

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Programme | : | **B.Tech** | Semester | : | **Win Sem 21-22** |
| Course | : | **Web Mining Lab** | Code | : | **CSE3024** |
| Faculty | : | **Dr.Bhuvaneswari A** | Slot | : | **L7+L8** |
| Date | : | **01-01-2022** | Marks | : | **10 Points** |

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

**GOOGLE COLAB LINK:**

<https://colab.research.google.com/drive/1D4RUXVIRX-RmkAI7NoC1PvWqKtaofcEe?usp=sharing>

**PROCEDURE:**

1. Install the necessary libraries (stop words, glob, nltk)
2. Create dictionary where words would be stored
3. Perform data cleaning on the file opened in the Document file path by removing uppercase letters
4. Tokenize the strings present in the text files and remove all the unnecessary words (in, is, a, the)
5. Add the tokenized words into the dictionary files\_with\_index{}
6. Calculate the word frequency by creating a dictionary word\_freq{}
7. Enter the query and send the query string into the \_init\_ function
8. The output is displayed which document contains the word present in the input query

Consider the following documents.

Doc 1 : Information Retrieval Systems is used with database systems

Doc 2 : Information is in 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 systems

1. Perform the text pre-processing of the given documents.
2. Construct a Boolean Model for the vocabulary list by considering documents 1, 2, 3,4 and 5.
   1. Retrieve the documents for the Boolean query “Information Retrieval Synthesis” using simple match.
   2. Retrieve the documents for the Boolean query “Database Retrieval Storage” using weighted match.(Rank the documents in the order of relevance)

**CODE:**

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

import nltk

nltk.download('stopwords')

Stopwords = set(stopwords.words('english'))

all\_words = []

dict\_global = {}

file\_folder = 'sample\_data/lab4/\*'

idx = 1

files\_with\_index = {}

for file in glob.glob(file\_folder):

    print(file)

    fname = file

    file = open(file , "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())

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

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

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 file in glob.glob(file\_folder):

    file = open(file, "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

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);

    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:

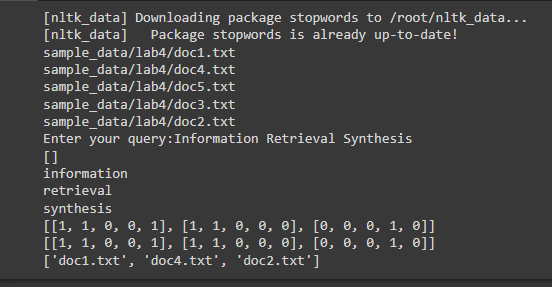
    if index == 1:

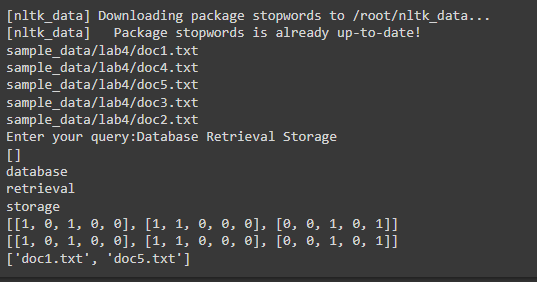
        files.append(files\_with\_index[cnt])

    cnt = cnt+1

print(files)

**OUTPUT:**

****

****

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

**CODE:**

import math

import nltk

nltk.download('punkt')

from textblob import TextBlob as tb

def tf(word, blob):

 return blob.words.count(word) / len(blob.words)

def n\_containing(word, bloblist):

 return sum(1 for blob in bloblist if word in blob.words)

def idf(word, bloblist):

  return math.log(len(bloblist) / (1 + n\_containing(word, bloblist)))

def tfidf(word, blob, bloblist):

 return tf(word, blob) \* idf(word, bloblist)

document1 = tb("""Information Retrieval Systems is used with database systems""")

document2 = tb("""Information is in Storage Storage""")

document3 = tb("""Digital Speech systems can be used in Synthesis and Systems """)

document4 = tb("""Speech Filtering, Speech Retrieval systems are applications of Information Retrieval """)

document5 = tb("""Database Management system is used for storage storage""")

bloblist = [document1, document2, document3, document4, document5]

for i, blob in enumerate(bloblist):

 print("Top words in document {}".format(i + 1))

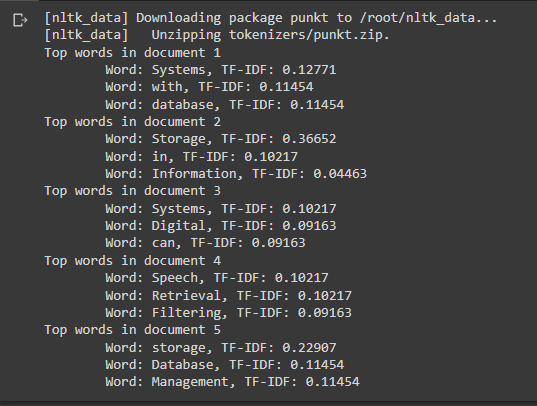
 scores = {word: tfidf(word, blob, bloblist) for word in blob.words}

 sorted\_words = sorted(scores.items(), key=lambda x: x[1], reverse=True)

 for word, score in sorted\_words[:3]:

    print("\tWord: {}, TF-IDF: {}".format(word, round(score,5)))

**OUTPUT:**

****

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

**CODE:**

doc1 = "Information Retrieval Systems is used with database systems"

doc2 = "Information is in Storage Storage"

documents = [doc1, doc2]

# Scikit Learn

from sklearn.feature\_extraction.text import CountVectorizer

import pandas as pd

# Create the Document Term Matrix

count\_vectorizer = CountVectorizer(stop\_words='english')

count\_vectorizer = CountVectorizer()

sparse\_matrix = count\_vectorizer.fit\_transform(documents)

# OPTIONAL: Convert Sparse Matrix to Pandas Dataframe if you want to see the word frequencies.

doc\_term\_matrix = sparse\_matrix.todense()

df = pd.DataFrame(doc\_term\_matrix,

 columns=count\_vectorizer.get\_feature\_names(),  index=['doc1', 'doc2'])

df

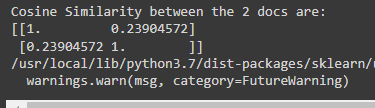
# Compute Cosine Similarity

from sklearn.metrics.pairwise import cosine\_similarity

print("Cosine Similarity between the 2 docs are: ")

print(cosine\_similarity(df, df))

**OUTPUT:**



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

**CODE:**

def dice\_coefficient(a, b):

    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

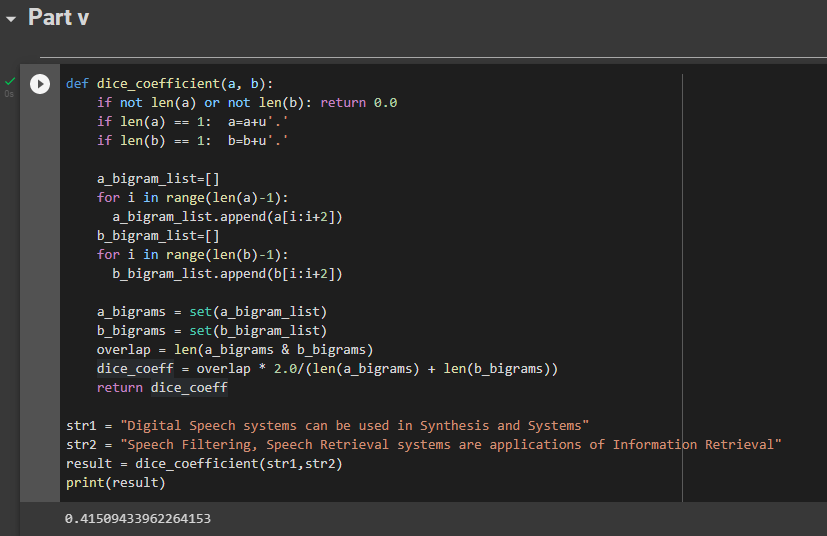
str1 = "Digital Speech systems can be used in Synthesis and Systems"

str2 = "Speech Filtering, Speech Retrieval systems are applications of Information Retrieval"

result = dice\_coefficient(str1,str2)

print(result)

**OUTPUT:**

****

1. Compute the Jaccard co-efficientbetween docs 4 and docs 5.

**CODE:**

A ="information Retrieval systems is used with database systems"

B ="Information is in Storage Storage"

def get\_jaccard\_sim(str1, str2):

    a = set(str1.split())

    b = set(str2.split())

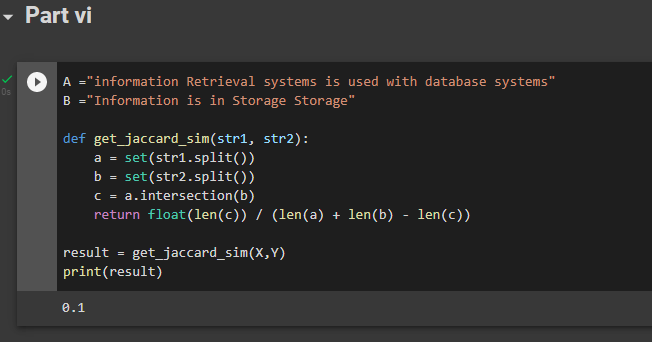
    c = a.intersection(b)

    return float(len(c)) / (len(a) + len(b) - len(c))

result = get\_jaccard\_sim(X,Y)

print(result)

**OUTPUT:**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*