Web Mining Lab - 5

Aadhitya Swarnesh

[redacted]

- 17 March 2021

Question

Build the inverted index for the following documents:
ID1 : Selenium is a portable framework for testing web applications
ID2 : Beautiful Soup is useful for web scraping
ID3: It is a python package for parsing the pages

Perform Index Compression for the integer values in the inverted index (duplicates to be eliminated) using Elias delta coding and variable byte scheme .

We will first pre-process the documents, and then split the documents into tokens or words. The pre-processing includes conversion to lower case, removal of numbers and other special characters, and stop word removal. After this we tokenise this document using the **NLTK** library.

After this we then take all these words after tokenisation and form a inverted index, which will include the word, and a list of occurrences in all the documents, and in each entry of this array would include the document number, number of times this word has occurred in this document, and also a list of offset position where the word occurs in the document.

This map developed in the above stated process is our **Inverted Index** map for the documents stated above, it is as follows :

```
Inverted Index :
('selenium', [(1, 1, [0])])
('portable', [(1, 1, [3])])
('framework', [(1, 1, [4])])
('testing', [(1, 1, [6])])
('web', [(1, 1, [7]), (2, 1, [5])])
('applications', [(1, 1, [8])])
('beautiful', [(2, 1, [0])])
('soup', [(2, 1, [1])])
('useful', [(2, 1, [3])])
('scraping', [(2, 1, [6])])
('python', [(3, 1, [3])])
('package', [(3, 1, [4])])
('parsing', [(3, 1, [6])])
('pages', [(3, 1, [8])])
```

Now for the second part of the question, we then extract all the numbers in this inverted index and then perform compression using the above stated two methods.

In the case of the **Elias Delta Encoding Scheme**, the results obtained are as follows :

```
Elias Delta Encoding Map :
(1, '1')
(0, '0')
(3, '0101')
(4, '01100')
(6, '01110')
(7, '01111')
(2, '0100')
(5, '01101')
(8, '00100000')
```

Similarly, the results obtained for the **Variable Byte Encoding Scheme** is as follows :

```
Variable Byte Encoding Map :
(1, '00000010')
(0, '00000000')
(3, '00000110')
(4, '00001000')
(6, '00001100')
(7, '00001110')
(2, '00000100')
(5, '00001010')
(8, '00010000')
```

The **code** that has delivered these results are as follows :

Inverted Indexing and Index Compression

Build the inverted index for the following documents:

- ID1 : Selenium is a portable framework for testing web applications
- ID2 : Beautiful Soup is useful for web scraping
- ID3: It is a python package for parsing the pages

Perform Index Compression for the integer values in the inverted index (duplicates to be eliminated) using Elias delta coding and variable byte scheme.

In [23]:

```
# Import statements

import math
import re
import string

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

In [2]:

```
# Documents list

documents = [
    "Selenium is a portable framework for testing web applications",
    "Beautiful Soup is useful for web scraping",
    "It is a python package for parsing the pages"
]
```

1. Inverted Index Construction

1.1 Pre-processing

Refer to this link for more details : [Text Preprocessing Reference (https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908)](https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908)

In [11]:

```
def preprocess(text) :
    """
    Given a text, we pre-process and return an array of the words from the text.
    """

    s = text

    # Convert to lower case
    s = s.lower()

    # Removing numbers and other numerical data
    # We substitute all the occurances of numbers by an empty string, thereby ef
fectively removing them.
    s = re.sub(r'\d+', '', s)

    # Remove punctuation signs
    #s = s.translate(string.maketrans("",""), string.punctuation)
    s = s.replace('/[^A-Za-z0-9]/g', '')

    # Trim the leading and trailing spaces
    s = s.strip()

    # Tokenize the text
    words = word_tokenize(s)

    # Stop Word Removal
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]

    # Return the word list
    return words
```

1.2 Find Occurance Function

In [7]:

```
def findOccurance(text, word) :
    """
    Given a text and the word to be found, we partially pre-process the text and
then return the count of occurances of the word,
    and the positions in the text where they occur.
    """

    # Split the text into tokens and remove the punctuation signs and convert to
lower case
    # This is to find the position of the words to have in the inverted index
    text = text.replace('/[^A-Za-z0-9]/g', '')
    text = text.replace('  ', ' ')
    text = text.lower()
    text_words = text.strip().split()

    word_count = 0
    word_positions = []

    for i in range(len(text_words)) :
        if word == text_words[i] :
            word_count += 1
            word_positions.append(i)

    return (word_count, word_positions)
```

1.3 Inverted Indexing

In [17]:

```
inverted_index = {}

# Process each of the documnet
for (i, doc) in enumerate(documents) :
    # Pre-Processing of each individual document
    words = preprocess(doc)

    # Add the words into the inverted index
    for word in words :
        # Create an entry for the word if one does not exist
        if word not in inverted_index :
            inverted_index[word] = []

        # Find all the occurances of the word in the doc.
        occurance_count, occurance_pos_list = findOccurance(doc, word)

        # Add these details into the inverted index
        inverted_index[word].append(((i+1), occurance_count, occurance_pos_list
))
```

Format for the inverted index is :

- inverted index :

```
{
  word : [
      (document_id, number_of_occurances_in_document, [offset_of_occur
ances]),
      ...
  ],
  ...
}
```

In [52]:

```
print('Inverted Index : ')
for item in inverted_index.items() :
    print(item)
```

```
Inverted Index :
('selenium', [(1, 1, [0])])
('portable', [(1, 1, [3])])
('framework', [(1, 1, [4])])
('testing', [(1, 1, [6])])
('web', [(1, 1, [7]), (2, 1, [5])])
('applications', [(1, 1, [8])])
('beautiful', [(2, 1, [0])])
('soup', [(2, 1, [1])])
('useful', [(2, 1, [3])])
('scraping', [(2, 1, [6])])
('python', [(3, 1, [3])])
('package', [(3, 1, [4])])
('parsing', [(3, 1, [6])])
('pages', [(3, 1, [8])])
```

2. Index Compression

For every unique number in the index, we create a map of the number to a encoded version of the number which occupies a lower size, thereby ensuring compression.

2.1 Binary Conversion

In [19]:

```
def binary(n) :
    """
    Given an integer number returns the equivalent binary string.
    """

    # Convert to binary string
    num = bin(n)

    # Remove the `0b` which is present in the front of the string
    num = num[2:]

    return num
```

2.2 Elias Gamma Encoding

In [42]:

```
def eliasGammaEncoding(n) :
    """
    Given an integer number `n`, we encode the number using the `Elias Gamma Enc
oding` scheme, and return the compressed value as a string.
    """

    # Zero is already encoded
    if n == 0 :
        return "0"

    # Find the binary value of number
    num = binary(n)

    # Prepend the value with (length-1) zeros
    num = ('0' * (len(num) - 1)) + num

    return num
```

2.3 Elias Delta Encoding

In [43]:

```
def eliasDeltaEncoding(n) :
    """
    Given an integer number `n`, we encode the number using the `Elias Delta Enc
oding` scheme, and return the compressed value as a string.
    """

    # Zero is already encoded
    if n == 0 :
        return "0"

    # Find the gamma code for (1 + log2(n))
    num1 = 1 + int(math.log2(n))
    num1 = eliasGammaEncoding(num1)

    # Number in binary form after removing the MSB
    num2 = binary(n)
    num2 = str(num2)[1:]

    # Combine the gamma code and the other code value
    num = num1 + num2

    return num
```

2.4 Variable Byte Encoding Scheme

In [47]:

```
def variableByteEncoding(n) :
    """
    Given an integer number `n`, we encode the number using the `Variable Byte E
ncoding` scheme, and return the compressed value as a string.
    """

    # Convert the number into binary form
    s = binary(n)

    result = ""

    while len(s) > 0 :
        # Get the term and update the binary string
        if len(s) > 7 :
            term = s[-7:]
            s = s[:-7]
        else :
            term = s
            s = ""
            term = ("0" * (7 - len(term))) + term

        if len(result) == 0 :
            result = term + "0"
        else :
            result = term + "1" + result

    return result
```

2.5 Index Compression Function

In [48]:

```
def indexCompression(inverted_index, encoding_scheme) :
    """
    Given an inverted index, we perform compression for all the integers in the
 inverted index and return the encoding map.
    """

    compression_map = {}

    for word_indices in inverted_index.values() :
        for word_index in word_indices :
            # Prepare an array to have all the numbers involved in this
            i, count, positions = word_index
            arr = [i, count] + positions

            # For each number compute and store the elias delta encoded value if
not already present
            for n in arr :
                if n not in compression_map :
                    if encoding_scheme == 'ELIAS_DELTA' :
                        compression_map[n] = eliasDeltaEncoding(n)
                    elif encoding_scheme == 'VARIABLE_BYTE' :
                        compression_map[n] = variableByteEncoding(n)

    return compression_map
```

2.6 Index Compression By Elias Delta

We perform compression for all the numbers using `Elias Delta` encoding scheme in the inverted index created in `Section 1`

In [49]:

```
elias_delta_compression_map = indexCompression(inverted_index, 'ELIAS_DELTA')

print("Elias Delta Encoding Map :")
for item in elias_delta_compression_map.items() :
    print(item)
```

```
Elias Delta Encoding Map :
(1, '1')
(0, '0')
(3, '0101')
(4, '01100')
(6, '01110')
(7, '01111')
(2, '0100')
(5, '01101')
(8, '00100000')
```

2.7 Index Compression By Variable Byte Encoding

We perform compression for all the numbers using `Variable Byte` encoding scheme in the inverted index created in `Section 1`

In [50]:

```
variable_byte_compression_map = indexCompression(inverted_index, 'VARIABLE_BYTE'
)

print("Variable Byte Encoding Map :")
for item in variable_byte_compression_map.items() :
    print(item)
```

```
Variable Byte Encoding Map :
(1, '00000010')
(0, '00000000')
(3, '00000110')
(4, '00001000')
(6, '00001100')
(7, '00001110')
(2, '00000100')
(5, '00001010')
(8, '00010000')
```

In [ ]:

```

```