Programme	:	**B.Tech** Semester : **Win Sem 21-22**
Course	:	**Web Mining Lab** Code : **CSE3024**
Faculty	:	**Dr.Bhuvaneswari A** Slot : **L7+L8**
Date	:	**04-02--2022** Marks : **10 Points**

Vaibhav Agarwal

19BCE1413

Exercise 4: Boolean and Vector Model, TF-IDF, Similarity Measures

Consider the following documents.

Doc 1 : Information Retrieval Systems is used with database systems

Doc 2 : Information is in Storage Storage

Doc 3 : Digital Speech systems can be used in Synthesis and Systems

Doc 4 : Speech Filtering, Speech Retrieval systems are applications of Information Retrieval

Doc 5: Database Management system is used for storage storage

i. Perform the text pre-processing of the given documents.

ii. Construct a Boolean Model for the vocabulary list by considering documents 1, 2, 3,4 and 5.

   a. Retrieve the documents for the Boolean query "Information Retrieval Synthesis" using simple match.

   b. Retrieve the documents for the Boolean query "Database Retrieval Storage" using weighted match. (Rank the documents in the order of relevance)

iii. Construct a vector space model to build the term weights. Compute the TF-IDF and identify the most important terms across the documents.

a. Rank all the documents in the collection for the query “Speech” AND “Systems”? (Rank the Top 3 documents in the order of relevance)

b. Rank all the documents in the collection for the query “Database” OR “Systems”? (Rank the Top N documents in the order of relevance)

c. Rank all the documents in the collection for the query contains “Systems” but NOT “Information” (Rank the documents in the order of relevance)

iv. Compute the cosine similarities between docs 1 and docs 2

v. Compute Dice Co-efficient between docs 3 and docs 4.

vi. Compute the Jaccard co-efficient between docs 4 and docs 5.

Question 1

```python
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.tokenize import sent_tokenize , word_tokenize
import glob
import re
import os
import numpy as np
import sys
Stopwords = set(stopwords.words('english'))
```

Python

```python
nltk.download('punkt')
nltk.download('stopwords')
```

Python

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\aayus\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\aayus\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```python
document=[
    "doc1.TXT",
    "doc2.TXT",
    "doc3.TXT",
    "doc4.TXT",
    "doc5.TXT"
]
```

Python

```python
def preprocess(text):
    s=text
    s=s.lower()
    s=s.replace('[A-Za-z0-9]/g','')
    s=s.strip()
    words=nltk.word_tokenize(s)
    stop_words=set(stopwords.words('english'))
    words=[word for word in words if word not in stop_words]
    return words
```

[4] Python

```python
for(i,doc) in enumerate(document):
    f = open(doc, "r")
    texts=f.read().strip()
    print(texts)
    words=preprocess(texts)
```

[5] Python

```
Information Retrieval Systems is used with database systems
Information is in Storage Storage
Digital Speech systems can be used in Synthesis and Systems
Speech Filtering, Speech Retrieval systems are applications of Information Retrieval
Database Management system is used for storage storage
```

Question 2

```python
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.tokenize import sent_tokenize , word_tokenize
import glob
import re
import os
import numpy as np
import sys
Stopwords = set(stopwords.words('english'))
```

[1] Python

```python
document=[
    "doc1.TXT",
    "doc2.TXT",
    "doc3.TXT",
    "doc4.TXT",
    "doc5.TXT"
]
```

[2] Python

```python
class Node:
    def __init__(self ,docId, freq = None):
        self.freq = freq
        self.doc = docId
        self.nextval = None

class SlinkedList:
    def __init__(self ,head = None):
        self.head = head
```

[3] Python

+ Code + Markdown

```python
def finding_all_unique_words_and_freq(words):
    words_unique = []
    word_freq = {}
    for word in words:
        if word not in words_unique:
            words_unique.append(word)
    for word in words_unique:
        word_freq[word] = words.count(word)
    return word_freq
def finding_freq_of_word_in_doc(word,words):
    freq = words.count(word)

def remove_special_characters(text):
    regex = re.compile('[^a-zA-Z0-9\s]')
    text_returned = re.sub(regex,'',text)
    return text_returned
```

[4] Python

```python
all_words = []
dict_global = {}
idx = 1
files_with_index = {}
for(i,doc) in enumerate(document):
    print(doc)
    fname = doc
    file = open(doc , "r")
    text = file.read()
    text = remove_special_characters(text)
    text = re.sub(re.compile('\d'),'',text)
    sentences = sent_tokenize(text)
    words = word_tokenize(text)
    words = [word for word in words if len(words)>1]
    words = [word.lower() for word in words]
    words = [word for word in words if word not in Stopwords]
    dict_global.update(finding_all_unique_words_and_freq(words))
    files_with_index[idx] = os.path.basename(fname)
    idx = idx + 1

unique_words_all = set(dict_global.keys())
```

[5] Python

```
doc1.TXT
doc2.TXT
doc3.TXT
doc4.TXT
doc5.TXT
```

```python
linked_list_data = {}
for word in unique_words_all:
    linked_list_data[word] = SlinkedList()
    linked_list_data[word].head = Node(1,Node)
word_freq_in_doc = {}
idx = 1
for(i,doc) in enumerate(document):
    file = open(doc, "r")
    text = file.read()
    text = remove_special_characters(text)
    text = re.sub(re.compile('\d'),'',text)
    sentences = sent_tokenize(text)
    words = word_tokenize(text)
    words = [word for word in words if len(words)>1]
    words = [word.lower() for word in words]
    words = [word for word in words if word not in Stopwords]
    word_freq_in_doc = finding_all_unique_words_and_freq(words)
    for word in word_freq_in_doc.keys():
        linked_list = linked_list_data[word].head
        while linked_list.nextval is not None:
            linked_list = linked_list.nextval
        linked_list.nextval = Node(idx ,word_freq_in_doc[word])
    idx = idx + 1
```

[6] Python

```
query = input('Enter your query:')
query = word_tokenize(query)
connecting_words = []
cnt = 1
different_words = []
for word in query:
    if word.lower() != "and" and word.lower() != "or" and word.lower() != "not":
        different_words.append(word.lower())
    else:
        connecting_words.append(word.lower())
print(connecting_words)
total_files = len(files_with_index)
zeroes_and_ones = []
zeroes_and_ones_of_all_words = []
for word in (different_words):
    if word.lower() in unique_words_all:
        zeroes_and_ones = [0] * total_files
        linkedlist = linked_list_data[word].head
        print(word)
        while linkedlist.nextval is not None:
            zeroes_and_ones[linkedlist.nextval.doc - 1] = 1
            linkedlist = linkedlist.nextval
        zeroes_and_ones_of_all_words.append(zeroes_and_ones)
    else:
        print(word," not found")
        sys.exit()
print(zeroes_and_ones_of_all_words)
for word in connecting_words:
    word_list1 = zeroes_and_ones_of_all_words[0]
    word_list2 = zeroes_and_ones_of_all_words[1]
    if word == "and":
        bitwise_op = [w1 & w2 for (w1,w2) in zip(word_list1,word_list2)]
        zeroes_and_ones_of_all_words.remove(word_list1)
        zeroes_and_ones_of_all_words.remove(word_list2)
        zeroes_and_ones_of_all_words.insert(0, bitwise_op);
    elif word == "or":
        bitwise_op = [w1 | w2 for (w1,w2) in zip(word_list1,word_list2)]
        zeroes_and_ones_of_all_words.remove(word_list1)
        zeroes_and_ones_of_all_words.remove(word_list2)
        zeroes_and_ones_of_all_words.insert(0, bitwise_op);
```

+ Code + Markdown | Run All | Clear Outputs of All Cells | Outline ··· Python 3.8.9 64-bit

```
        zeroes_and_ones_of_all_words.insert(0, bitwise_op);
    elif word == "or":
        bitwise_op = [w1 | w2 for (w1,w2) in zip(word_list1,word_list2)]
        zeroes_and_ones_of_all_words.remove(word_list1)
        zeroes_and_ones_of_all_words.remove(word_list2)
        zeroes_and_ones_of_all_words.insert(0, bitwise_op);
    elif word == "not":
        bitwise_op = [not w1 for w1 in word_list2]
        bitwise_op = [int(b == True) for b in bitwise_op]
        zeroes_and_ones_of_all_words.remove(word_list2)
        zeroes_and_ones_of_all_words.remove(word_list1)
        bitwise_op = [w1 & w2 for (w1,w2) in zip(word_list1,bitwise_op)]
        zeroes_and_ones_of_all_words.insert(0, bitwise_op);

files = []
print(zeroes_and_ones_of_all_words)
lis = zeroes_and_ones_of_all_words[0]
cnt = 1
for index in lis:
    if index == 1:
        files.append(files_with_index[cnt])
    cnt = cnt+1

print(files)
```

[7] Python

```
[]
database
retrieval
storage
[[1, 0, 0, 0, 1], [1, 0, 0, 1, 0], [0, 1, 0, 0, 1]]
[[1, 0, 0, 0, 1], [1, 0, 0, 1, 0], [0, 1, 0, 0, 1]]
['doc1.TXT', 'doc5.TXT']
```

+ Code + Markdown

Question 4

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

doc_1 ="Information Retrieval Systems is used with database systems"
doc_2 ="Information is in Storage Storage"
X=doc_1
Y=doc_2

# tokenization
X_list = word_tokenize(X)
Y_list = word_tokenize(Y)

# sw contains the list of stopwords
sw = stopwords.words('english')
l1 =[];
l2 =[]

# remove stop words from the string
X_set = {w for w in X_list if not w in sw}
Y_set = {w for w in Y_list if not w in sw}

# form a set containing keywords of both strings
rvector = X_set.union(Y_set)
for w in rvector:
    if w in X_set: l1.append(1) # create a vector
    else: l1.append(0)
    if w in Y_set: l2.append(1)
    else: l2.append(0)
c = 0

# cosine formula
for i in range(len(rvector)):
        c+= l1[i]*l2[i]
cosine = c / float((sum(l1)*sum(l2))*0.5)
print("Cosine similarity: ", cosine)
```

[8] Python

```
... Cosine similarity:  0.16666666666666666
```

Question 5

```
def dice_coefficient(a, b):
    """dice coefficient 2nt/(na + nb)."""
    if not len(a) or not len(b): return 0.0
    if len(a) == 1:  a=a+u'.'
    if len(b) == 1:  b=b+u'.'

    a_bigram_list=[]
    for i in range(len(a)-1):
      a_bigram_list.append(a[i:i+2])
    b_bigram_list=[]
    for i in range(len(b)-1):
      b_bigram_list.append(b[i:i+2])

    a_bigrams = set(a_bigram_list)
    b_bigrams = set(b_bigram_list)
    overlap = len(a_bigrams & b_bigrams)
    dice_coeff = overlap * 2.0/(len(a_bigrams) + len(b_bigrams))
    return dice_coeff

""" duplicate bigrams in a word should be counted distinctly
(per discussion), otherwise 'AA' and 'AAAA' would have a
dice coefficient of 1...
"""
def dice_coefficient(a,b):
    if not len(a) or not len(b): return 0.0
    """ quick case for true duplicates """
    if a == b: return 1.0
    """ if a != b, and a or b are single chars, then they can't possibly match """
    if len(a) == 1 or len(b) == 1: return 0.0

    """ use python list comprehension, preferred over list.append() """
    a_bigram_list = [a[i:i+2] for i in range(len(a)-1)]
    b_bigram_list = [b[i:i+2] for i in range(len(b)-1)]

    a_bigram_list.sort()
    b_bigram_list.sort()
```

```
    """ use python list comprehension, preferred over list.append() """
    a_bigram_list = [a[i:i+2] for i in range(len(a)-1)]
    b_bigram_list = [b[i:i+2] for i in range(len(b)-1)]

    a_bigram_list.sort()
    b_bigram_list.sort()

    # assignments to save function calls
    lena = len(a_bigram_list)
    lenb = len(b_bigram_list)
    # initialize match counters
    matches = i = j = 0
    while (i < lena and j < lenb):
        if a_bigram_list[i] == b_bigram_list[j]:
            matches += 2
            i += 1
            j += 1
        elif a_bigram_list[i] < b_bigram_list[j]:
            i += 1
        else:
            j += 1

    score = float(matches)/float(lena + lenb)
    return score

dice_coefficient("Digital Speech systems can be used in Synthesis and Systems", "Speech Filtering, Speech Retrieval systems are applications of Information Retrieval")
```

[9] Python

```
0.3404255319148936
```

Question 6

```
def Jaccard_Similarity(doc1, doc2):

    # List the unique words in a document
    words_doc1 = set(doc1.split())
    words_doc2 = set(doc2.split())

    # Find the intersection of words list of doc1 & doc2
    intersection = words_doc1.intersection(words_doc2)

    # Find the union of words list of doc1 & doc2
    union = words_doc1.union(words_doc2)

    # Calculate Jaccard similarity score
    # using length of intersection set divided by length of union set
    return float(len(intersection)) / len(union)

doc_1 = "Speech Filtering, Speech Retrieval systems are applications of Information Retrieval "
doc_2 = "Database Management system is used for storage storage"

Jaccard_Similarity(doc_1,doc_2)
```

[10] Python

```
0.0
```