Web Search Engines and Information Retrieval

Report: Assignment 2

(Ranked retrieval and Advance information retrieval feature)

Submitted by:
Radhika Zawar S3734939
Parvi Verma S3744398

Introduction

In this report, we present Ranked Retrieval Implementation using okapi BM25 similarity function and Advance Information Retrieval Feature of Phrase-Search. This assignment heavily depends upon the implementation prototype of Assignment 1 where we presented Indexing and Querying. This Assignment re-uses and modifies most of the functionality implemented in Assignment 1. To implement okapi BM25 and Phrase-search feature, we have used various online sources to utilise optimised algorithms.

Python Version: Python 2.7.14

We have used in-built python libraries such as paramiko, pathlib, os, re, struct, time, heap, sys to implement various components.

Ranked Retrieval

Index Search

When queries are passed from command line to search r top ranked relevant documents, it reads the lexicon documents and map file which is created in indexing phase. These two files are stored in local memory. To make the search efficient, we have used a dictionary (hash map) and a list for lexicon and map respectively.

Algorithm

When the query term is received from command line, we read the term and normalize it using normalise() function from index.py file and store each term in a list. Then we check the similarity metric argument for BM25 and send the query

list along with arguments, lexicon file and map file to function_bm25(). Then we set the values for N which number of documents in collection that is length of map file and constant parameter as k1=1.2 and b=0.75 as mentioned. Then we split the query terms in sub-list and pass each term along with inverted list file location, lexicon file and map file to function seek_query_occurence(). Then we use the pointer term which is an offset to enter into invlists file. Invlists file has 32 bit data which we have packed in write_lexicon_invs() function in index.py file using struct.pack('>i',n) function from struct package.[1] Once we locate the occurrence of query term in lexicon and enter into invlists file using offset, we read binary data back into text form using struct.unpack('>i', f.read(4))[0] function. Once we find (32-bit) integer as a set of four bytes, which is document frequency, and this enables us to learn how much data to calculate for this query occurrence. Then if similarity metric is BM25 then retrieve the value for K which is calculated in indexing phase for each document and stored in map file using formula

K = k1 * ((1 − b) + (b * Ld) / AL)

Where k1=1.2 and b=0.75 are constants and AL is average document length and Ld is document length in bytes.

To calculate the partial accumulator for that particular document, calculateBM25() function with term frequency along with K and in-document frequency as parameters is called where using the Okapi BM25 formula

log((N - ft + 0.5) / (ft + 0.5)) * (((k1 + 1) * fdt) / (K + fdt))

where fdt is the number of occurrences of t in d, ft is the number of documents containing term t and N is the number of documents in the collection.

After we calculate the partial accumulator, we check for document id in the doc_score hash map which stores the document id and its BM25 score, if document id exists then we add the score to previous score otherwise we create a new entry with the score in hashmap.

Data Structure

In order to retrieve top r ranked relevant documents, we use min heap onto hashmap of accumulators for each document of particular query. Property of min heap is that each parent node should be smaller or equal to both of its children.

The data structure used for min heap implementation is a List which has the following property:

parent(i)=⌊i/2⌋

left-child(i)=2i

right-child(i)=2i +1

The size of minheap should be equal to the number of top ranked relevant documents which are to be retrieved. The idea behind using min heap is that the root node of min heap contains the minimum element of all the elements in heap that is why the root node of min heap is compared with the accumulator and if the value of accumulator is greater than root node then it is replaced with item at the root node and min heap is re-heapified. After which the content of sorted min heap is displayed in descending order of scores.

The one of the advantages of min heap is that it reduces the number of comparisons during sorting the accumulators and is much more efficient than full sort of accumulators. The second advantage of heap is that it only requires memory space proportional to r(number of top ranked documents).[2]

Advance Information Retrieval Feature

Phrase Search Implementation

To implement Advance Retrieval Feature, we chose Phrase Search as it was intuitive and it was also very similar to google's implementation of phrase search. When a query of multiple term is passed to Phrase-search, it picks up intersection set where both the query terms occur in the same order. To implement this we extended our indexing not only to record occurrences but also there locations in the document[4].

An advanced prototype to implement phrase search could be to use a combination of nextword indexes with inverted files as a solution to this problem[3]. However, this implementation was complex considering current coding skills.

Algorithm

In the proposed implementation of this assignment, we have extended our inverted index file to record term occurrence and its location in the document in

indexifier() function of index.py file. We have saved this location into inverted list file in write_lexicon_invs() function of index.py file. When query is passed to invoke search.py with the following command ./search --pharse-search -q <> -l <> -i <> -m <> <Query term>, function function_pharse_search() is invoked. Then we first find out all the documents which have the query terms using seek_query_occurence() function of search.py file and then in function function_pharse_search(), we take the intersection of that by using and operation of the document set of each query terms. Then we iterate over all the document to check if both the term location are in the sequence to one another in the same order. When we validate the output and present it. The implementation is explained in detail in the source code comments in search.py file

We have used list to record occurrences and term locations. The query_loc_list has implementation like [term_num -> {doc_id: [location]}] where we are storing all the intermediate results. We are using set data structure to identify common documents where all the query terms are present

For phrase search, we have recorder term location along with the term occurrence. The document collection latimes is originally of size 476M. The extended inverted list is now of size almost 314M. In assignment 1, the presented inverted list was of size almost 180M. It suggests that to incorporate trem location, the inverted lists takes almost thrice the original size. We we had implemented a combination of combination of nextword indexes with inverted files, then size would have been almost twice of 314M.

Effectiveness Evaluation

For effectiveness evaluation, we have used the following queries which are present in topics file on /home/inforet/a2 path

401 foreign minorities germany
402 behavioral genetics
403 osteoporosis
405 cosmic events
408 tropical storms

1. By running above queries using BM25 and phrase search retrieval approach we have gathered following results.

401 foreign minorities germany

BM25 Precision at 10 = (1/10)	Phrase Search Precision at 10 = (0/0)
la101790-0075 13.3861 0 la021890-0100 12.844 0 la100890-0131 12.283 1 la050690-0109 12.2473 0 la040789-0015 12.0197 0 la021490-0049 11.9212 0 la031590-0102 11.7885 0 la111089-0188 11.7253 0 la071890-0073 11.6982 0 la020789-0133 11.5928 0 la050790-0042 11.5473 0 la060890-0011 11.5188 0 la082789-0152 11.4392 0 la021190-0168 11.3931 0 la040590-0157 11.2477 0 la062290-0172 11.2186 0 la030990-0189 11.2132 0 la050990-0043 11.2126 0 la050390-0176 11.2089 0 la112189-0066 11.1544 0	None

402 behavioral genetics

BM25 Precision at 10 = (1/10)	Phrase Search Precision at 10 = (1/1)
la101290-0115 20.2379 1 la052290-0110 14.0733 0 la020389-0077 13.3808 0 la121289-0055 13.1943 0 la082590-0108 12.6448 0 la080190-0099 12.1113 0 la042990-0032 11.3407 1 la020789-0112 11.1128 0 la071689-0143 11.0381 0 la110889-0005 10.7825 0 la071489-0085 10.6958 0	la101290-0115 1 1

la042390-0048 10.5863 0 la021290-0061 10.5037 0 la030289-0084 10.4791 0 la012589-0063 10.4673 0 la051689-0102 10.402 0 la040790-0127 10.3971 0 la060289-0090 10.3203 0 la020789-0113 10.1997 0 la051389-0010 10.1589 0	

403 osteoporosis

BM25 Precision at 10 = (6/10)	Phrase Search Precision at 10 = (6/10)
la030689-0082 15.7548 0 la071290-0133 15.6328 1 la083089-0024 14.953 0 la020490-0136 14.7091 1 la011389-0029 14.4912 1 la010790-0103 14.1759 1 la051490-0120 13.0018 0 la032290-0151 11.943 1 la120689-0083 11.7318 0 la010390-0067 11.6095 1 la042589-0052 11.1688 0 la111589-0004 11.1662 0 la041990-0072 10.9128 1 la042189-0027 10.2747 0 la051889-0006 10.1207 1 la082390-0094 9.8372 0 la052290-0110 9.5891 1 la022790-0099 9.5656 0 la012990-0041 9.2837 0 la080190-0135 9.2179 0	la071290-0133 17 1 la020490-0136 13 1 la051889-0006 6 1 la030689-0082 5 0 la011389-0029 5 1 la111589-0004 3 0 la051490-0120 3 0 la010790-0103 3 1 la033089-0013 3 1 la082890-0074 3 0 la042189-0027 3 0 la083089-0024 3 0 la041990-0072 2 1 la052290-0110 2 1 la072289-0144 2 1 la080289-0067 2 0 la110890-0217 2 0 la110490-0091 1 0 la101890-0267 1 0 la022790-0099 1 0 la032489-0093 1 1 la120390-0005 1 0 la092890-0067 1 1 la010390-0067 1 1

	la032290-0151 1 1 la010689-0040 1 0 la042690-0002 1 0 la120490-0022 1 0 la012990-0041 1 0 la033089-0019 1 0 la111989-0048 1 0 la050290-0050 1 0 la111689-0032 1 0 la020990-0100 1 0 la042689-0065 1 0 la110189-0018 1 0 la120490-0023 1 0 la010490-0218 1 0 la021590-0062 1 0 la042589-0052 1 0 la011289-0149 1 0 la080190-0135 1 0 la082390-0094 1 0 la120689-0083 1 0

405 cosmic events

BM25 Precision at 10 = (2/10)	Phrase Search Precision at 10 = (0/0)
la012289-0002 14.6117 0 la010889-0109 12.1843 0 la063089-0071 11.9564 0 la031290-0034 11.9331 0 la021690-0057 11.3599 0 la042089-0083 11.1792 1 la020190-0053 10.4749 1 la121690-0039 10.4747 0 la061490-0089 10.4439 0 la122989-0137 10.2095 0 la121589-0173 10.1162 0 la031389-0056 10.1123 0 la111789-0134 10.0771 0	None

la081190-0002 9.8662 0 la090190-0129 9.7745 0 la100690-0017 9.6544 0 la050990-0163 9.3896 0 la091789-0029 9.3828 0 la052090-0005 9.279 0 la010790-0016 9.224 0	

408 tropical storms

BM25 Precision at 10 = (4/10)	Phrase search Precision at 10 = (1/10)
la101490-0142 20.2108 1 la062290-0070 18.2766 0 la101390-0102 17.6238 1 la101289-0148 17.0259 1 la103190-0052 16.2199 0 la091989-0049 16.1607 0 la120389-0130 15.912 1 la021690-0167 15.525 0 la030990-0199 14.4539 0 la092089-0027 12.2958 0 la110390-0071 11.7541 0 la082189-0033 10.9568 0 la102189-0071 10.8086 0 la030989-0189 10.8064 0 la051190-0106 10.6741 0 la040289-0192 10.655 0 la060689-0099 10.625 0 la020289-0156 10.5446 0 la050489-0061 10.4861 0 la101489-0074 10.333 0	la101490-0142 1 1 la030990-0199 1 0 la022790-0002 1 0 la080890-0116 1 0 la050490-0057 1 0 la062290-0070 1 0 la082890-0051 1 0 la101589-0082 1 0 la071490-0129 1 0

2. Precision at 10

The obtained results after the evaluation of effectiveness
To calculate precision we used the following formula:

Precision = (Correct answer/ Total Answer)

Where Total answer we consider are top 10

We get following results

Query	BM25	Phrase search
401 foreign minorities germany	0.1	0
402 behavioral genetics	0.2	1
403 osteoporosis	0.6	0.6
405 cosmic events	0.2	0
408 tropical storms	0.4	0.1

It is clear that BM25 is performing better that Phrase search for all the queries, only once it actually gives the same result to phrase search.

At the outset, we can say that BM25 outperforms Phrase search; however, it would not be accurate to say that.

Firstly, here we are dealing with only top 10 documents and secondly, we also need to consider basic concept on Phrase retrieval that it only provide list of documents where the phrase is actually present. Wherein, BM25 provides all the documents where the terms are present using weighted score in highest first order. Thus, comparing two features on top 10 result set may not be correct.

3. Proposal

We may consider including more top results to calculate the precision of the results. This might also bring upon those documents where term is actually present in the same phrase format.

For example : On the search of query term “stanford university” we get following results

BM25	Phrase Search
10 la011890-0090 1 13.9686 10 la111190-0200 2 13.6754	100 la060390-0080 3 100 la082489-0131 3

10 la101190-0234 3 13.5605 10 la121490-0026 4 13.5161 10 la051590-0090 5 13.3733 10 la072689-0029 6 13.2562 10 la101789-0005 7 13.2498 10 la012590-0221 8 13.1123 10 la082589-0006 9 13.014 10 la052990-0076 10 12.8317	100 la092390-0208 3 100 la092790-0044 2 100 la103089-0095 2 100 la062789-0066 2 100 la060990-0125 2 100 la061089-0088 2 ...

BM25 has retrieved all the documents where term stanford or university has occurred, obviously considering the score. But Phrase search has got us those documents where bot term occurred together as stanford university. Now this totally depends on user requirements if she is looking only for university or place stanford in general or the specific university. At the same time it is possible that whenever stanford is used in a particular document, it is nothing but referring to the same university

Also, we can implement the suggested solution on the research paper [3] published by researchers at RMIT University but considering current skillset, it was a little ambitious to implement.

References:

[1] https://docs.python.org/3/library/struct.html (online resources), last visit 20/08/2019

[2] Class notes for ISYS 1078/1079 by Dr Falk Scholer

[3] Efficient Phrase Querying with an Auxiliary Index - by Dirk Bahle, Hugh E. Williams, Justin Zobel, RMIT - Research Paper - online resource - https://dl.acm.org/citation.cfm?id=564415 (pdf link available), last visit 8/10/2019

[4]Phrase search algorithm implementation- http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.html&r=1&f=G&l=50&s1=%20060031195%22.PGNR.&OS=DN/20060031195&RS=DN/20060031195, last visit 07/08/2019