INFORMATION RETRIEVAL

SIMPLE SEARCH ENGINE

<u>Goal:</u>

Create a Search engine with process of term generation, creating Inverted Index for generated tokens and querying the search engine with Boolean or vector processing algorithm by using stored Inverted Index followed by calculating NDCG for set of random queries, Average NDCG for both vector and Boolean and output Average NDCG and p-value using t_test and Wilcoxon test

<u>Technical details:</u>

Project was done on python3 so commands need to be executed as below if default version in the system is not Python3.If default version is python3 then run commands with python.

<u>Setup:</u>

<u>Install nltk:</u>

```
sudo pip install -U nltk
```

<u>Install nltk corpus with below commands:</u>

```
Python3
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download()
```

<u>Install scipy as below:</u>

```
pip install scipy
```

<u>Execute commands:</u>

1. **<u>creating Index file:</u>**

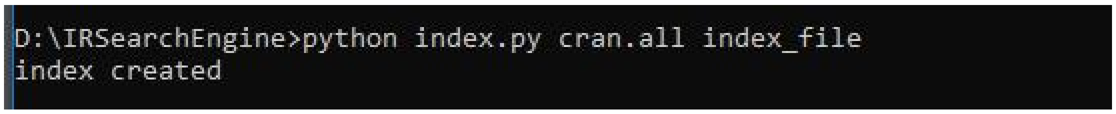
Execute below command as is.

python3 index.py cran.all index_file

or

python index.py cran.all index_file

Note: Executing this for entire **“cran.all”** takes lot of time, readymade **“Index_file”** been provided for further execution.



```
D:\IRSearchEngine>python index.py cran.all index_file
index created
```

2. Query Processing:

This below command returns result for randomly generated query

python3 query.py index_file processing_algorithm query.text or

python query.py index_file processing_algorithm query.text

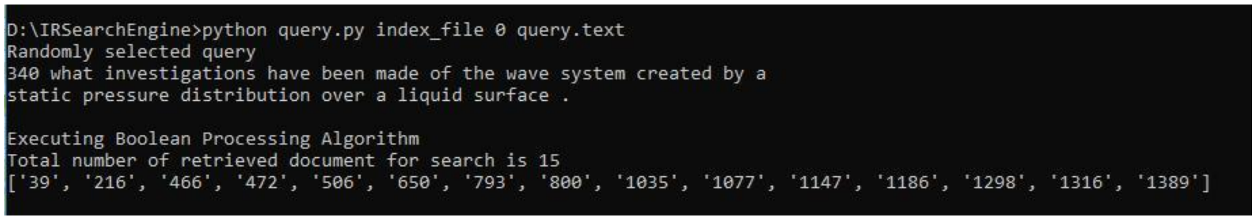
Example:

Boolean:

python3 query.py index_file 0 query.text

or

python query.py index_file 0 query.text



```
D:\IRSearchEngine>python query.py index_file 0 query.text
Randomly selected query
340 what investigations have been made of the wave system created by a
static pressure distribution over a liquid surface .

Executing Boolean Processing Algorithm
Total number of retrieved document for search is 15
['39', '216', '466', '472', '506', '650', '793', '800', '1035', '1077', '1147', '1186', '1298', '1316', '1389']
```

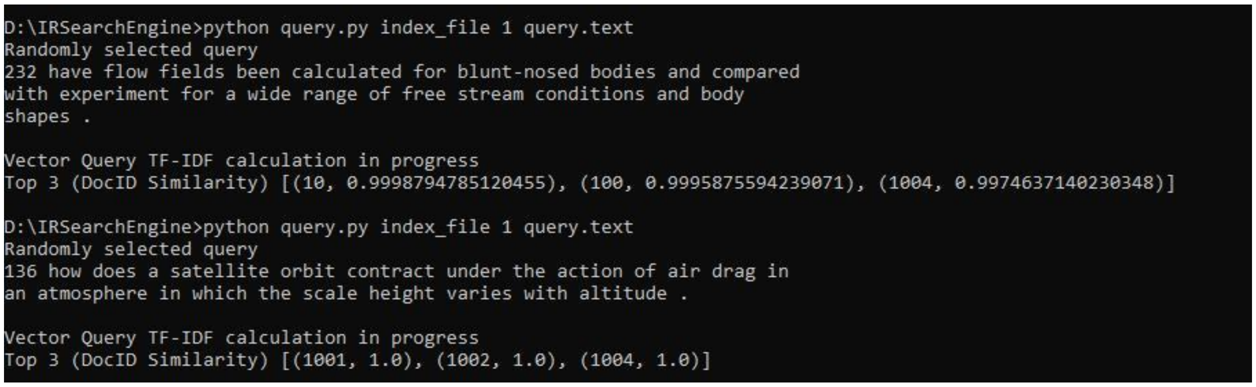
Vector:

python3 query.py index_file 1 query.text

or

python query.py index_file 1 query.text

screenshot:



```
D:\IRSearchEngine>python query.py index_file 1 query.text
Randomly selected query
232 have flow fields been calculated for blunt-nosed bodies and compared
with experiment for a wide range of free stream conditions and body
shapes .

Vector Query TF-IDF calculation in progress
Top 3 (DocID Similarity) [(10, 0.9998794785120455), (100, 0.9995875594239071), (1004, 0.9974637140230348)]

D:\IRSearchEngine>python query.py index_file 1 query.text
Randomly selected query
136 how does a satellite orbit contract under the action of air drag in
an atmosphere in which the scale height varies with altitude .

Vector Query TF-IDF calculation in progress
Top 3 (DocID Similarity) [(1001, 1.0), (1002, 1.0), (1004, 1.0)]
```

3. Check Result Quality:

python3 batch_eval.py index_file query.text qrels.text N

or

python batch_eval.py index_file query.text qrels.text N

Example:

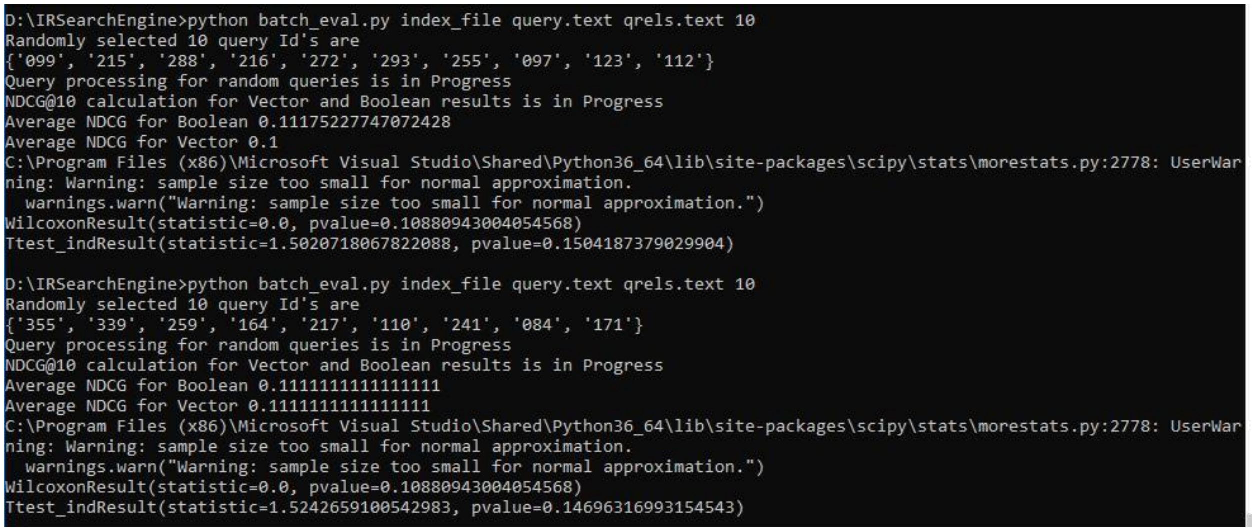
N is the number of randomly selected queries

python3 batch_eval.py index_file query.text qrels.text 10
or

python batch_eval.py index_file query.text qrels.text 10

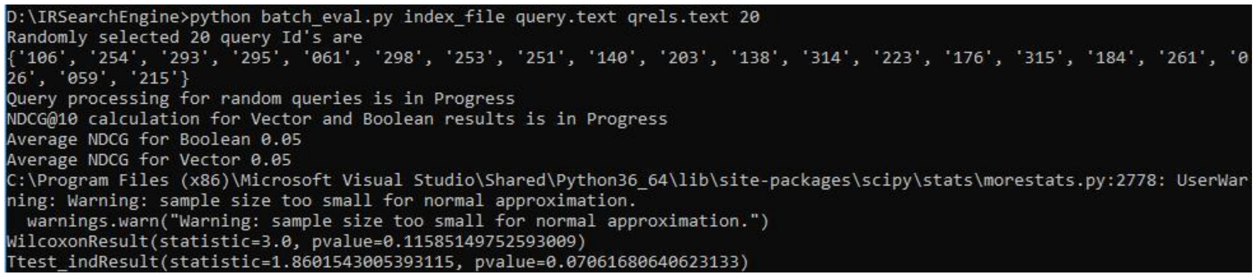
Note: To avoid warning of small sample size for normal approximation with Wilcoxon Result, N should be >=20

Below :20



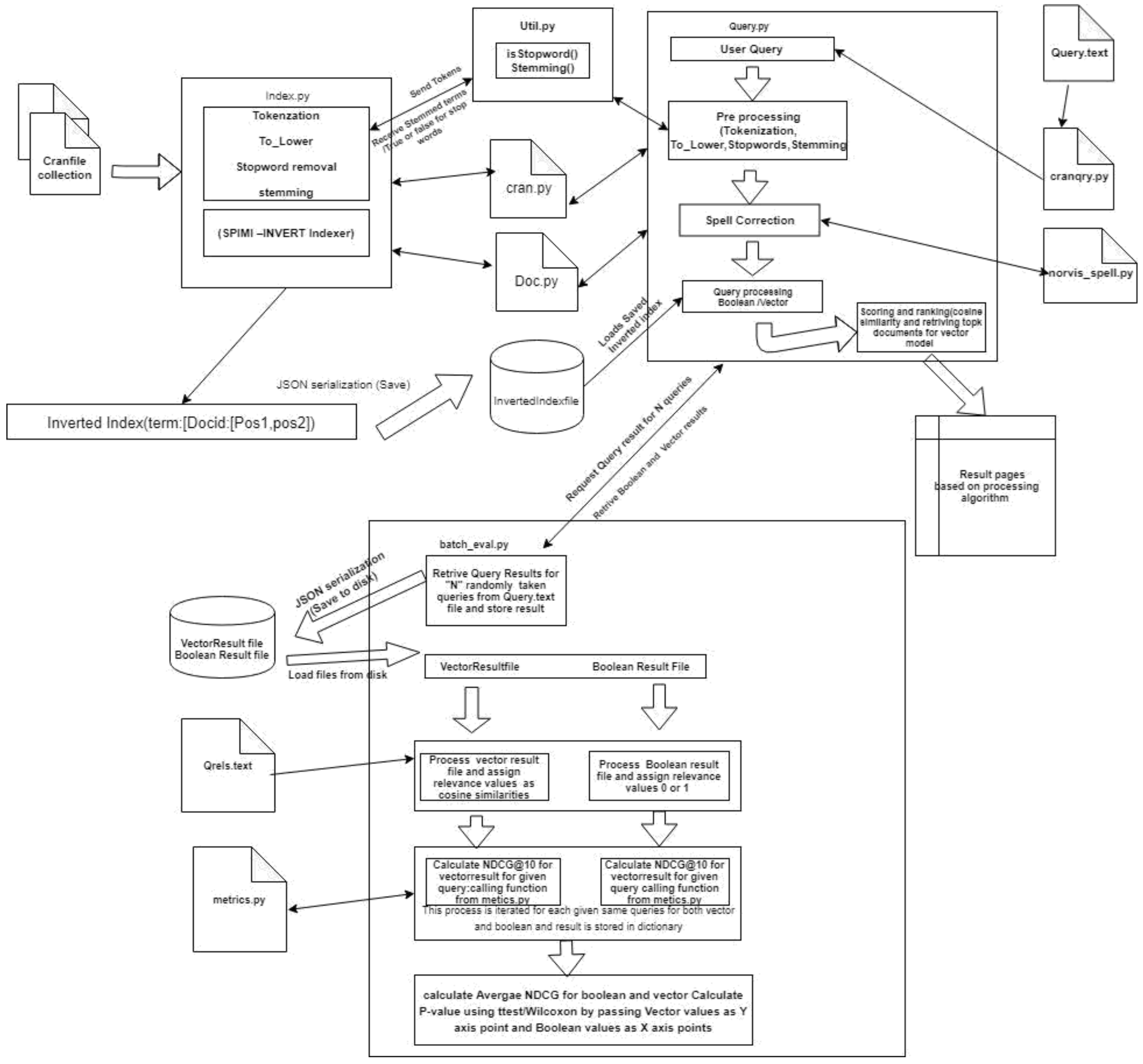
```
D:\IRSearchEngine>python batch_eval.py index_file query.text qrels.text 10
Randomly selected 10 query Id's are
{'099', '215', '288', '216', '272', '293', '255', '097', '123', '112'}
Query processing for random queries is in Progress
NDCG@10 calculation for Vector and Boolean results is in Progress
Average NDCG for Boolean 0.11175227747072428
Average NDCG for Vector 0.1
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64\lib\site-packages\scipy\stats\morestats.py:2778: UserWar
ning: Warning: sample size too small for normal approximation.
  warnings.warn("Warning: sample size too small for normal approximation.")
WilcoxonResult(statistic=0.0, pvalue=0.10880943004054568)
Ttest_indResult(statistic=1.5020718067822088, pvalue=0.1504187379029904)

D:\IRSearchEngine>python batch_eval.py index_file query.text qrels.text 10
Randomly selected 10 query Id's are
{'355', '339', '259', '164', '217', '110', '241', '084', '171'}
Query processing for random queries is in Progress
NDCG@10 calculation for Vector and Boolean results is in Progress
Average NDCG for Boolean 0.1111111111111111
Average NDCG for Vector 0.1111111111111111
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64\lib\site-packages\scipy\stats\morestats.py:2778: UserWar
ning: Warning: sample size too small for normal approximation.
  warnings.warn("Warning: sample size too small for normal approximation.")
WilcoxonResult(statistic=0.0, pvalue=0.10880943004054568)
Ttest_indResult(statistic=1.5242659100542983, pvalue=0.14696316993154543)
```

N=20



```
D:\IRSearchEngine>python batch_eval.py index_file query.text qrels.text 20
Randomly selected 20 query Id's are
{'106', '254', '293', '295', '061', '298', '253', '251', '140', '203', '138', '314', '223', '176', '315', '184', '261', '0
26', '059', '215'}
Query processing for random queries is in Progress
NDCG@10 calculation for Vector and Boolean results is in Progress
Average NDCG for Boolean 0.05
Average NDCG for Vector 0.05
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64\lib\site-packages\scipy\stats\morestats.py:2778: UserWar
ning: Warning: sample size too small for normal approximation.
  warnings.warn("Warning: sample size too small for normal approximation.")
WilcoxonResult(statistic=3.0, pvalue=0.11585149752593009)
Ttest_indResult(statistic=1.8601543005393115, pvalue=0.07061680640623133)
```

Project Design


Cranfile collection
Index.py
Tokenzation
To_Lower
Stopword removal
stemming
(SPIMI –INVERT Indexer)
Send Tokens
Receive Stemmed terms /True or false for stop words
Util.py
is Stopword()
Stemming()
cran.py
Doc.py
Query.py
User Query
Pre processing (Tokenization, To_Lower, Stopwords, Stemming
Spell Correction
Query processing Boolean /Vector
Scoring and ranking(cosine similarity and retriving topk documents for vector model
Query.text
cranqry.py
norvis_spell.py
Inverted Index(term:[Docid:[Pos1,pos2])
JSON serialization (Save)
InvertedIndexfile
Loads Saved Inverted index
Request Query result for N queries
Retrive Boolean and Vector results
Result pages based on processing algorithm
batch_eval.py
Retrive Query Results for "N" randomly taken queries from Query.text file and store result
JSON serialization (Save to disk)
VectorResult file Boolean Result file
Load files from disk
VectorResultfile
Boolean Result File
Qrels.text
Process vector result file and assign relevance values as cosine similarities
Process Boolean result file and assign relevance values 0 or 1
metrics.py
Calculate NDCG@10 for vectorresult for given query:calling function from metics.py
Calculate NDCG@10 for vectorresult for given query calling function from metics.py
This process is iterated for each given same queries for both vector and boolean and result is stored in dictionary
calculate Avergae NDCG for boolean and vector Calculate P-value using ttest/Wilcoxon by passing Vector values as Y axis point and Boolean values as X axis points

Initially documents are retrieved from cran.py file and preprocessing of documents is been done using SPIMI_Inverted algorithm inverted index has been created and stored to disk using JSON serialization.

Once the index been created, random query number is selected from data processed by cranqry.py file and sent to Query processor class in Query.py for preprocessing the query and processing further based on given input for processing algorithm. If given input matches the

Boolean retrieval mode for a given processed query terms list of postings will be retrieved from inverted index file and intersection of these postings been made to select the documents which contains all words of the query. The result of the intersection is posted as output or stored in another Boolean result dictionary for Quality check.

If algorithm processing input matches the Vector model, Term frequency and IDF are calculated for both query and document. Both Query vector and Document vectors are passed as inputs to cosine function (which is dot product of two vector followed by division of query and document length normalization), Output of the result contains set of document ID and cosine values. This output result is sorted based on Cosine value in decreasing order and top 3 documents with cosine value nearly or equal to 1.0 is displayed. These results are either display or stored in vector result dictionary for quality check.

Third phase in project is to do the quality check of retrieved results with sample data file called qrels.text. We first store Boolean and vector results for same set of random generated queries.The stores results are processed to assign relevance value 0 or 1 or Cosine similarity after comparison with data from qrels.text file. Once the relevance assignment has been done for both Query processing models, relevance for each query are passed to metrics.py file for retrieving NDCG values for each given query for both Boolean and vector models. These dictionary of NDCG are used to calculate avergae NDCG for Boolean and vector model. These dictionary values are further used to test function as X and Y axis values for Boolean and vector respectively. Output of the result display p-value for the given input.

Implementation

Setup details:

The project implementation started with initial setup of installing python 3 and its related content like pip3.

Later NLTK file been downloaded and NLTK corpus been imported for processing document ,to get tokens and for other NLTK related operation like stop words removal and Stemming.

Code level implementation:

First, we fetch the documents from cran.all file using cran.py file at index.py file and documents are stored in Collection class at doc.py.

Call to indexdoc(docs) been made with documents data and this document been converted to tokens by removing all punctuations. Later these tokens been sent to IsStopword(word) to check if the token is in stop words. If it is a stop word token is ignored and token list is iterated to check another token and position of token been stored using “Index Item” Class functions. Once tokens are preprocessed with stop words these tokens are stemmed used SPIMI-INVERT algorithm and created Inverted Index file and positions and Docid been sorted using Sort() function at indexitems and InvertedIndex class.implementation of IsStopword(word),Stemming(word) been done at Util.py file.Created Inverted Index has been stored to disk using JSON serialization technique implemenated at save() function of “InvertedIndex” class

Once index been created Query processing been done at Query.py file.Query.py uses cranqry.py to fetch query ID and Query text from “Query.text” file.Later these free language query text has been preprocessed using IsStopword(word),Stemming(word),Spellcorrection function at norvig_spell.py .Once the Query text is preprocessed these query terms are been queried using Boolean retrieval model or vector model. Boolean model code get postings list of the query using getpostinglist() which returns postings list from inverted index file. Finally all postings are merged (intersection) using mergeList(result1,results2) at Index.py file. Result is either displayed or stored if quality check of result is requested.

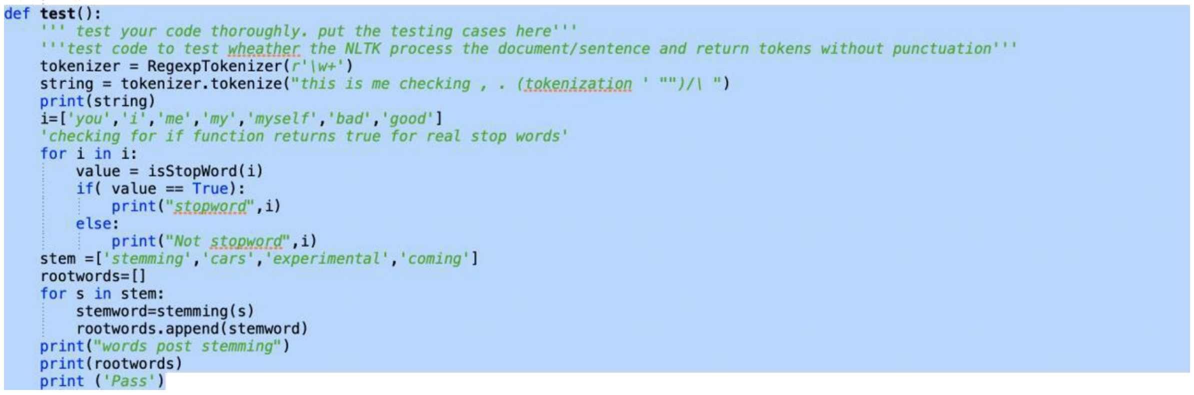
In Vector model for given query tokens Termfrequency and IDF are calculated by calling term_frequency() and IDF functions which are implemented in index.py file. Once these termfrequency and IDF for given terms are returned to calling function .these values are used to calculated TF-IDF of document and query terms in vectorquery() function of Query.py file and these Query and documents are converted to vectors and Score of the query w.r.t. document has been calculated using cosine_similaritys(queryvector,Document vector) and topk (document,SimilarityID) been displayed or these results been stored for quality checks.

In thirdPhase of the quality check of the results been implemented at batch_eval.py .Here call to query processing been made for randomly selected “N” queries and results are saved to disk using save() later these are retrieved using Load() function from index.py and relevance for the both Boolean and vector been assigned at eval() function and these results are passed to ndcg_score(true,score,=10,gains) function at metrics.py and NDCG @ 10 been calculated and stored.Avegae of these NDCG been calculated for both Vector and Boolean models along with calculating p-values with ttest and Wilcoxon test using NDCG for each given query of Boolean and vector. Finally it outputs Average NDCG and p-values.

Test

Testing is process of verification and validation of the product, below are test been done during project development and on final End product.

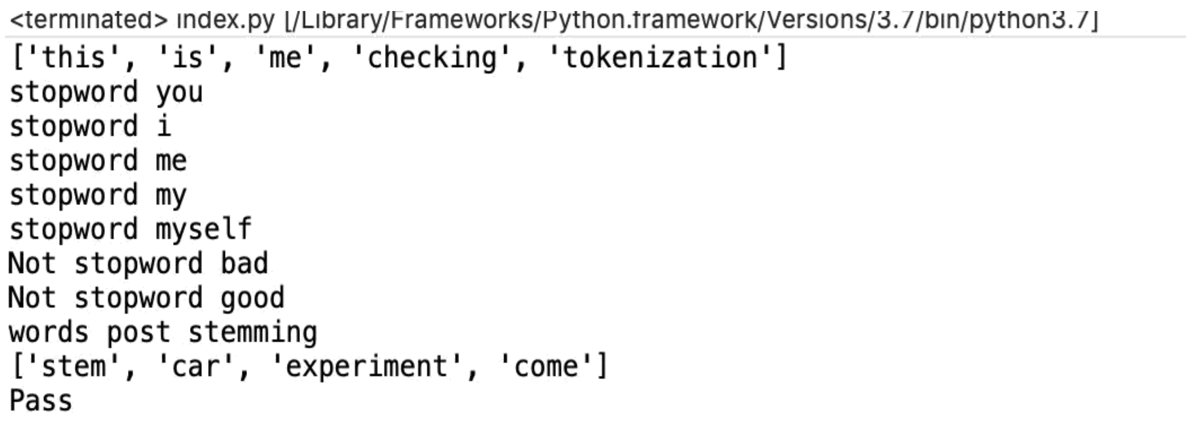
PreProcessing step correctness:



```
def test():
    ''' test your code thoroughly. put the testing cases here'''
    '''test code to test wheather the NLTK process the document/sentence and return tokens without punctuation'''
    tokenizer = RegexpTokenizer(r'\w+')
    string = tokenizer.tokenize("this is me checking , . (tokenization ' "")/\ ")
    print(string)
    i=['you','i','me','my','myself','bad','good']
    'checking for if function returns true for real stop words'
    for i in i:
        value = isStopWord(i)
        if( value == True):
            print("stopword",i)
        else:
            print("Not stopword",i)
    stem =['stemming','cars','experimental','coming']
    rootwords=[]
    for s in stem:
        stemword=stemming(s)
        rootwords.append(stemword)
    print("words post stemming")
    print(rootwords)
    print ('Pass')
```

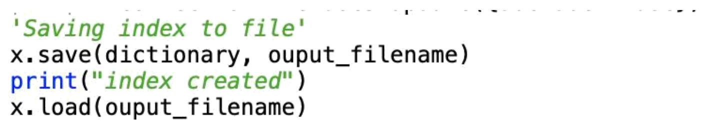
Output:

This output confirms the preprocessing step for stemming and stopword boolean result been classified correctly.



```
<terminated> index.py [/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7]
['this', 'is', 'me', 'checking', 'tokenization']
stopword you
stopword i
stopword me
stopword my
stopword myself
Not stopword bad
Not stopword good
words post stemming
['stem', 'car', 'experiment', 'come']
Pass
```

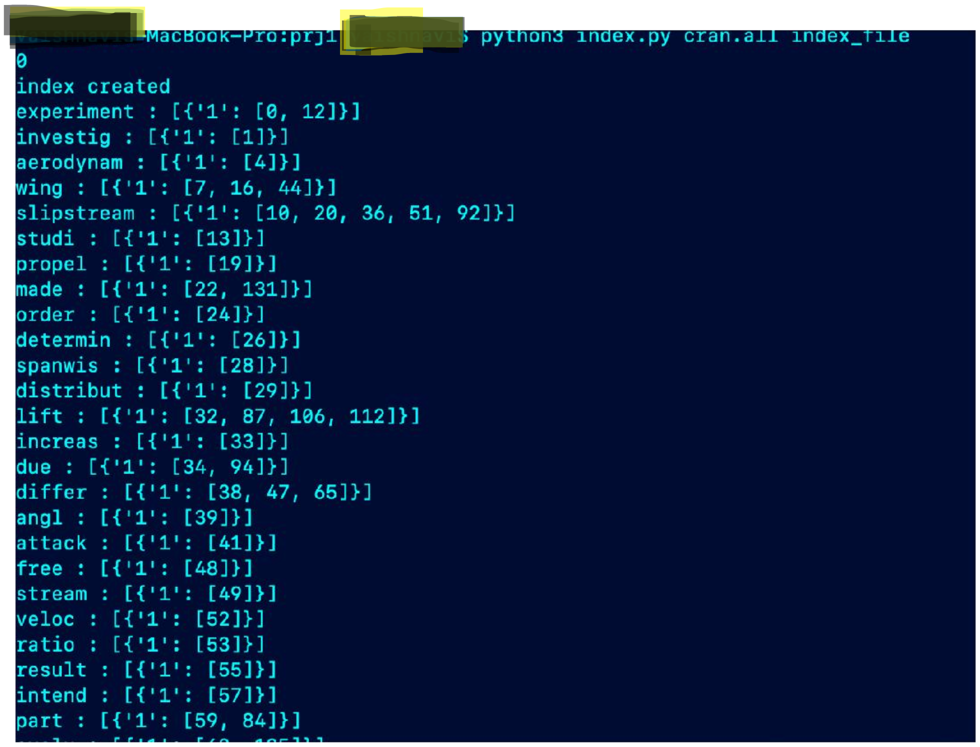
Verifying saved jsonfiles are loading properly for a single document from cranfile:



```
'Saving index to file'
x.save(dictionary, ouput_filename)
print("index created")
x.load(ouput_filename)
```

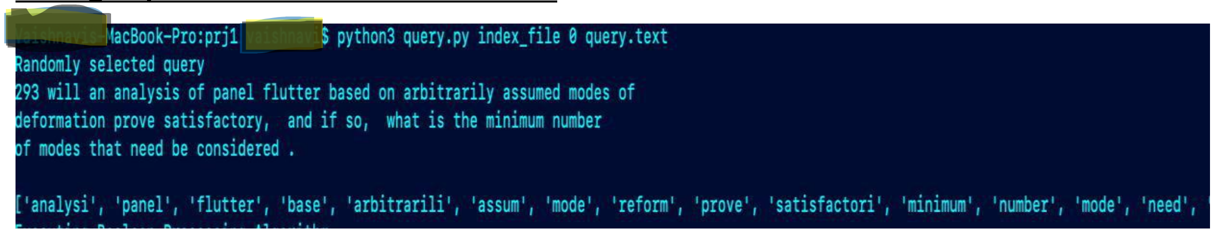
Output:

Saved files are loaded again and displayed on the screen.



```
[redacted]-MacBook-Pro:prj1 [redacted] python3 index.py cran.all index_file
0
index created
experiment : [{'1': [0, 12]}]
investig : [{'1': [1]}]
aerodynam : [{'1': [4]}]
wing : [{'1': [7, 16, 44]}]
slipstream : [{'1': [10, 20, 36, 51, 92]}]
studi : [{'1': [13]}]
propel : [{'1': [19]}]
made : [{'1': [22, 131]}]
order : [{'1': [24]}]
determin : [{'1': [26]}]
spanwis : [{'1': [28]}]
distribut : [{'1': [29]}]
lift : [{'1': [32, 87, 106, 112]}]
increas : [{'1': [33]}]
due : [{'1': [34, 94]}]
differ : [{'1': [38, 47, 65]}]
angl : [{'1': [39]}]
attack : [{'1': [41]}]
free : [{'1': [48]}]
stream : [{'1': [49]}]
veloc : [{'1': [52]}]
ratio : [{'1': [53]}]
result : [{'1': [55]}]
intend : [{'1': [57]}]
part : [{'1': [59, 84]}]
```

Testng if queries are converted to terms:



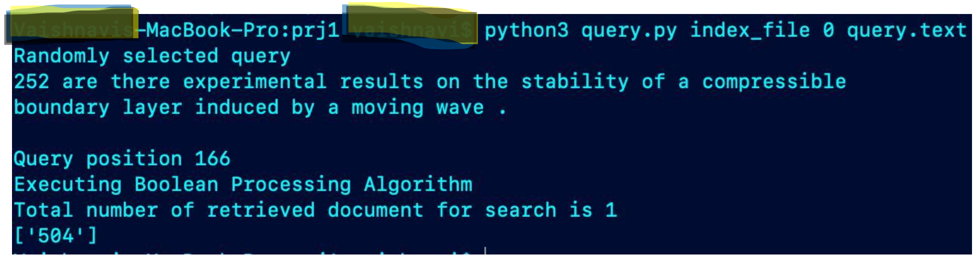
```
[redacted]MacBook-Pro:prj1 [redacted]$ python3 query.py index_file 0 query.text
Randomly selected query
293 will an analysis of panel flutter based on arbitrarily assumed modes of
deformation prove satisfactory,  and if so,  what is the minimum number
of modes that need be considered .

['analysi', 'panel', 'flutter', 'base', 'arbitrarili', 'assum', 'mode', 'reform', 'prove', 'satisfactori', 'minimum', 'number', 'mode', 'need',
```

sample queries getting the same results as you expect (manually computed) for boolean model processing

For a given random query

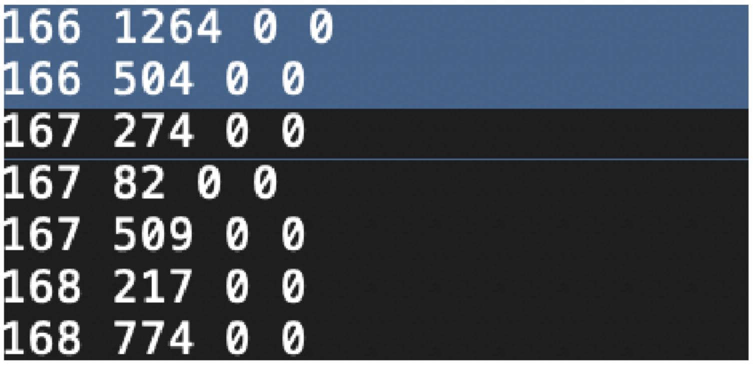
Output result matched with data from qrels.text file:

Actual output:retrived document 504 for the given query 252 and position 166



```
[redacted]-MacBook-Pro:prj1 [redacted] python3 query.py index_file 0 query.text
Randomly selected query
252 are there experimental results on the stability of a compressible
boundary layer induced by a moving wave .

Query position 166
Executing Boolean Processing Algorithm
Total number of retrieved document for search is 1
['504']
```

True output:In qrels.etxt for given Query position 166 output is 504 which is relevant.

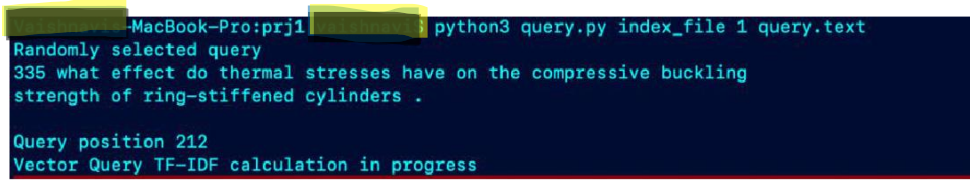


```
166 1264 0 0
166 504 0 0
167 274 0 0
167 82 0 0
167 509 0 0
168 217 0 0
168 774 0 0
```

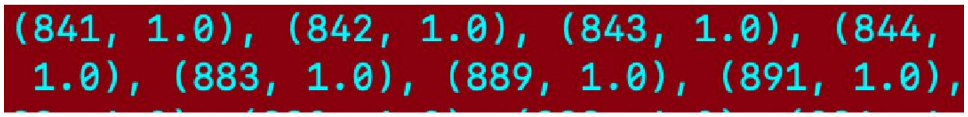
Verification: getting the same results as you expect for vector model processing

For a random selected query 335 with query position 212 below is the result for vector model is

Actual output:

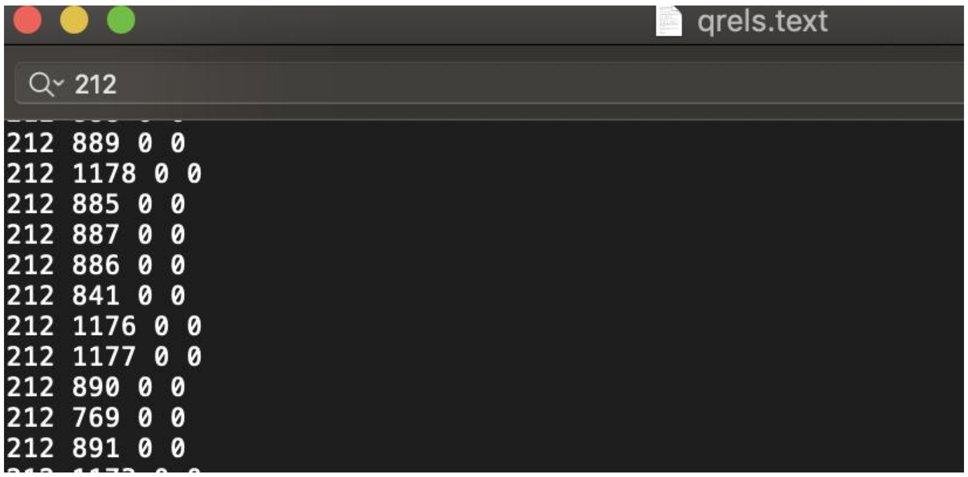


```
[redacted]-MacBook-Pro:prj1 [redacted] python3 query.py index_file 1 query.text
Randomly selected query
335 what effect do thermal stresses have on the compressive buckling
strength of ring-stiffened cylinders .

Query position 212
Vector Query TF-IDF calculation in progress
```

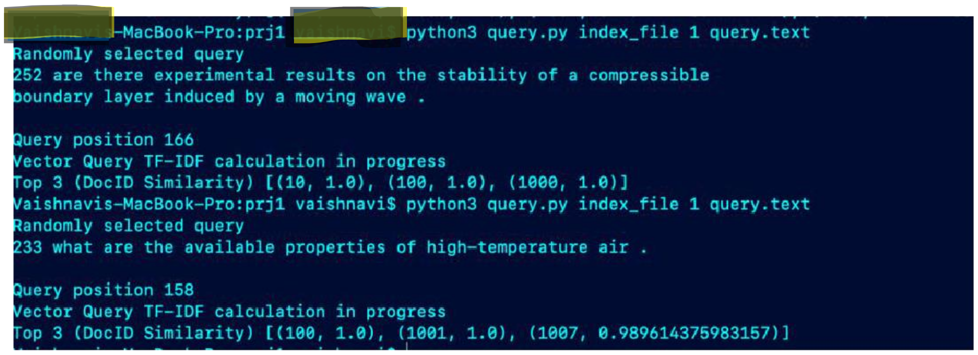


```
(841, 1.0), (842, 1.0), (843, 1.0), (844,
 1.0), (883, 1.0), (889, 1.0), (891, 1.0),
```

True output:

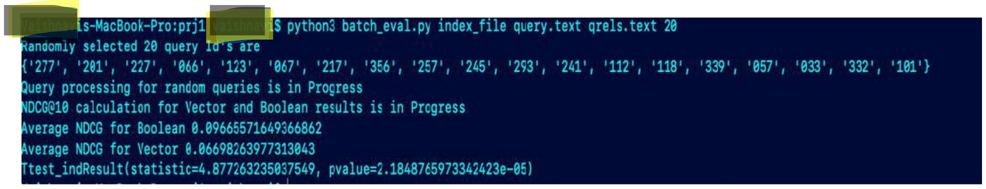


```
212 889 0 0
212 1178 0 0
212 885 0 0
212 887 0 0
212 886 0 0
212 841 0 0
212 1176 0 0
212 1177 0 0
212 890 0 0
212 769 0 0
212 891 0 0
```

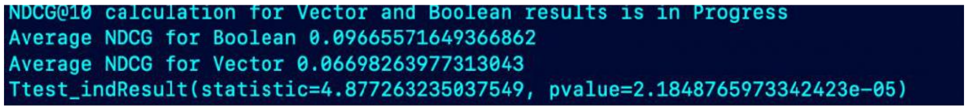


```
[redacted]-MacBook-Pro:prj1 [redacted] python3 query.py index_file 1 query.text
Randomly selected query
252 are there experimental results on the stability of a compressible
boundary layer induced by a moving wave .

Query position 166
Vector Query TF-IDF calculation in progress
Top 3 (DocID Similarity) [(10, 1.0), (100, 1.0), (1000, 1.0)]
Vaishnavis-MacBook-Pro:prj1 vaishnavi$ python3 query.py index_file 1 query.text
Randomly selected query
233 what are the available properties of high-temperature air .

Query position 158
Vector Query TF-IDF calculation in progress
Top 3 (DocID Similarity) [(100, 1.0), (1001, 1.0), (1007, 0.989614375983157)]
```

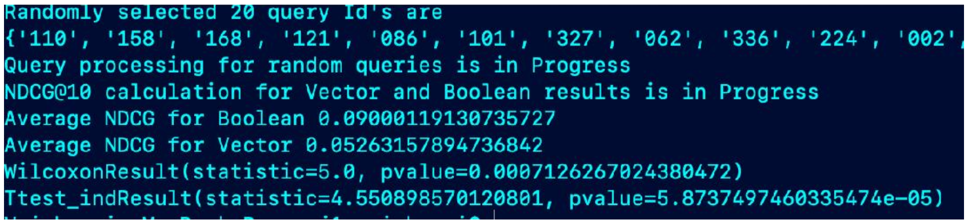
Ttest and Avergae NDGC for random 20 queries:



```
[redacted]is-MacBook-Pro:prj1 [redacted]$ python3 batch_eval.py index_file query.text qrels.text 20
Randomly selected 20 query Id's are
{'277', '201', '227', '066', '123', '067', '217', '356', '257', '245', '293', '241', '112', '118', '339', '057', '033', '332', '101'}
Query processing for random queries is in Progress
NDCG@10 calculation for Vector and Boolean results is in Progress
Average NDCG for Boolean 0.09665571649366862
Average NDCG for Vector 0.06698263977313043
Ttest_indResult(statistic=4.877263235037549, pvalue=2.1848765973342423e-05)
```



```
NDCG@10 calculation for Vector and Boolean results is in Progress
Average NDCG for Boolean 0.09665571649366862
Average NDCG for Vector 0.06698263977313043
Ttest_indResult(statistic=4.877263235037549, pvalue=2.1848765973342423e-05)
```



```
Randomly selected 20 query Id's are
{'110', '158', '168', '121', '086', '101', '327', '062', '336', '224', '002',
Query processing for random queries is in Progress
NDCG@10 calculation for Vector and Boolean results is in Progress
Average NDCG for Boolean 0.09000119130735727
Average NDCG for Vector 0.05263157894736842
WilcoxonResult(statistic=5.0, pvalue=0.0007126267024380472)
Ttest_indResult(statistic=4.550898570120801, pvalue=5.8737497460335474e-05)
```