Tiny Search Engine Project

Building positional inverted index and works phrase&free text 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:

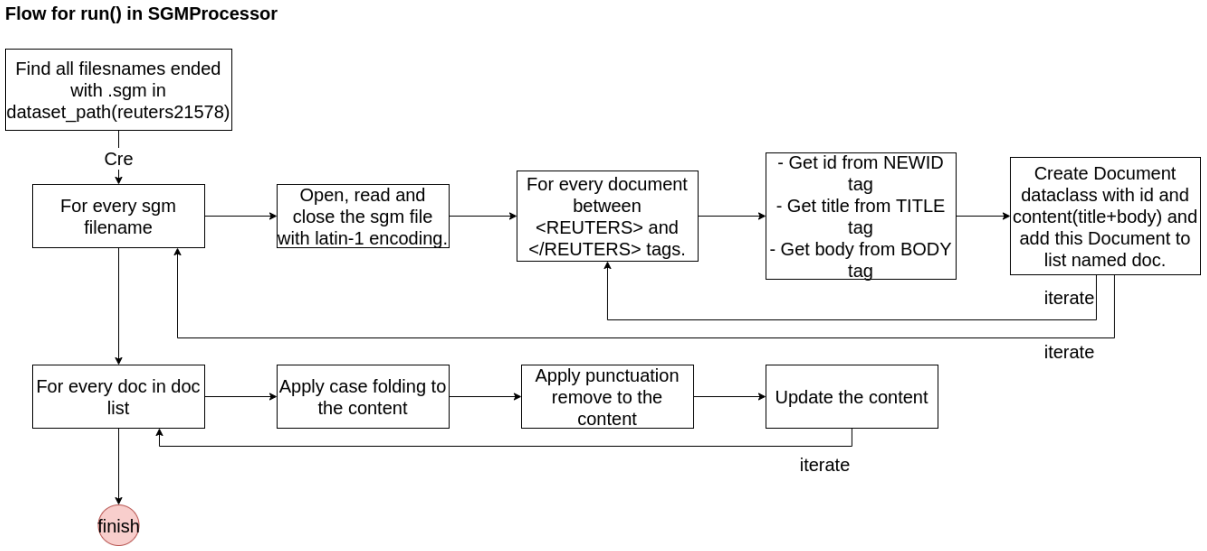
1. **tokenize**: Tokenization operation for converting strings to tokens i.e words
   Python split() function is used for this purpose to parse the text according to whitespaces.
2. **punctuation_remove:** Operation for removing punctuations.
   Python string.punctuations used for determining punctuations which includes !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~.
3. **stopwords:** Returns stopword list read from `stopwords.txt`
4. **stopwords_remove:** Remove stopwords in the given text.
5. **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
```

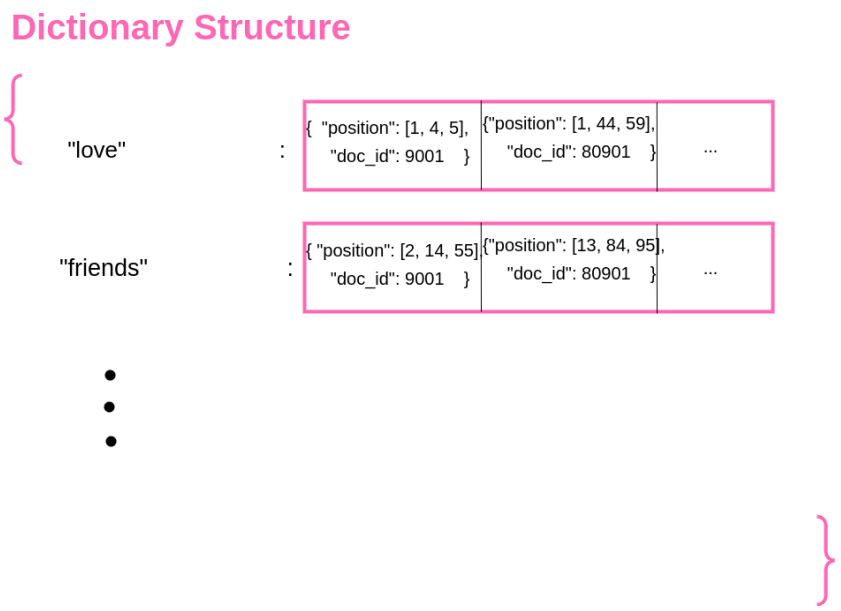
The flow is like this:



Positional Inverted Index

BaseInvertedIndex class in base.py implemented to perform inverted index functions. All data operations are done in this class. To build a positional inverted index, the new class called PositionalInvertedIndex is inherited from the inverted index class. And some functions such as merge, union, and update for updating the index are overridden.

Before functions let's take a look at the structure of the positional 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 **positions with document ids**.

1. **public get():** Function for return the posting list of a 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 any 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 a positional inverted index. Because of this new word is added to dictionary with position in list. (“position”:[4], “doc_id”: 7)
   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. This function is overridden for the positional inverted index.
   The key idea here is still keeping document ids sorted. In addition, each document id has a posting list here. Because of the positional issue, the function first check if the document id is in the list. If it is not in the list, the function append it with given position info. Otherwise, the function extends the existing positions list with a new coming position. The implementation pf this check is like this:

```
if the_index == -1:
   posting.insert(start, value)
else:
   posting[the_index].get("positions")
                     .extend(value.get("positions"))
```

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 **(overridden for Positional Inverted Index)**
2. **save**: to save the dictionary
3. **load**: to load the saved dictionary if exist
4. **merge**: for intersection operation (AND) **(overridden for Positional Inverted Index)**
5. **union**: for union operation (OR) **(overridden for Positional Inverted Index)**
6. **difference**: for difference operation (NOT) **(overridden for Positional Inverted Index for preventing the use)**

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.

Build, merge and union functions are overridden to use them with the position info besides the document id. The difference function is overridden to prevent the use of this function. For this assignmentPositionalInvertedIndex has another key, value pair to keep the number of documents, called N. It is kept to use for inverse document frequency. e.g: { “N”: 20578}

Query Processor

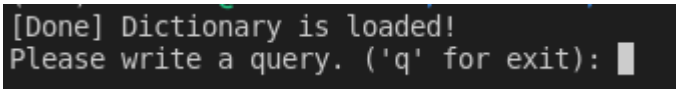
The system has a **BooleanQueryProcessor** class to implement the required functions. **QueryProcessor** class is created and inherited from **BooleanQueryProcessor** class.
All query processing operations are done in **QueryProcessor** class. In this design, every QueryProcessor object has its own PositionalInvertedIndex and BaseTextProcessor objects. The positional inverted index instance is initialized in the constructor of the QueryPorcessor. The constructor of the QueryProcessor do these functions in order:

1. Initialize own PositionalInvertedIndex called it as index
2. Call the load function in the 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. Get the N from the index.
7. Create a new dictionary to keep idf values.

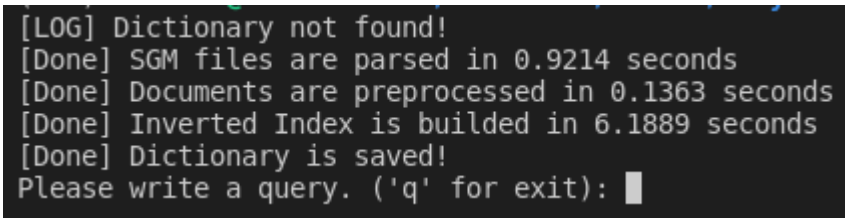
The preprocessor of the query do the same thing that done to the train data which are punctuation removal, stopword removal and case folding.

When the free-text query is given the system calculates log scaled term frequency and inverse document frequency values. These values can be calculated in the run time. The idea behind creating a new dictionary to keep inverse document frequency values is performance. After calculating the inverse document frequency value for the given token the system keeps this value and if this token is given again it is not calculated it can get from this dictionary.

When the build inverted index is found, query automatically works:



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



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

Phrase query: “w 1 w 2 ...w n ”

The system can detect this kind of query by checking the first and last characters before removing the punctuations. AND operator is used when processing the results that come from each word.

The queries are processed with pairs. For 3 words queries, first, w1 and w2 are processed then the sub_result of this operation is processed with w2. sub_result is come from merge(AND) function. These queries are processed like that:

The sub_result is like:

```
[
   {
      "left" : [1,2,3,89],
      "right": [2,3,4,34,145],
      "doc_id": 8907
   },
    {
      "left" : [9,26,37,89],
      "right": [2,3,4,38,145],
      "doc_id": 8910
   },
   ...
]
```

The idea: The system process over left and right positions, checks if left positions + count gives the right positions. If so keep them in a new key called positions.

:return: The processed sub_result will be like:

```
[
   {
      "positions" : [1,2,3],
      "doc_id": 8907
   },
    {
      "positions" : [37],
      "doc_id": 8910
   },
   ...
]
```

Free text query: w 1 w 2 ...w n

For free text queries, cosine similarity with (log-scaled) TF-IDF weighting is used. The query processor returns the IDs of the documents as well as their cosine similarity scores, ranked by their cosine similarities to the query.

The queries are processed with pairs. For 3 words queries, first, w1 and w2 are processed then the sub_result of this operation is processed with w2. sub_result is come from merge(AND) function. These queries are processed like that:

The sub_result is like:

```
[
   {
      "left" : [1,2,3,89],
      "right": [2,3,4,34,145],
      "doc_id": 8907
   },
    {
      "left" : [15, 19, 89],
      "right": [16, 34, 134,145],
      "doc_id": 8910
   },
   ...
]
```

The idea: Just eleminate unnecessary position informations. And keep doc ids

:return: The processed sub_result will be like:
[{'doc_id': 8907}, {'doc_id': 8910}, ..]

If the query is common topics and the document ids that include this words together are 3, 5 and 6. According to this example the table can be imagined:

	query	doc_id = 3	doc_id = 5
common	tf*idf(q, common)	tf*idf(doc3, common)	tf*idf(doc5, common)
topics	tf*idf(q, topics)	tf*idf(doc3, topics)	tf*idf(doc5, topics)

After constructing this table, the system extracts the document vectors and query vector. According to this table:
the vector belongs to query: [tf*idf(q, common), tf*idf(q, topics)]
the vector belongs to d3: [tf*idf(doc3, common), tf*idf(doc5, common)]
the vector belongs to d5: [tf*idf(doc3, topics), tf*idf(doc5, topics)]

Then cosine similarity is calculated for:

- between query and d3
- between query and d5

Then the results are ranked by cosine similarities.

Run Query Processor Examples

Here are some run examples from the query processor:

1. Phrase query: “w 1 w 2 ...w n ”

```
[Done] Dictionary is loaded!
Please write a query. ('q' for exit): "common stock based"
The result:  [{'positions': [29], 'doc_id': 5349}, {'positions': [41], 'doc_id': 6689}]
Please write a query. ('q' for exit): "received five mln"
The result:  [{'positions': [36], 'doc_id': 1221}, {'positions': [16], 'doc_id': 6706}]
Please write a query. ('q' for exit): "higher than expected"
The result:  [{'positions': [123], 'doc_id': 364}, {'positions': [100], 'doc_id': 1674}, {'positions': [540]
, 'doc_id': 5214}, {'positions': [342], 'doc_id': 7313}, {'positions': [96], 'doc_id': 7893}, {'positions':
[358], 'doc_id': 8137}, {'positions': [89], 'doc_id': 8716}, {'positions': [57], 'doc_id': 10845}, {'positio
ns': [57], 'doc_id': 10873}, {'positions': [67], 'doc_id': 11272}, {'positions': [142], 'doc_id': 13650}, {'
positions': [21], 'doc_id': 17836}, {'positions': [91, 200], 'doc_id': 18507}, {'positions': [11], 'doc_id':
 18511}, {'positions': [54], 'doc_id': 18672}, {'positions': [23, 144], 'doc_id': 19543}, {'positions': [56]
, 'doc_id': 20243}]
Please write a query. ('q' for exit): "wrong media"
The result:  []
Please write a query. ('q' for exit): "wrong field"
The result:  []
Please write a query. ('q' for exit): "wrong packaging"
The result:  [{'positions': [18], 'doc_id': 12415}]
Please write a query. ('q' for exit): q
```

2. Free text query: w 1 w 2 ...w n

```
[Done] Dictionary is loaded!
Please write a query. ('q' for exit): wrong media
The result:  ['Document-12673 with cosine similarity:1.000', 'Document-12750 with cosine similarity:1.000', 'Document-12879 with cosi
ne similarity:1.000']
Please write a query. ('q' for exit): wrong field
The result:  ['Document-15685 with cosine similarity:1.000']
Please write a query. ('q' for exit): wrong packaging
The result:  ['Document-12415 with cosine similarity:1.000']
Please write a query. ('q' for exit): received five mln
The result:  ['Document-493 with cosine similarity:1.000', 'Document-907 with cosine similarity:1.000', 'Document-946 with cosine sim
ilarity:1.000', 'Document-6930 with cosine similarity:1.000', 'Document-7088 with cosine similarity:1.000', 'Document-7473 with cosin
e similarity:1.000', 'Document-7892 with cosine similarity:1.000', 'Document-9526 with cosine similarity:1.000', 'Document-9924 with
cosine similarity:1.000', 'Document-17694 with cosine similarity:1.000', 'Document-17725 with cosine similarity:1.000', 'Document-185
31 with cosine similarity:1.000', 'Document-9761 with cosine similarity:0.996', 'Document-12938 with cosine similarity:0.993', 'Docum
ent-6706 with cosine similarity:0.990', 'Document-7306 with cosine similarity:0.990', 'Document-15370 with cosine similarity:0.990',
'Document-18568 with cosine similarity:0.990', 'Document-1369 with cosine similarity:0.989', 'Document-2707 with cosine similarity:0.
977', 'Document-6665 with cosine similarity:0.977', 'Document-10133 with cosine similarity:0.977', 'Document-17796 with cosine simila
rity:0.977', 'Document-18470 with cosine similarity:0.977', 'Document-21141 with cosine similarity:0.977', 'Document-1221 with cosine
 similarity:0.971', 'Document-3662 with cosine similarity:0.966', 'Document-9244 with cosine similarity:0.966', 'Document-7925 with c
osine similarity:0.965', 'Document-14731 with cosine similarity:0.965', 'Document-19292 with cosine similarity:0.965', 'Document-9655
 with cosine similarity:0.961', 'Document-1919 with cosine similarity:0.955', 'Document-16163 with cosine similarity:0.955', 'Documen
t-827 with cosine similarity:0.954', 'Document-10691 with cosine similarity:0.954', 'Document-12449 with cosine similarity:0.954', 'D
ocument-6618 with cosine similarity:0.949', 'Document-8115 with cosine similarity:0.949', 'Document-16028 with cosine similarity:0.94
9', 'Document-1456 with cosine similarity:0.944', 'Document-6209 with cosine similarity:0.937', 'Document-12292 with cosine similarit
y:0.937', 'Document-16887 with cosine similarity:0.937', 'Document-17891 with cosine similarity:0.937', 'Document-17571 with cosine s
imilarity:0.929', 'Document-9650 with cosine similarity:0.923']
Please write a query. ('q' for exit): q
```

Run Build

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

After doc 9001 is processed:

```
len(self.dictionary)
53
self.dictionary.get("advanced")
[{'positions': [...], 'doc_id': 9001}]
› special variables
› function variables
› 0: {'positions': [0, 4, 28, 50], 'doc_id': 9001}
  len(): 1
self.dictionary.get("magnetics")
[{'positions': [...], 'doc_id': 9001}]
› special variables
› function variables
› 0: {'positions': [1, 5, 29, 51], 'doc_id': 9001}
  len(): 1
```

A few steps later, after more doc is processed:

```
len(self.dictionary)
23548
self.dictionary.get("advanced")
[{'positions': [...], 'doc_id': 1232}, {'positions': [...], 'doc_id': 1310}, {'position
s': [...], 'doc_id': 1595}, {'positions': [...], 'doc_id': 1697}, {'positions': [...],
'doc_id': 1796}, {'positions': [...], 'doc_id': 1808}, {'positions': [...], 'doc_id': 1
> special variables
> function variables
> 00: {'positions': [62], 'doc_id': 1232}
> 01: {'positions': [25], 'doc_id': 1310}
> 02: {'positions': [39], 'doc_id': 1418}
> 03: {'positions': [11, 47], 'doc_id': 1542}
> 04: {'positions': [225], 'doc_id': 1582}
> 05: {'positions': [58], 'doc_id': 1595}
```

After all documents are processed:

```
len(self.dictionary)
73845
self.dictionary
{'advanced': [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, ...], 'magnetics': [{...},
{...}, {...}, {...}, {...}, {...}], 'ltadmg': [{...}], 'agreement': [{...}, {...}, {...}, {...}, {...},
{...}, {...}, {...}, {...}, ...], 'inc': [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...},
...], 'said': [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, ...], 'reached': [{...},
{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, ...], 'four': [{...}, {...}, {...}, {...}, {...},
{...}, {...}, {...}, {...}, ...], 'mln': [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...},
...], 'dlrs': [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, ...], 'research': [{...},
{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, ...], 'development': [{...}, {...}, {...}, {...},
{...}, {...}, {...}, {...}, {...}, ...], 'ml': [{...}, {...}, {...}, {...}], 'technolgy': [{...}, {...},
{...}], ...}
```

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