CMPE493 - Information Retrieval - Assignment 2

Berk Atıl - 2016400102

Introduction

In this assignment, we are asked to write a simple information retrieval system which can handle single word queries and wildcard queries via inverted indexing schema and a trie data structure. The database of the system is $Reuters - 21578$ data set. Firstly, some pre-processing is applied on the data and then both the inverted index schema and the trie structure is created. The details are explained later.

Preprocessing

In order to parse the data using "xml" library, the following pattern is removed from the data: "&#\d*;". This is because there were some strings like this which leads to not adhering xml rules. After this, "body" and "title" tags of each document are read using xml library and the following normalization operations are applied in this order:

- New line characters("\n") are replaced the with space character(" ").
- Case folding is applied to make uppercase letters lowercase.
- Punctuation marks(the ones in string.punctuations in Python) are removed.
- The text is split into tokens with separator " ".
- Stop words are removed

Inverted Index

Since inverted index contains posting lists for each word in our vocabulary, I chose to use map/dictionary data structure to store it. While creating it, firstly I made the value part as a "set" to ensure 1 document is added at most once into the posting list and after that I converted the sets into a "list". As a result, the inverted index is stored in a dictionary whose keys are the words in our vocabulary and values are the list of document IDs.

Trie

Trie is actually a tree so I created a class called "Node" which represents the nodes in a tree. The node class contains the following fields:

- letter: it is a letter stored in that node.(char)
- children_letters: it is a list of letters stored in the children of this node. (list/array)
- children_nodes: it is a list of pointers to children nodes
- is_terminal: it is a boolean value. It is true if there is a word that ends with this node and false otherwise. (boolean) For example if there is a word "turkey" the node storing "y" stores true for is_terminal.(Of course the node which is reached via using all previous characters t,u,r etc.)

```
class Node:
    """
    Parameters
    ----------
    letter:
        The value of this node
    children_letters:
        An array which contains the letters of children nodes
    children_nodes:
        An array of pointers to children nodes
    is_terminal:
        Stores if this node is the end of any word(last character of any word)
    """
    def __init__(self, letter,children_letters,children_nodes):
        self.letter = letter
        self.children_letters = children_letters
        self.children_nodes = children_nodes
        self.is_terminal = False

def find(array, elem):
    i = bisect.bisect_left(array, elem)
    if i != len(array) and array[i] == elem:
        return i
    return -1
```

Figure 1: Node Class

```
def create_trie(post_list):# post_list is a map whose keys are document IDs and values are the words in that document
    all_words = set()
    for words in post_list.values():
        for word in words:
            all_words.add(word)

    root = Trie.Node('',[],[])
    for word in all_words:
        current = root
        for i in range(len(word)):
            char = word[i]
            index = Trie.find(current.children_letters, char)# check if the children of our current node contains this char
            if index == -1:# this sequence has not been added so we need to add this char into a child node
                pos = bisect.bisect_left(current.children_letters, char) # find the position to insert
                bisect.insort(current.children_letters,char)
                node = Trie.Node(char,[],[])
                current.children_nodes.insert(pos,node)# insert into the same position
                current = node
            else:# this sequence is already added so move to that node
                current = current.children_nodes[index]
            if i == len(word) - 1: current.is_terminal = True

    return root
```

Figure 2: Trie Creation Code

Sample Input - Output

Input: berk*
Output :

```
berk@berk:~/Desktop/Course Books and Slides/CMPE493/InformationRetrieval/Assignment 2$ python3 query.py "berk*"
[635, 1066, 1393, 3145, 9440, 15232, 16326, 17554, 20765, 21411, 21438, 21461]
```

Figure 3: Sample Input - Output