{tq}} \tag{11.33}$$

The notebook claims that experiments have shown reasonable values are to set k1 to a value between 1.2 and 2 and b = 0.75. So, in this case, I set k1=1.2, b=0.75.

In this query processing, we should split the query to terms and gets the posting list of them and document frequency(dft), total number of documents in the collection(N), length of document (Ld), average length of all documents (Lave), term frequency of term t in the document d(tftd), term frequency of term t in the query q(tftq). And then calculating and accumulating the score RSV of the documents and sorted them by <*RSVd_ranked*> method and helper <*RSV*> method.



```python
def RSVd_ranked(query, dict_tokens, total_docs, length_docs):
    """
    calculates the related document's (contain at least one term in the query) scores by using RSV method
    :param query: a sentence or several terms are separated by space
    :param dict_tokens: he dictionary contains the information of  tokens' frequency and posting list
    :param total_docs: the numbers of documents in the collection
    :param length_docs: dictionary {docID : length}
    :return: ranked_list (docID, scores) sorted by scores (decreasing)
    """
```



```python
def RSV(dft, N, Ld, Lave, tftd, tftq, k1=1.2, k3=0, b=0.75):
    """
    Calculates the RSVd of one term
    :param tftq: term frequency in query
    :param tftd: term frequency in document
    :param N: the total number of documents in the whole collection
    :param k1: parameter associated with term frequency in document
    :param k3: parameter associated with term frequency in query
    :param b: parameter associated with Ld and Lave
    :param Ld: number of tokens in document
    :param Lave: average document length in the whole collection
    :return: RSVd, score for ranked retrieve
    """
```

The queries result as follows:

Note: 1. The ranked list will show the document ID associated with the RSVd score for the document by the certain query.

2.The documents with zero score by BM25 processing which mean there are not one term in the query exists in the document will not included in this ranked list.

(a) Democrats' welfare and healthcare reform policies

- ✓ length of the ranked list: 14206

- ✓ Top 10 of the ranked list:

  [(20449, 12.417187829637276), (7433, 11.518692832508446), (18722, 11.439618113679405), (6940, 11.39225130045293), (21577, 11.157969356614702), (18878, 11.007990778579508), (8072, 10.628517999912916), (4006, 10.522416531176622), (14976, 10.39015126606718), (18731, 10.354750303021671)]

(b) Drug company bankruptcies

- ✓ length of the ranked list :5313
- ✓ Top 10 of the ranked list:

  [(16771, 13.983440588911423), (4050, 12.587796813150131), (8209, 11.157355038278686), (16994, 10.045602224974205), (13764, 9.788452464062258), (9542, 9.437609277615124), (1805, 9.386781031742132), (6089, 9.37424448154082), (21239, 9.072692141678681), (18716, 8.917608223050786)]

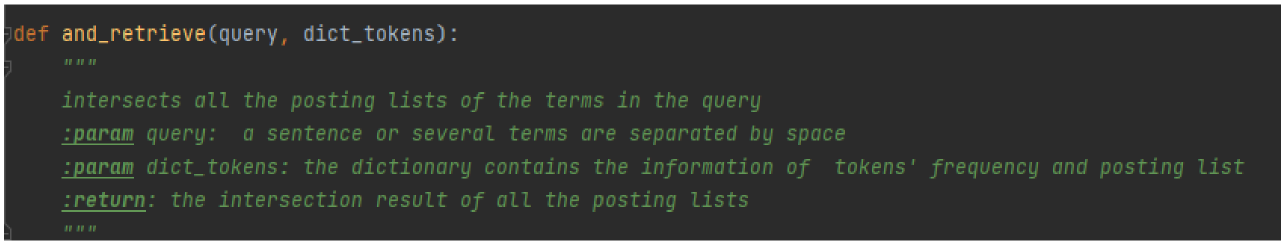
(c) George Bush

- ✓ length of the ranked list: 140
- ✓ Top 10 of the ranked list:

  [(20891, 14.066773941206687), (8593, 14.004584061951167), (4008, 11.490156593953971), (16780, 11.238406692097994), (20719, 10.039691560065283), (3560, 9.145677297037798), (16824, 8.781853173082755), (8500, 8.040006219421606), (7525, 8.020805820824556), (2711, 7.775388621950719)]

2.1.3 a multiple keyword query returning documents containing all the keywords (AND)

In this query processing, we should split the query to terms and gets the posting list of them and intersect the posting list, then return the result by <*and_retrieve*> method.



```
def and_retrieve(query, dict_tokens):
    """
    intersects all the posting lists of the terms in the query
    :param query:  a sentence or several terms are separated by space
    :param dict_tokens: the dictionary contains the information of  tokens' frequency and posting list
    :return: the intersection result of all the posting lists
    """
```

The queries result as follows:

The return list just shows the posting list which contains all the terms in the certain query.

(a) Democrats' welfare and healthcare reform policies

Empty list [] means that there is not existing a document including all the term in this query.

(b) Drug company bankruptcies

Empty list [] means that there is not existing a document including all the term in this query.

(c) George Bush

[7525, 965, 3560, 4008, 2796, 16780, 20719, 8593, 8500, 854, 20891, 20860, 5405]

2.1.4 a multiple keywords query returning documents containing at least one keyword (OR)

In this query processing, we should split the query to terms and gets the posting list of them and merges the posting list and ordered by how many keywords they contain, then return the result by *<or_retrieve>* method.

The queries result as follows:

The return list shows the posting list which means these documents contains at least one term in the certain query and they are in an order of the number of keywords they contain.

(a) Democrats' welfare and healthcare reform policies

- ✓ length of the or retrieve ordered list :14206
- ✓ Top 10 of the or retrieve ordered list:

[1895, 2132, 28, 1999, 9774, 15369, 17915, 18271, 18328, 18357]

(b) Drug company bankruptcies

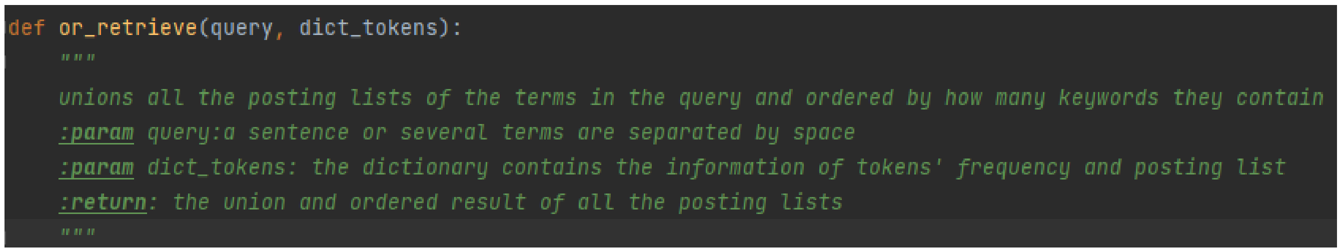
- ✓ length of the or retrieve ordered list 5313
- ✓ Top 10 of the or retrieve ordered list:

  [192, 696, 730, 1391, 1805, 1819, 2271, 2868, 3106, 3176]

(c) George Bush

- ✓ length of the or retrieve ordered list :140
- ✓ Top 10 of the or retrieve ordered list:

  [854, 965, 2796, 3560, 4008, 5405, 7525, 8500, 8593, 16780]



```
def or_retrieve(query, dict_tokens):
    """
    unions all the posting lists of the terms in the query and ordered by how many keywords they contain
    :param query:a sentence or several terms are separated by space
    :param dict_tokens: the dictionary contains the information of tokens' frequency and posting list
    :return: the union and ordered result of all the posting lists
    """
```

3. Conclusion

3.1 Analysis of the results of the sample runs

From the results, we could find it exists difference among BM25 ranked list, AND retrieve and OR retrieve for the same query. This could lead us to consider which result will be better achieved the customer' information needs. For the first query, if customer query "Democrats' welfare and healthcare reform policies", the result of BM25 will give you the shorter document which including some of the terms in query, the result of OR retrieve will give you the longer one as it does not consider the length of the documents and term frequency and just simple ordered in the number of terms they contained. Of course, the AND retrieve is very hard to get lots of results as should including all the terms in one document. That's why BM25 is one of the most widely used and robust retrieval models.

From the third query, "George Bush", so it's a name, should be consider have a AND operator between them from the information need of user. And you could find OR retrieve could have a better performance as it ordered by the number of terms they contained. Part of Top10 result in BM25 includes one term with a short length. Thus, it could combine Boolean retrieve with BM25 at this situation as should consider “George Bush” as one term to search will get better ranked result.