BOĞAZİÇİ UNIVERSITY

Cmpe 493

Assignment 2 - Report

Arzucan ÖZGÜR

Abdüllatif KÖKSAL

Baran Deniz KORKMAZ

FALL 2020

1 Introduction

In this assignment, we are asked to implement a retrieval system for single word queries and wildcard queries based on **Reuters-21578** data set. The retrieval system uses the inverted index and trie as data structures to process these queries successfully.

2 Program Interface

The program has been written in **Python 3**. The program has been tested in **Python 3.6.9**. The following standard-library modules are used:

1. os
2. sys
3. json
4. pickle
5. string
6. re

3 Program Execution

A query can be given into the program in two steps. In order to run the first step, the program assumes that **reuters21578** and **stopwords.txt** are provided in the working directory.

1. Construct the required data structures by entering the following command in terminal:

   ```
   >>> python3 main.py
   ```

   NOTE: The program saves the constructed structures in the files **inverted_index.json** and **trie.pickle**. Therefore, it assumes that the content and name of these files will not be changed.

2. Now that we have already constructed the required structures, enter the following command to process a query:

   ```
   >>> python3 main.py [string]
   ```

   Example: In order to search for the word **'retrieval'**, you should enter the following command.

```
>>> python3 main.py retrieval
```

4 Program Structure

This section describes the main stages of the program.

4.1 Data Preprocessing

The following steps have been performed for data preprocessing:

4.1.1 Parsing the Reuters21578 Dataset

In this step, the **SGM** files provided in the **Reuters21578** dataset have been parsed as articles using **Regex**. Each article in the files lies within <**REUTERS....**<**REUTERS**>. After an article is obtained, it can be used to derive the title and body using the patterns in the files. In the article, the title lies within <**TITLE**>**....**<**TITLE**>, while the body lies within <**BODY**>**....**<**BODY**>. Finally, the field of **NEWID**, can also be obtained using **Regex** as well to complete the parsing of an article. Please note that an article may not contain either a title or body. In this case, the title or body of the article will be regarded as an empty string.

4.1.2 Tokenization

In this step, the plain text content of title and body has been tokenized in a way that each word has been divided by whitespaces.

4.1.3 Normalization

In this step, the following operations have been applied to tokens obtained at the previous stage:

1. Case-Folding
2. Punctuation Removal
3. Stopword Removal

We must note that after the punctuations are replaced by a single whitespace, the tokens have been retokenized in a way that there exists no whitespaces in the final tokens.

4.2 Construction of Data Structures

This section describes the construction of required data structures, namely inverted index and trie, in order to process a query.

4.2.1 Inverted Index

The inverted index has been implemented as a dictionary where keys are strings of document ids and values are the lists of corresponding tokens within that document. The format of an inverted index is as follows:

```
{'1': ['token1_1', 'token1_2', ..., 'token1_a'],
 '2': ['token2_1', 'token2_2', ..., 'token2_b'],
  ...
  ...
  ...
  '21578': ['token21578_1', 'token21578_2', ..., 'token21578_c']
}
```

4.2.2 Trie

The implementation of trie has been done by class definitions. The trie has the following operations:

1. Insertion of a word.
2. Processing of a query that is actually a prefix. The words stored in the trie with that prefix have been returned.

The following code snippet includes the class definitions written in the implementation of trie data structure.

```
"""
    Defines the node structure for trie.
    Fields:
        char: Character - The character stored in this node.
        is_word: Boolean - If this node represents the end of a word or not.
        children: Dictionary - The dictionary that stores the child nodes.
            Key: Character
            Value: Node()
"""
class Node():
    def __init__(self,char):
        # character stored in this node
        self.char = char
        # whether this node is a word or not
        self.is_word = False
        # a dictionary that keeps the child nodes
        # dict: keys are characters, values are nodes
        self.children = dict()

"""
    Defines the trie structure.
```

```
"""
class Trie(object):
    def __init__(self):
        self.root = Node('')

    # Inserts a node into the trie.
    def insert(self,word):
        current = self.root
        for char in word:
            if char in current.children:
                current = current.children[char]
            else:
                current.children[char] = Node(char)
                current = current.children[char]
        current.is_word = True

    # Returns the output of a query.
    # Arguments:
    #   prefix: String - The string of prefix
    # Returns:
    #   List of words that are stored with that prefix in the trie.
    def query(self,prefix):
        self.query_output = list()
        current = self.root
        # Check if the prefix is in the trie
        for char in prefix:
            if char in current.children:
                current = current.children[char]
            else: # No prefix found!
                return []
        self.dfs(current,prefix)
        return self.query_output

    #
    # Performs dfs algorithm on a given node. The function recursively calls for all
    # children of a node until the currently called node has no children. The function
    # adds the valid words in the path into the query_output list.
    #
    def dfs(self,node,current_word):
        if node.is_word == True:
            self.query_output.append(current_word)
        for child in node.children.values():
            self.dfs(child, current_word+child.char)
```

4.3 Query Processing

In the final step, given that the required preprocessing and construction of data structures have been satisfied, we are ready to process a given query. The program returns the IDs of the matching documents sorted in **ascending** order in the following output format.

```
Query: [string]
Document IDs: [id1] [id2] ... [idN]
```

Please note that, at this stage, you do not need to reconstruct the data structures as they are imported from the files named **inverted_index.json** and **trie.pickle**. Therefore, we do not need to keep the folder of **reuters21578** in the working directory. However, also note that the program assumes that the names of files that store the previously saved structures are **unchanged** and the files are still in the working directory.

5 Input & Output

5.1 Input

The execution of program has two steps. In the first step, the program requires no inputs. In the second step, you must provide a string of query into the program. The query words can be in two different forms:

1. Single-word query: word
2. Prefix query: prefix*

5.2 Output

The execution of program has two steps. In the first step, the program saves the constructed data structures into the following files:

1. **inverted_index.json**
2. **trie.pickle**

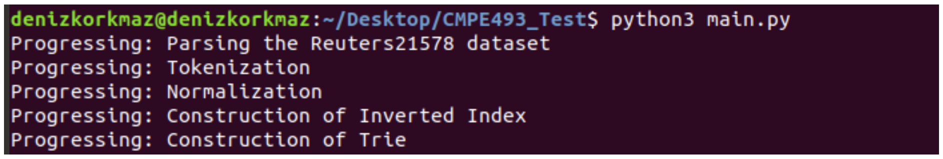
In the second step, the program provides the output for the query given in the following format:

```
Query: [string]
Document IDs: [id1] [id2] ... [idN]
```

5.3 Examples

1. Run the first step of execution:

```
>>> python3 main.py
Progressing: Parsing the Reuters21578 dataset
Progressing: Tokenization
Progressing: Normalization
Progressing: Construction of Inverted Index
Progressing: Construction of Trie
```



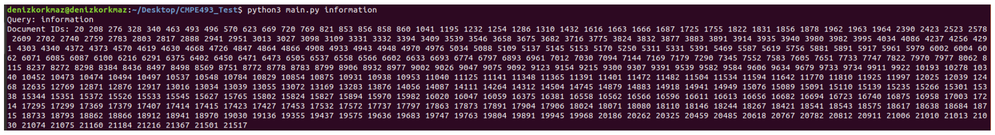
```
denizkorkmaz@denizkorkmaz:~/Desktop/CMPE493_Test$ python3 main.py
Progressing: Parsing the Reuters21578 dataset
Progressing: Tokenization
Progressing: Normalization
Progressing: Construction of Inverted Index
Progressing: Construction of Trie
```

Figure 1: Preprocessing and Construction of Data Structures

2. Request the query **'information'**:

```
>>> python3 main.py information
Query: information
Document IDs: 20 208 276 328 340 463 493 496 570 623 669 720 769 821 853 856 858 860
1041 1195 1232 1254 1286 1310 1432 1616 1663 1666 1687 1725 1755 1822 1831 1856 1878
1962 1963 1964 2390 2423 2523 2578 2609 2702 2740 2759 2783 2803 2817 2888 2941 2951
3013 3027 3098 3109 3331 3332 3394 3409 3539 3546 3658 3675 3682 3716 3775 3824 3832
3877 3883 3891 3914 3935 3940 3980 3982 3995 4034 4086 4237 4256 4291 4303 4340 4372
4373 4570 4619 4630 4668 4726 4847 4864 4866 4908 4933 4943 4948 4970 4976 5034 5088
5109 5137 5145 5153 5170 5250 5311 5331 5391 5469 5587 5619 5756 5881 5891 5917 5961
5979 6002 6004 6062 6071 6085 6087 6100 6216 6291 6375 6402 6450 6471 6473 6505 6537
6558 6566 6602 6633 6693 6774 6797 6893 6961 7012 7030 7094 7144 7169 7179 7290 7345
7552 7583 7605 7651 7733 7747 7822 7970 7977 8062 8115 8237 8272 8298 8384 8436 8497
8498 8569 8751 8772 8778 8783 8799 8906 8932 8977 9002 9026 9047 9075 9092 9123 9154
9215 9300 9307 9391 9539 9582 9584 9606 9634 9679 9733 9734 9911 9922 10193 10278 10340
10452 10473 10474 10494 10497 10537 10548 10784 10829 10854 10875 10931 10938 10953
11040 11125 11141 11348 11365 11391 11401 11472 11482 11504 11534 11594 11642 11770
11810 11925 11997 12025 12039 12468 12635 12769 12871 12876 12917 13016 13034 13039
13055 13072 13169 13283 13876 14056 14087 14111 14264 14312 14504 14745 14879 14883
14918 14941 14949 15076 15089 15091 15110 15139 15235 15266 15301 15338 15344 15351
15372 15526 15533 15545 15627 15765 15802 15824 15827 15894 15970 15982 16020 16047
16059 16375 16381 16558 16562 16566 16596 16611 16613 16656 16682 16694 16723 16740
16875 16958 17003 17214 17295 17299 17369 17379 17407 17414 17415 17423 17427 17453
17532 17572 17737 17797 17863 17873 17891 17904 17906 18024 18071 18080 18110 18146
18244
18267 18421 18541 18543 18575 18617 18638 18684 18715 18733 18793 18862 18866 18912
18941 18970 19030 19136 19355 19437 19575 19636 19683 19747 19763 19804 19891 19945
19968 20186 20262 20325 20459 20485 20618 20767 20782 20812 20911 21006 21010 21013
```

```
21030 21074 21075 21160 21184 21216 21367 21501 21517
```

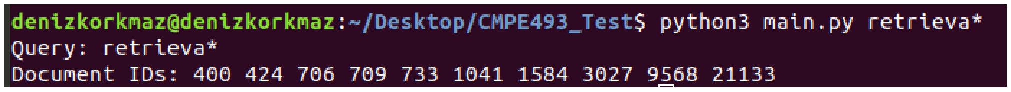


```
denizkorkmaz@denizkorkmaz:~/Desktop/CMPE493_Test$ python3 main.py information
Query: information
Document IDs: 20 208 276 328 340 463 493 496 570 623 669 720 769 821 853 856 858 860 1041 1195 1232 1254 1286 1310 1432 1616 1663 1666 1687 1725 1755 1822 1831 1856 1878 1962 1963 1964 2390 2423 2523 2578
 2609 2702 2740 2759 2783 2803 2817 2888 2941 2951 3013 3027 3098 3109 3331 3332 3394 3409 3539 3546 3658 3675 3682 3716 3775 3824 3832 3877 3883 3891 3914 3935 3940 3980 3982 3995 4034 4086 4237 4256 429
1 4303 4340 4372 4373 4570 4619 4630 4668 4726 4847 4864 4866 4908 4933 4943 4948 4970 4976 5034 5088 5109 5137 5145 5153 5170 5250 5311 5331 5391 5469 5587 5619 5756 5881 5891 5917 5961 5979 6002 6004 60
62 6071 6085 6087 6100 6216 6291 6375 6402 6450 6471 6473 6505 6537 6558 6566 6602 6633 6693 6774 6797 6893 6961 7012 7030 7094 7144 7169 7179 7290 7345 7552 7583 7605 7651 7733 7747 7822 7970 7977 8062 8
115 8237 8272 8298 8384 8436 8497 8498 8569 8751 8772 8778 8783 8799 8906 8932 8977 9002 9026 9047 9075 9092 9123 9154 9215 9300 9307 9391 9539 9582 9584 9606 9634 9679 9733 9734 9911 9922 10193 10278 103
40 10452 10473 10474 10494 10497 10537 10548 10784 10829 10854 10875 10931 10938 10953 11040 11125 11141 11348 11365 11391 11401 11472 11482 11504 11534 11594 11642 11770 11810 11925 11997 12025 12039 124
68 12635 12769 12871 12876 12917 13016 13034 13039 13055 13072 13169 13283 13876 14056 14087 14111 14264 14312 14504 14745 14879 14883 14918 14941 14949 15076 15089 15091 15110 15139 15235 15266 15301 153
38 15344 15351 15372 15526 15533 15545 15627 15765 15802 15824 15827 15894 15970 15982 16020 16047 16059 16375 16381 16558 16562 16566 16596 16611 16613 16656 16682 16694 16723 16740 16875 16958 17003 172
14 17295 17299 17369 17379 17407 17414 17415 17423 17427 17453 17532 17572 17737 17797 17863 17873 17891 17904 17906 18024 18071 18080 18110 18146 18244 18267 18421 18541 18543 18575 18617 18638 18684 187
15 18733 18793 18862 18866 18912 18941 18970 19030 19136 19355 19437 19575 19636 19683 19747 19763 19804 19891 19945 19968 20186 20262 20325 20459 20485 20618 20767 20782 20812 20911 21006 21010 21013 210
30 21074 21075 21160 21184 21216 21367 21501 21517
```

Figure 2: Query: information

3. Request the query **'retrieva*'**:

```
>>> python3 main.py retrieva*
Query: retrieva*
Document IDs: 400 424 706 709 733 1041 1584 3027 9568 21133
```



```
denizkorkmaz@denizkorkmaz:~/Desktop/CMPE493_Test$ python3 main.py retrieva*
Query: retrieva*
Document IDs: 400 424 706 709 733 1041 1584 3027 9568 21133
```

Figure 3: Query: retrieva*

6 References

- `https://dzlab.github.io/nlp/2018/11/17/parsing-xml-into-data frame/`
- `https://albertauyeung.github.io/2020/06/15/python-trie.html`