SEARCH ENGINE
Boolean Retrieval

Abstract—**This project includes implementation of search engine on CRAN dataset and optimizing the weights of each zone so that we get the best performance IR system.**

I. Introduction

The goal of this project is build a search tool (a small retrieval system) using Boolean retrieval method for cranfield dataset. and apply various queries with conjunction, disjunction and negation of terms. Here we also have to see the effectiveness of our implementation using MAP and Precision Recall graph for 1.without stemming 2.with stemming 3. zone indexes for title and body text fields and come up with optimal weights.

August 13, 2021

II. Theory

A. Need of Boolean Retrieval

The amount of online data increased almost as quickly as the speed of computers, so we now want the ability to search collections that measure in the billions to trillions of words. By indexing the documents, we can avoid linearly looking up the texts for each query.

B. Tokenization

Tokenization involves breaking up a document into pieces, called tokens[1], and perhaps removing certain characters, like punctuation, as part of the process.

C. inverted index

We assume each document in a collection has a unique document identifier (docID). When we construct the index, we simply assign successive integers to every new document and thus we create inverted index, it is like the index which is at back of our book.It makes the information retrieval fast as now we don't deal with sparse matrix but rather posting list [1] which is quite small in size in comparison to the matrix.

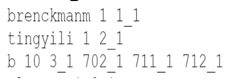
III. Procedure

A. Search Engine

(1)Our dataset CRAN has several fields like 1)Title, 2)Author, 3)info_abt_paper, 4)text_block. First we created Posting List for each zone/field. Now in my Posting list along with term-docID, I have also maintained the frequency of each term in each document as frequency can tell how important each term is with respect to a given document. Like if the term occurs as often as always then it is probably not that much relevant in differentiating documents and if the term appears very rarely like('UFO') then it is probably not a characteristic of that document rather mere coincidence so less important.
So our Posting-List looks like:-

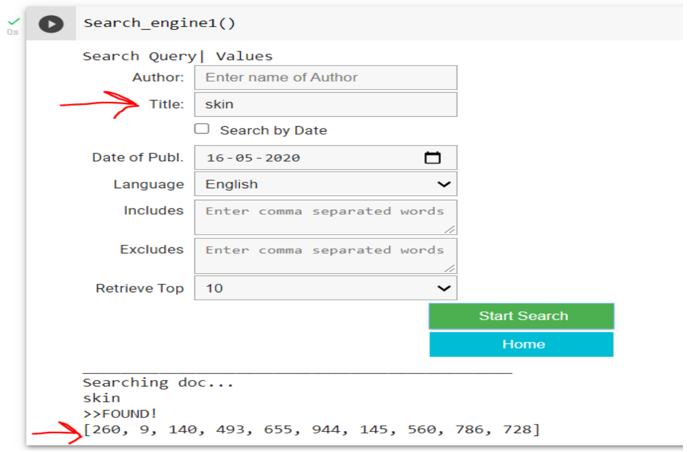
term	Total_Freq	ID_freq	ID_{freq}
water	29	24_1	28_2
lake	31	35_3	64_12

For example here I have attached a screenshot of my Posting_List for Author Field



```
brenckmanm 1 1_1
tingyili 1 2_1
b 10 3_1 702_1 711_1 712_1
```

(2)While creating this posting list I removed all the stopwords and applied stemming (3) My final search engine looks like this.

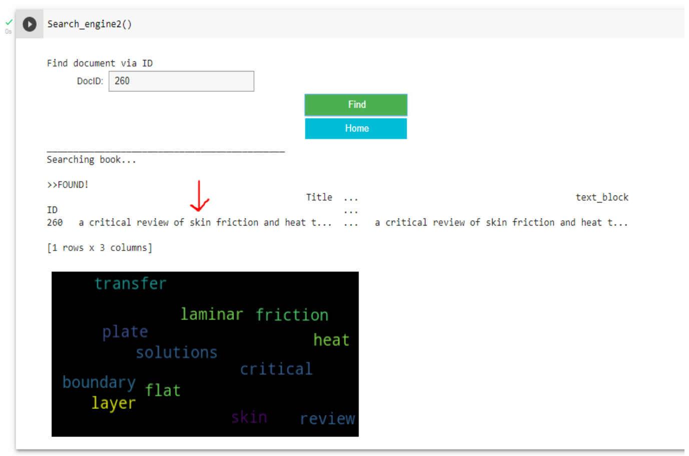


```
Search_engine1()
```

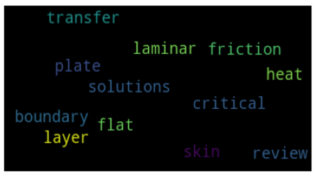
Search Query|	Values
Author:	Enter name of Author
Title:	skin
	☐ Search by Date
Date of Publ.	16-05-2020
Language	English
Includes	Enter comma separated words
Excludes	Enter comma separated words
Retrieve Top	10

```
Searching doc...
skin
>>FOUND!
[260, 9, 140, 493, 655, 944, 145, 560, 786, 728]
```

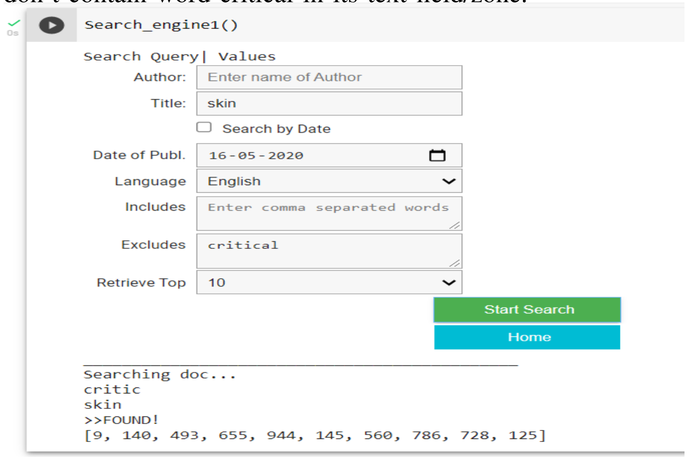
I have searched for docs whose title contains word skin, and I have found list of most relevent docs [260, 9, 140, 493, 655, 944, 145, 560, 786, 728]. Now let's us see what is docID 260.



```
Search_engine2()

Find document via ID
      DocID: 260
                                        Find
                                        Home
____________________________________________
Searching book...

>>FOUND!
                                  Title  ...                                  text_block
ID                                       ...
260   a critical review of skin friction and heat t...  ...   a critical review of skin friction and heat t...

[1 rows x 3 columns]
```



And we see that indeed word skin is in the title field. Let's refine our query, we also see there the wordcloud for this doc and it contains word critical, so this time we will search all those docs which contains word skin in its title field but don't contain word critical in its text field/zone.

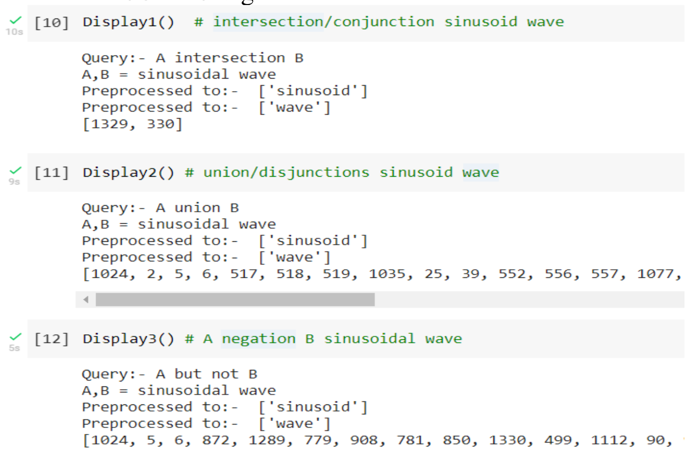


```
Search_engine1()

Search Query| Values
      Author:  Enter name of Author
       Title:  skin
               ☐ Search by Date
 Date of Publ. 16-05-2020
    Language   English
    Includes   Enter comma separated words
    Excludes   critical
 Retrieve Top  10
                                  Start Search
                                  Home
____________________________________________
Searching doc...
critic
skin
>>FOUND!
[9, 140, 493, 655, 944, 145, 560, 786, 728, 125]
```

And this time retrieved docs are [9, 140, 493, 655, 944, 145, 560, 786, 728, 125]. Thus we verified our search engine.

B. Intersection/Union/Negation

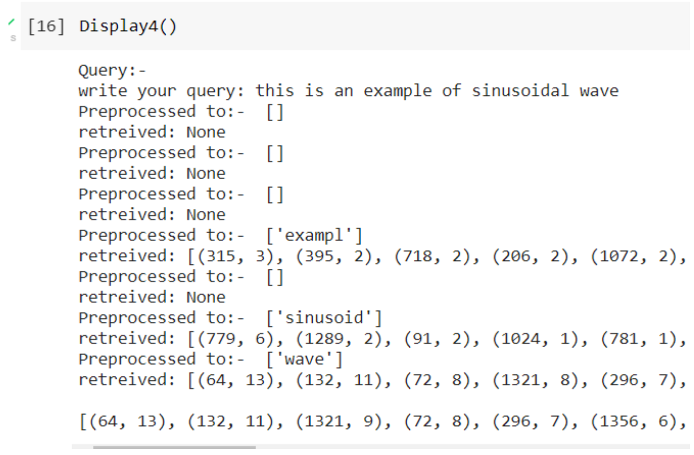
Here first we retrieved all the relevant docs for each of the term of the query and then we used python set operations .union(), .intersection() and - to find desired result. Intersection/Union/Negation:-



```
[10] Display1()  # intersection/conjunction sinusoid wave

     Query:- A intersection B
     A,B = sinusoidal wave
     Preprocessed to:-  ['sinusoid']
     Preprocessed to:-  ['wave']
     [1329, 330]

[11] Display2() # union/disjunctions sinusoid wave

     Query:- A union B
     A,B = sinusoidal wave
     Preprocessed to:-  ['sinusoid']
     Preprocessed to:-  ['wave']
     [1024, 2, 5, 6, 517, 518, 519, 1035, 25, 39, 552, 556, 557, 1077,

[12] Display3() # A negation B sinusoidal wave

     Query:- A but not B
     A,B = sinusoidal wave
     Preprocessed to:-  ['sinusoid']
     Preprocessed to:-  ['wave']
     [1024, 5, 6, 872, 1289, 779, 908, 781, 850, 1330, 499, 1112, 90,
```

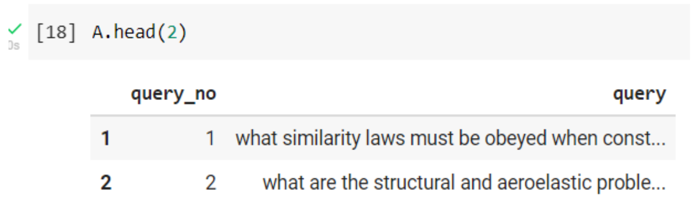
C. Ranked Retrieval with zone scoring

1) Retrieve Documents Rank based on freq of term in doc: This time we not only retrieved docs by intersection/union etc but rather we retrieved them by their frequency. As we can see docId 64 comes first as it is has highest frequency.



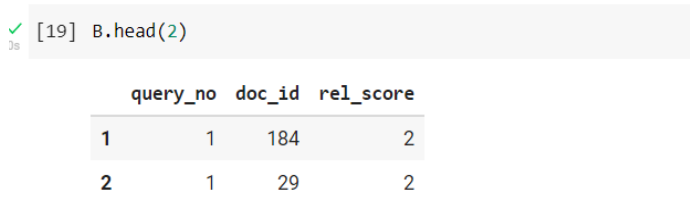
```
[16] Display4()

     Query:-
     write your query: this is an example of sinusoidal wave
     Preprocessed to:-  []
     retreived: None
     Preprocessed to:-  []
     retreived: None
     Preprocessed to:-  []
     retreived: None
     Preprocessed to:-  ['exampl']
     retreived: [(315, 3), (395, 2), (718, 2), (206, 2), (1072, 2),
     Preprocessed to:-  []
     retreived: None
     Preprocessed to:-  ['sinusoid']
     retreived: [(779, 6), (1289, 2), (91, 2), (1024, 1), (781, 1),
     Preprocessed to:-  ['wave']
     retreived: [(64, 13), (132, 11), (72, 8), (1321, 8), (296, 7),

     [(64, 13), (132, 11), (1321, 9), (72, 8), (296, 7), (1356, 6),
```

2) Zone Relevance score: We have two files as cran.qry and cranqrel which contains queries and relevances of documents with respect to each query, so since thers are small files so I have read them and converted to nice pandas dataframe.



```
[18] A.head(2)
```

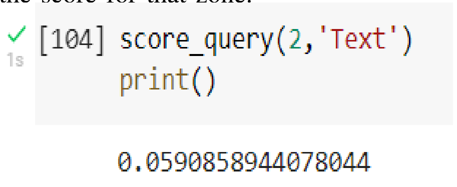
	query_no	query
1	1	what similarity laws must be obeyed when const...
2	2	what are the structural and aeroelastic proble...



```
[19] B.head(2)
```

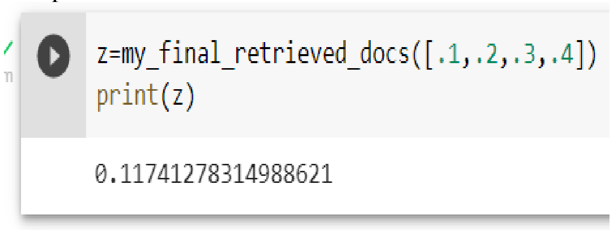
	query_no	doc_id	rel_score
1	1	184	2
2	1	29	2

Now we created a function in which we simply pass the query_no and the zone and it compares the MAP and returns the score for that zone.



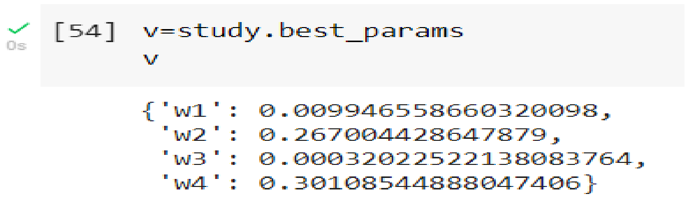
```
[104] score_query(2,'Text')
      print()

      0.0590858944078044
```

3) Use ML to find optimal weights: Next we created my_final_retrieved_docs([w1,w2,w3,w4]) function which takes parameter as list of weights of fields like Title Author Info Text and checks the score for each query one by one and return mean of mean squared error for each query, here our target label is 1 if doc is actually relevent and target label is 0 if doc is not actually relevent(this information is already given in our training dataset) and based on some weights we get/predict some value between 0 and 1 for each query doc pair. Then we computer the mean squared error from the true and predicted value.



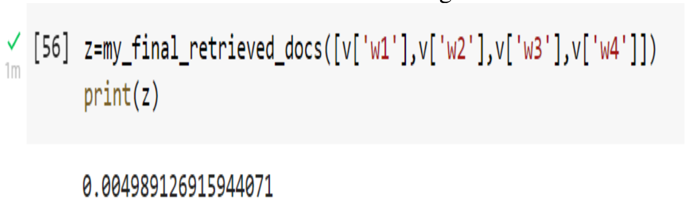
```
z=my_final_retrieved_docs([.1,.2,.3,.4])
print(z)

0.11741278314988621
```

Now we have to find optimal weights which will give us the lowest mean squared error. So for this I used Optuna[2] hyperparameter optimization technique and ran it for 50 trials. and we got our optimal parameters as:-



```
[54] v=study.best_params
     v

{'w1': 0.009946558660320098,
 'w2': 0.267004428647879,
 'w3': 0.00032022522138083764,
 'w4': 0.30108544888047406}
```

Now let's see how much error now we get.

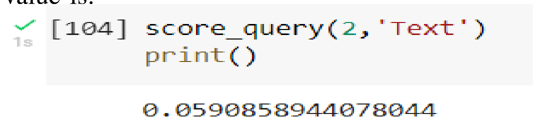


```
[56] z=my_final_retrieved_docs([v['w1'],v['w2'],v['w3'],v['w4']])
     print(z)

0.004989126915944071
```

0.0049 which is an improvement on our initial value of 0.117

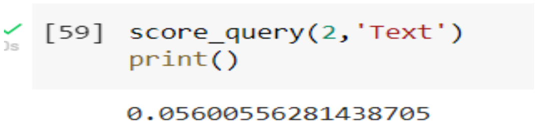
D. Comparing MAP and precision recall with and without stemming

Next we just commented all the porter.stem(word) everywhere in our document starting from creating our Posting lists and we will compare the MAP values. With stemming MAP value is:-



```
[104] score_query(2,'Text')
      print()

0.0590858944078044
```

Without stemming MAP value is:-



```
[59] score_query(2,'Text')
     print()

0.05600556281438705
```

Thus we see that stemming increased performance of our retrieval system.

IV. Conclusion

Boolean retrieval is the most crucial step in Information retrieval system which acts as the building block of information retrieval. Without having grasp of boolean retrieval we can't visualize, how more complex systems like vector representation and latent semantic index will work. This was a quite interesting project and I learned a lot during making of this search engine and running all the queries.

References

[1] Introduction to Information Retrieval, C.D. Manning, P. Raghavan and H. Schutze, Cambridge University Press, 2009

[2] Optuna: A Next-generation Hyperparameter Optimization Framework