Tiny Search Engine Project

Building inverted index and works boolean queries

Data Preprocessing

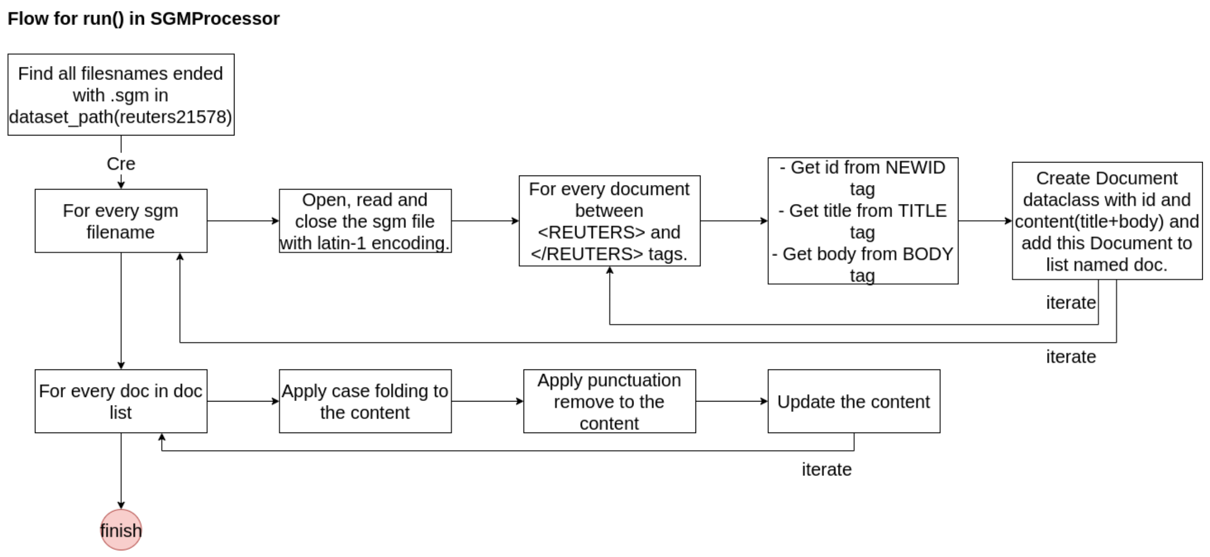
BaseTextProcessor class in base.py implemented to perform basic text preprocessing operations. Then SGMPreprocessor class inherits this class to implement specific operations for .sgm files used by Reuters-21578 Dataset.
BaseTextProcessor class includes these functions:

1. **tokenize**: Tokenization operation for converting strings to tokens i.e words
   Python split() function is used for this purpose to parse the text according to whitespaces.
2. **punctuation_remove:** Operation for removing punctuations.
   Python string.punctuations used for determining punctuations which includes !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~.
3. **stopwords:** Returns stopword list read from `stopwords.txt`
4. **case_folding:** Case-folding operation for string.
   The standard technique is to lowercase all words with the cost of losing information**[1]** There are 2 different options for this process. One is using simple lower() functions to make lower all the characters. The other is casefold() function in Python**[2].** Python casefold() function is chosen for this aim.

Reading documents in the given dataset and 2 preprocessing operations which are case folding and punctuation removal are done in run() function in SGMPreprocessor class. Python re(regular expressions) library is used for parsing SGM files. Every document between <REUTERS> and </REUTERS> tags is represented with a dataclass which is a Python data type. The definition of this dataclass is like this:

```
class Document():
   """
   Object for keeping necessary parts of each documents

   id -> NEWID element
   content -> includes both TITLE and BODY
   """
   id: int
   content: str
```

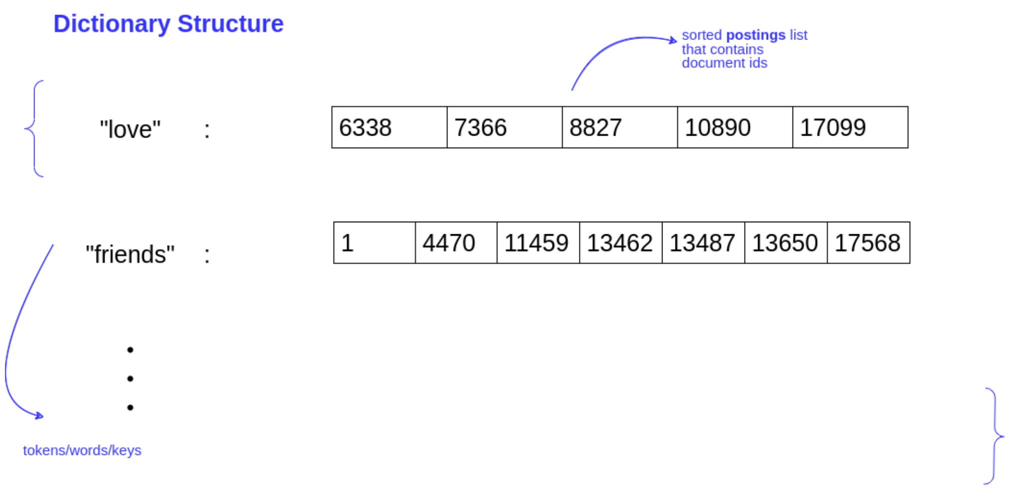
The flow is like this:



Flow for run() in SGMProcessor
Find all filesnames ended with .sgm in dataset_path(reuters21578)
Cre
For every sgm filename
Open, read and close the sgm file with latin-1 encoding.
For every document between <REUTERS> and </REUTERS> tags.
- Get id from NEWID tag
- Get title from TITLE tag
- Get body from BODY tag
Create Document dataclass with id and content(title+body) and add this Document to list named doc.
iterate
iterate
For every doc in doc list
Apply case folding to the content
Apply punctuation remove to the content
Update the content
iterate
finish

Inverted Index

BaseInvertedIndex class in base.py implemented to perform inverted index functions. All data operations are done in this class. Before functions let's take a look at the structure of the inverted index. The inverted index has a data structure named dictionary. The appearance of the dictionary can be shown like this:



Python dictionary data structure is used for this aim because it represents Hashmap in Python**[3]**. For map keys represent the tokens(words) and values represent the posting lists that keep document ids.

1. **public get():** Function for return the posting list(value) of given token. This function returns the posting list if the given token exists, otherwise returns an empty list. It uses to get the function of the hashmap so it can run in O(1) time.
2. **public add():** Function for adding a new key or value to the dictionary. When building an inverted index may new token can be added to the dictionary or a new document

id can be added to the posting list of the given key. There is a decision like this:

```
if token in self.dictionary.keys():
           self._insert(token, id)
       else:
           self._update(token, [id])
```

3. **private update():** It updates the dictionary with giving key-value pairs. It is used for adding new words to the dictionary when building an inverted index. It uses to update the function of the hashmap so it can run in O(1) time.
4. **private insert():** It inserts the given document id to the posting list of a given key. Because the posting lists are kept sorted, this function inserts the document id to where it belongs in order. It is implemented like a binary search tree so it can run in O(logn) time. The implementation is like this:

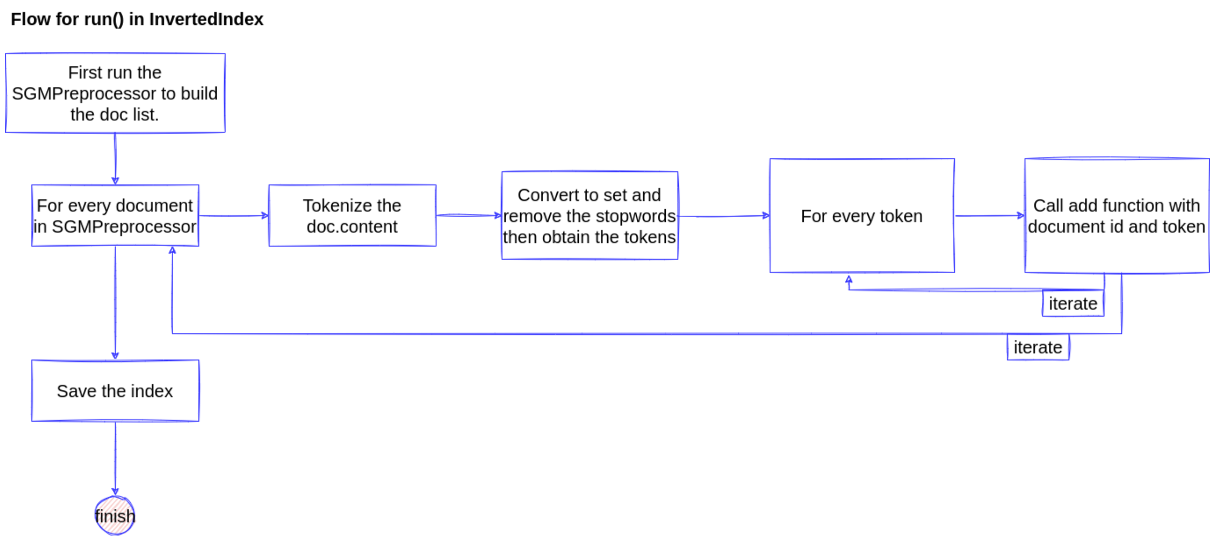
```
def _insert(self, key, id) -> None:
       posting = self.dictionary.get(key)
       start = 0
       end = len(posting) - 1

       while start <= end:
           middle = int((start+end)/2)
           if posting[middle] > id:
               end = middle - 1
           else:
               start = middle + 1
       posting.insert(start, id)
```

Required functions for building and processing the query which is an intersection, union, and difference implemented in InvertedIndex class that inherits BaseInvertedIndex. This class has 5 functions:

1. **build**: to build a dictionary
2. **save**: to save the dictionary
3. **load**: to load the saved dictionary if exist
4. **merge**: for intersection operation (AND)
5. **union**: for union operation (OR)
6. **difference**: for difference operation (NOT)

Build function is for building the inverted index. Save and load functions use the same dictionary_path. **pickle** is used for saving the dictionary. When the load function is called but the dictionary cannot be found, then the build function should be called. Merge, union, and difference functions are for processing the query. Merge(intersect) and union operations run in O(m+n) time.

InvertedIndex object has a SGMProcessor object to parse the dataset and get the documents. Flow of the build function is like this:


Flow for run() in InvertedIndex
First run the SGMPreprocessor to build the doc list.
For every document in SGMPreprocessor
Tokenize the doc.content
Convert to set and remove the stopwords then obtain the tokens
For every token
Call add function with document id and token
iterate
iterate
Save the index
finish

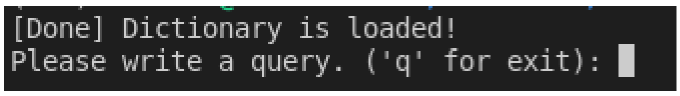
Query Processor

All query processing opreations are done in QueryProcessor class. In this design every QueryProcessor object has own InvertedIndex and BaseTextProcessor objects. The invertedIndex is initialized in constructor of the QueryPorcessor. The constructor of the QueryProcessor do this functions in order:

1. Initialize own InvertedIndex called it as index
2. Call the load function in index
3. If the load function returns with index skip the 5 step
4. If not then call the build function
5. Initialize own BaseTextProcessor called it as preprocessor
6. Define the operans (["AND", "OR", "NOT"])

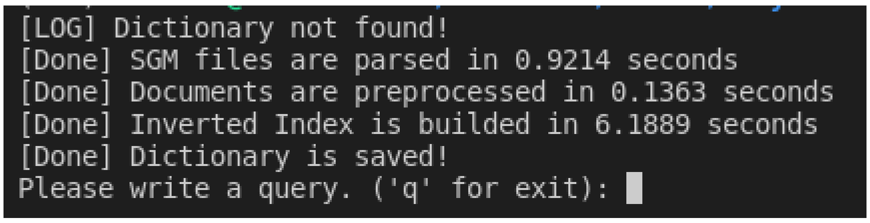
The preprocessor of the query do the same thing that done to the train data which are punctuation removal and case folding.

When the build inverted index found, query automatically works:



```
[Done] Dictionary is loaded!
Please write a query. ('q' for exit):
```

When the build inverted index not found, then it will builded:



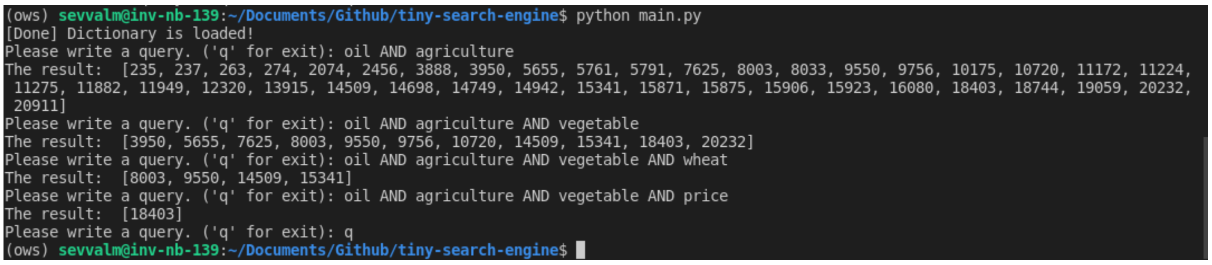
```
[LOG] Dictionary not found!
[Done] SGM files are parsed in 0.9214 seconds
[Done] Documents are preprocessed in 0.1363 seconds
[Done] Inverted Index is builded in 6.1889 seconds
[Done] Dictionary is saved!
Please write a query. ('q' for exit):
```

When the program started, it takes query to process until q is given.

Run Query Processor Examples

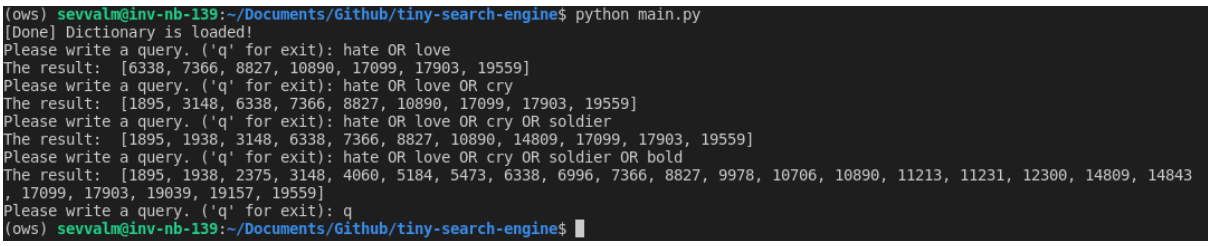
Here are some run examples from the query processor:

1. Conjunction: w1 AND w2 AND w3...AND wn



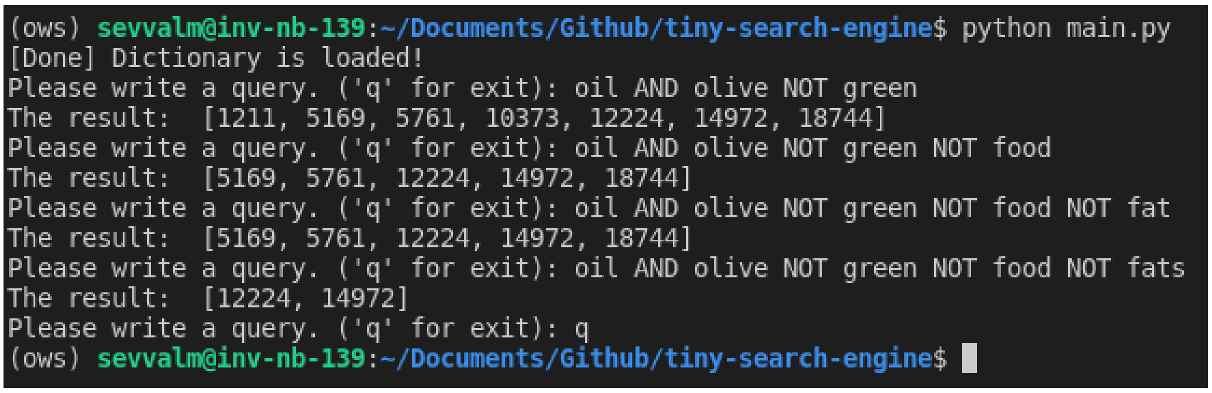
```
(ows) sevvalm@inv-nb-139:~/Documents/Github/tiny-search-engine$ python main.py
[Done] Dictionary is loaded!
Please write a query. ('q' for exit): oil AND agriculture
The result:  [235, 237, 263, 274, 2074, 2456, 3888, 3950, 5655, 5761, 5791, 7625, 8003, 8033, 9550, 9756, 10175, 10720, 11172, 11224,
 11275, 11882, 11949, 12320, 13915, 14509, 14698, 14749, 14942, 15341, 15871, 15875, 15906, 15923, 16080, 18403, 18744, 19059, 20232,
 20911]
Please write a query. ('q' for exit): oil AND agriculture AND vegetable
The result:  [3950, 5655, 7625, 8003, 9550, 9756, 10720, 14509, 15341, 18403, 20232]
Please write a query. ('q' for exit): oil AND agriculture AND vegetable AND wheat
The result:  [8003, 9550, 14509, 15341]
Please write a query. ('q' for exit): oil AND agriculture AND vegetable AND price
The result:  [18403]
Please write a query. ('q' for exit): q
(ows) sevvalm@inv-nb-139:~/Documents/Github/tiny-search-engine$
```

2. Disjunction: w1 OR w2 OR w3...OR wn



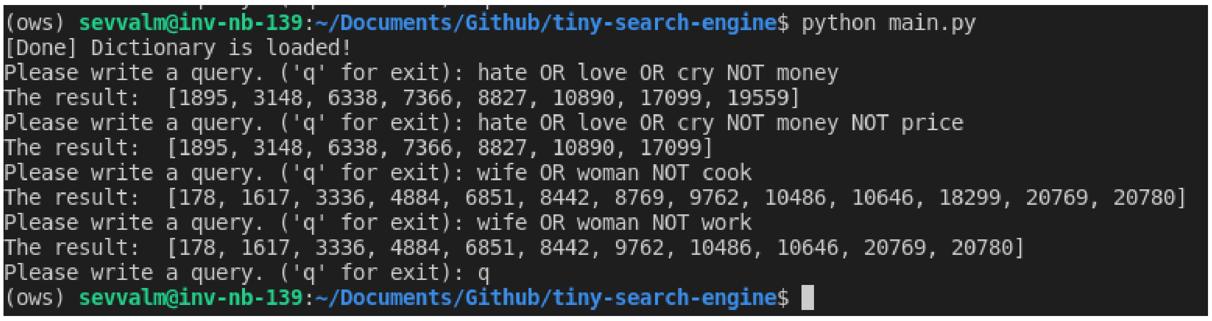
```
(ows) sevvalm@inv-nb-139:~/Documents/Github/tiny-search-engine$ python main.py
[Done] Dictionary is loaded!
Please write a query. ('q' for exit): hate OR love
The result:  [6338, 7366, 8827, 10890, 17099, 17903, 19559]
Please write a query. ('q' for exit): hate OR love OR cry
The result:  [1895, 3148, 6338, 7366, 8827, 10890, 17099, 17903, 19559]
Please write a query. ('q' for exit): hate OR love OR cry OR soldier
The result:  [1895, 1938, 3148, 6338, 7366, 8827, 10890, 14809, 17099, 17903, 19559]
Please write a query. ('q' for exit): hate OR love OR cry OR soldier OR bold
The result:  [1895, 1938, 2375, 3148, 4060, 5184, 5473, 6338, 6996, 7366, 8827, 9978, 10706, 10890, 11213, 11231, 12300, 14809, 14843
, 17099, 17903, 19039, 19157, 19559]
Please write a query. ('q' for exit): q
(ows) sevvalm@inv-nb-139:~/Documents/Github/tiny-search-engine$
```

3. Conjunction and Negation: w1 AND w2...AND wn NOT wn+1 NOT wn+2 ...NOT wn+m



```
(ows) sevvalm@inv-nb-139:~/Documents/Github/tiny-search-engine$ python main.py
[Done] Dictionary is loaded!
Please write a query. ('q' for exit): oil AND olive NOT green
The result:  [1211, 5169, 5761, 10373, 12224, 14972, 18744]
Please write a query. ('q' for exit): oil AND olive NOT green NOT food
The result:  [5169, 5761, 12224, 14972, 18744]
Please write a query. ('q' for exit): oil AND olive NOT green NOT food NOT fat
The result:  [5169, 5761, 12224, 14972, 18744]
Please write a query. ('q' for exit): oil AND olive NOT green NOT food NOT fats
The result:  [12224, 14972]
Please write a query. ('q' for exit): q
(ows) sevvalm@inv-nb-139:~/Documents/Github/tiny-search-engine$
```

4. Disjunction and Negation: w1 OR w2...OR wn NOT wn+1 NOT wn+2 ...NOT wn+m

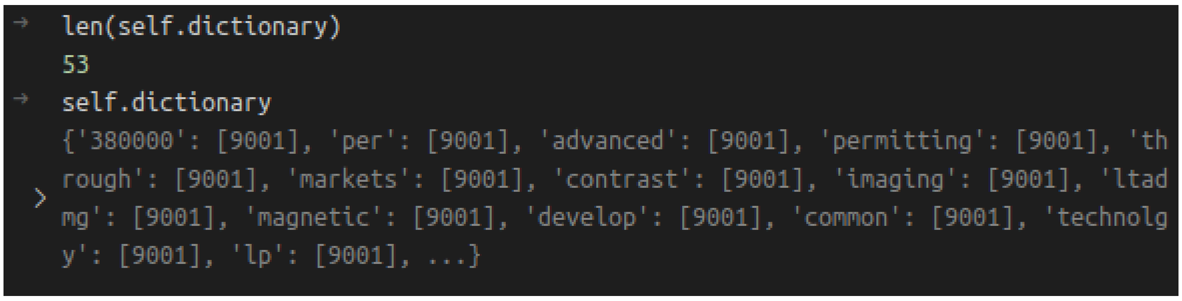


```
(ows) sevvalm@inv-nb-139:~/Documents/Github/tiny-search-engine$ python main.py
[Done] Dictionary is loaded!
Please write a query. ('q' for exit): hate OR love OR cry NOT money
The result:  [1895, 3148, 6338, 7366, 8827, 10890, 17099, 19559]
Please write a query. ('q' for exit): hate OR love OR cry NOT money NOT price
The result:  [1895, 3148, 6338, 7366, 8827, 10890, 17099]
Please write a query. ('q' for exit): wife OR woman NOT cook
The result:  [178, 1617, 3336, 4884, 6851, 8442, 8769, 9762, 10486, 10646, 18299, 20769, 20780]
Please write a query. ('q' for exit): wife OR woman NOT work
The result:  [178, 1617, 3336, 4884, 6851, 8442, 9762, 10486, 10646, 20769, 20780]
Please write a query. ('q' for exit): q
(ows) sevvalm@inv-nb-139:~/Documents/Github/tiny-search-engine$
```

Run Build

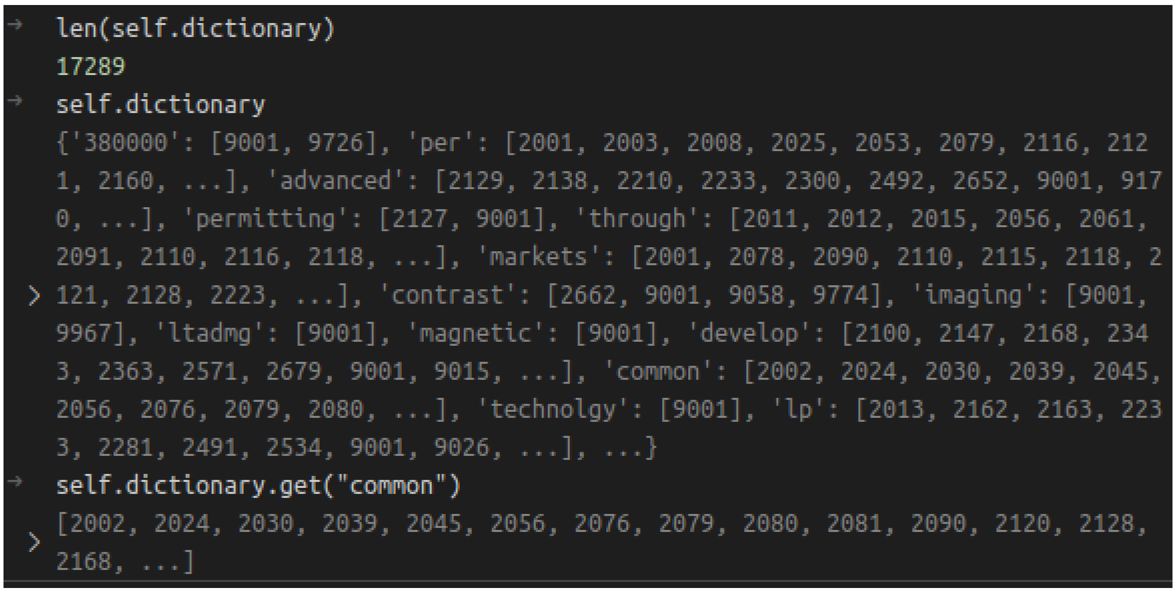
Here are some steps from while inverted index is building..

After doc 9001 is processed:



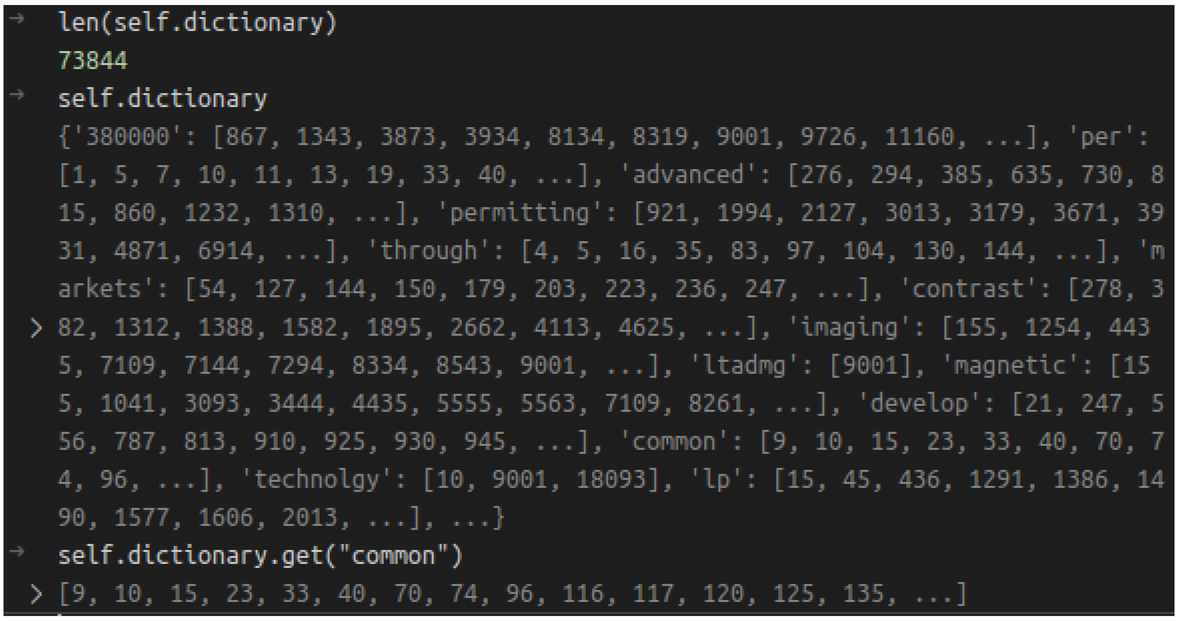
```
len(self.dictionary)
53
self.dictionary
{'380000': [9001], 'per': [9001], 'advanced': [9001], 'permitting': [9001], 'th
rough': [9001], 'markets': [9001], 'contrast': [9001], 'imaging': [9001], 'ltad
mg': [9001], 'magnetic': [9001], 'develop': [9001], 'common': [9001], 'technolg
y': [9001], 'lp': [9001], ...}
```

Few steps later, after more doc is processed:



```
len(self.dictionary)
17289
self.dictionary
{'380000': [9001, 9726], 'per': [2001, 2003, 2008, 2025, 2053, 2079, 2116, 212
1, 2160, ...], 'advanced': [2129, 2138, 2210, 2233, 2300, 2492, 2652, 9001, 917
0, ...], 'permitting': [2127, 9001], 'through': [2011, 2012, 2015, 2056, 2061,
2091, 2110, 2116, 2118, ...], 'markets': [2001, 2078, 2090, 2110, 2115, 2118, 2
121, 2128, 2223, ...], 'contrast': [2662, 9001, 9058, 9774], 'imaging': [9001,
9967], 'ltadmg': [9001], 'magnetic': [9001], 'develop': [2100, 2147, 2168, 234
3, 2363, 2571, 2679, 9001, 9015, ...], 'common': [2002, 2024, 2030, 2039, 2045,
2056, 2076, 2079, 2080, ...], 'technolgy': [9001], 'lp': [2013, 2162, 2163, 223
3, 2281, 2491, 2534, 9001, 9026, ...], ...}
self.dictionary.get("common")
[2002, 2024, 2030, 2039, 2045, 2056, 2076, 2079, 2080, 2081, 2090, 2120, 2128,
2168, ...]
```

After all documents are processed:



```
len(self.dictionary)
73844
self.dictionary
{'380000': [867, 1343, 3873, 3934, 8134, 8319, 9001, 9726, 11160, ...], 'per':
[1, 5, 7, 10, 11, 13, 19, 33, 40, ...], 'advanced': [276, 294, 385, 635, 730, 8
15, 860, 1232, 1310, ...], 'permitting': [921, 1994, 2127, 3013, 3179, 3671, 39
31, 4871, 6914, ...], 'through': [4, 5, 16, 35, 83, 97, 104, 130, 144, ...], 'm
arkets': [54, 127, 144, 150, 179, 203, 223, 236, 247, ...], 'contrast': [278, 3
82, 1312, 1388, 1582, 1895, 2662, 4113, 4625, ...], 'imaging': [155, 1254, 443
5, 7109, 7144, 7294, 8334, 8543, 9001, ...], 'ltadmg': [9001], 'magnetic': [15
5, 1041, 3093, 3444, 4435, 5555, 5563, 7109, 8261, ...], 'develop': [21, 247, 5
56, 787, 813, 910, 925, 930, 945, ...], 'common': [9, 10, 15, 23, 33, 40, 70, 7
4, 96, ...], 'technolgy': [10, 9001, 18093], 'lp': [15, 45, 436, 1291, 1386, 14
90, 1577, 1606, 2013, ...], ...}
self.dictionary.get("common")
[9, 10, 15, 23, 33, 40, 70, 74, 96, 116, 117, 120, 125, 135, ...]
```

Resources

[1]] MANNING, Christopher D, Prabhakar RAGHAVAN, and Hinrich SCHUTZE. Introduction to information retrieval. 1st pub. Cambridge: Cambridge University Press, 2008, xxi, 482 s. ISBN 9780521865715.
[2] https://docs.python.org/3/howto/unicode.html?highlight=casefold#comparing-strings
[3] https://docs.python.org/3/library/stdtypes.html#typesmapping