VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Programme	:	**B.Tech**	Semester	:	**Win Sem 21-22**
Course	:	**Web Mining**	Code	:	**CSE3024**
Faculty	:	**Dr.Bhuvaneswari A**	Slot	:	**L7+L8**
Date	:	**03-02-2022**	Marks	:	**10 Points**

Register Number : 19BCE1712

Name : Tanay Vaishnav

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.**

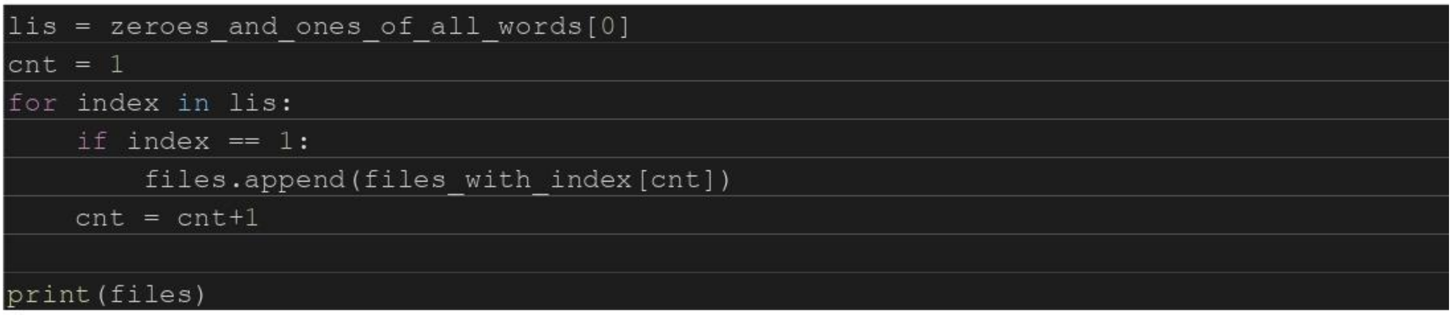
a. **Retrieve the documents for the Boolean query “Information Retrieval Synthesis” using simple match.**

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

Code:

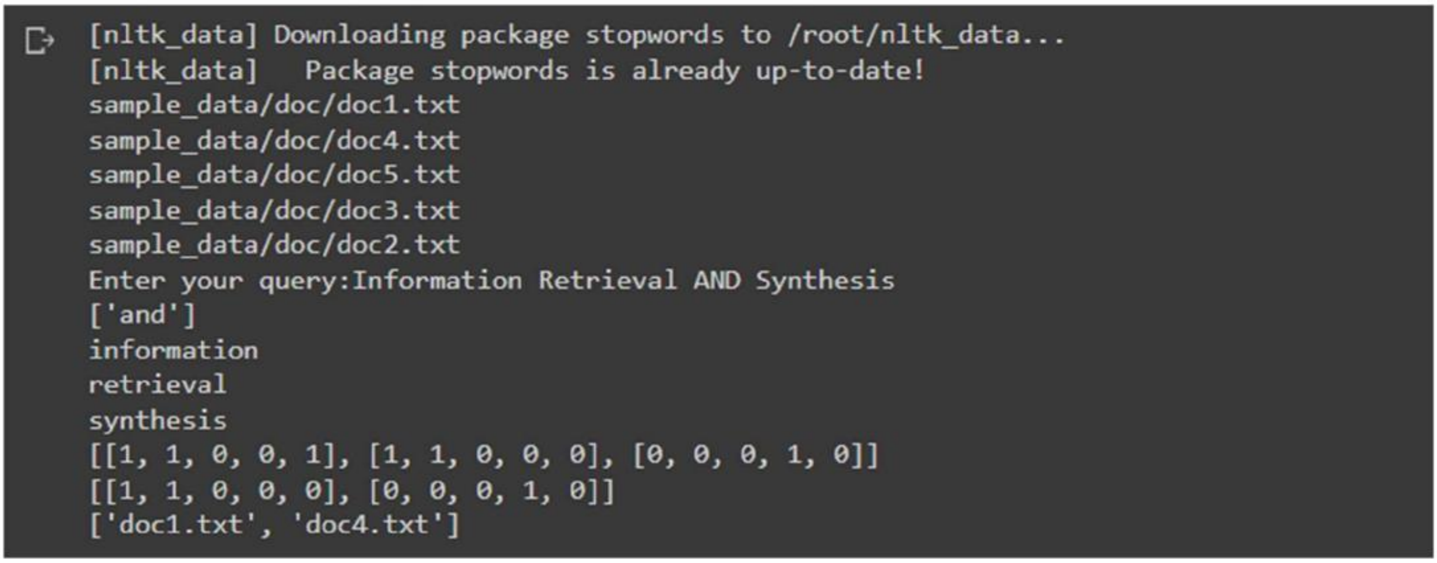
```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.tokenize import sent_tokenize , word_tokenize
import glob
import re
import os
import numpy as np
import sys
import nltk
nltk.download('stopwords')
Stopwords = set(stopwords.words('english'))
all_words = []
dict_global = {}
file_folder = 'sample_data/doc/*'
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)
```

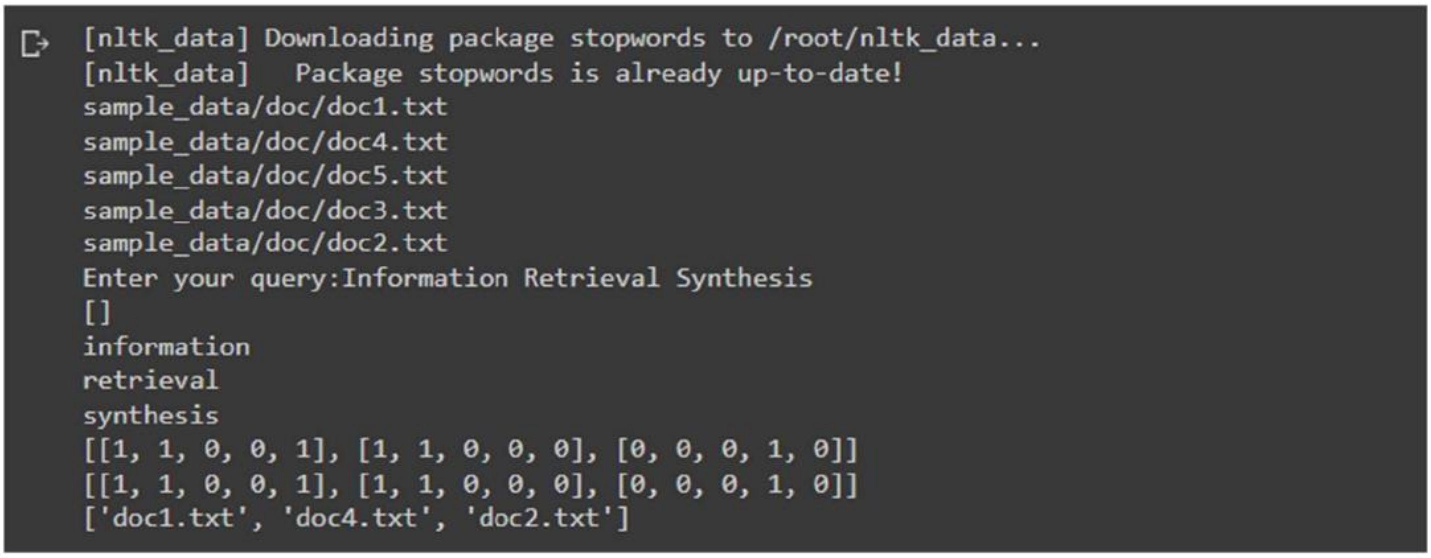


```
lis = zeroes_and_ones_of_all_words[0]
cnt = 1
for index in lis:
    if index == 1:
        files.append(files_with_index[cnt])
    cnt = cnt+1

print(files)
```

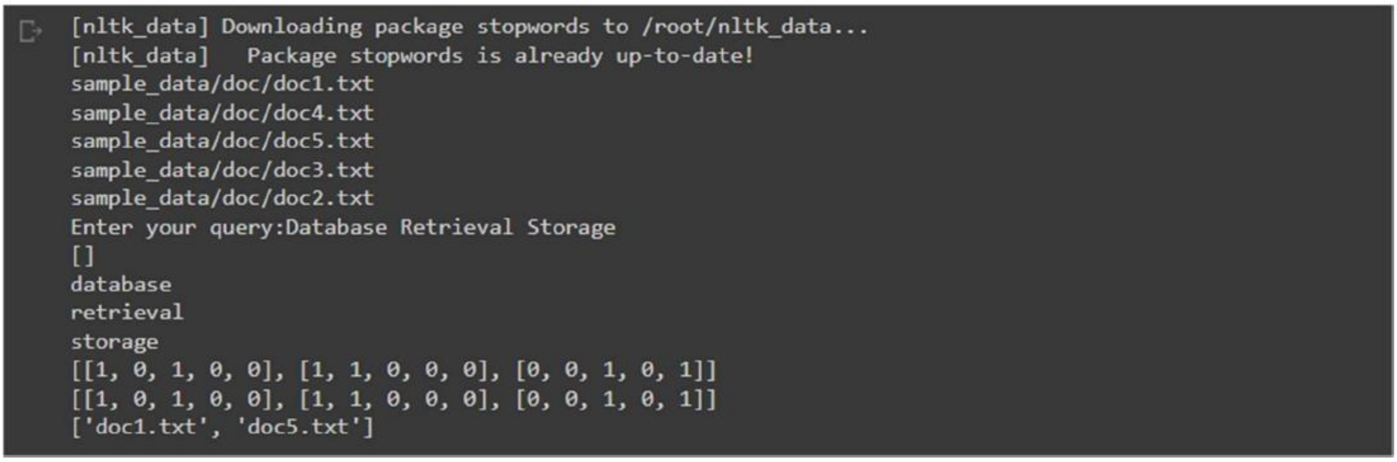
Output:



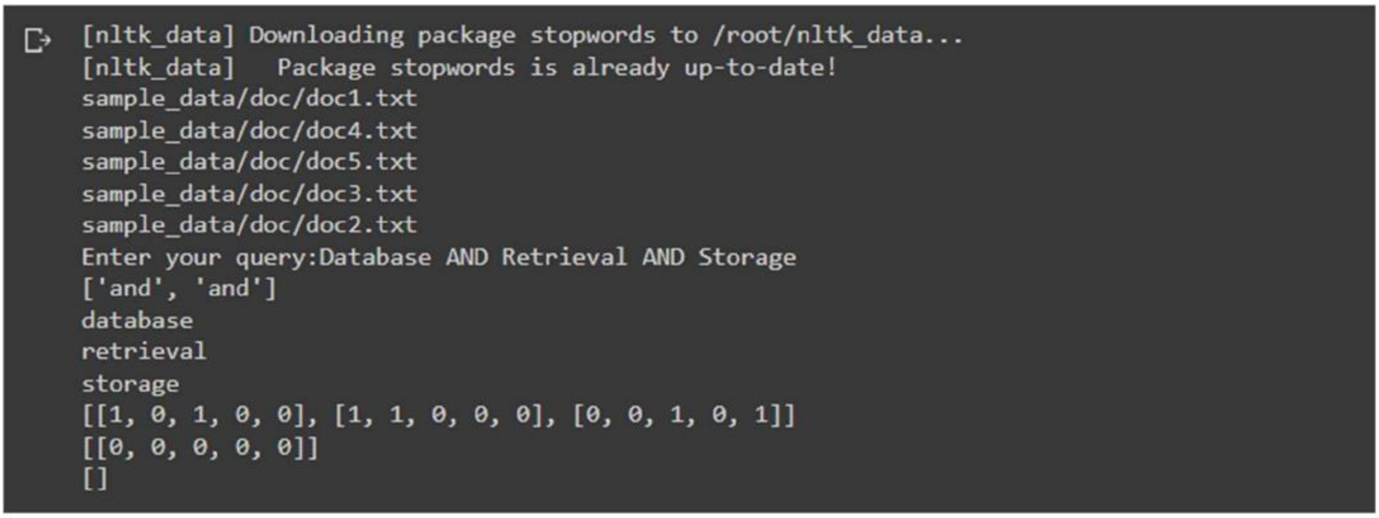
```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
sample_data/doc/doc1.txt
sample_data/doc/doc4.txt
sample_data/doc/doc5.txt
sample_data/doc/doc3.txt
sample_data/doc/doc2.txt
Enter your query:Information Retrieval AND Synthesis
['and']
information
retrieval
synthesis
[[1, 1, 0, 0, 1], [1, 1, 0, 0, 0], [0, 0, 0, 1, 0]]
[[1, 1, 0, 0, 0], [0, 0, 0, 1, 0]]
['doc1.txt', 'doc4.txt']
```



```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
sample_data/doc/doc1.txt
sample_data/doc/doc4.txt
sample_data/doc/doc5.txt
sample_data/doc/doc3.txt
sample_data/doc/doc2.txt
Enter your query:Information Retrieval Synthesis
[]
information
retrieval
synthesis
[[1, 1, 0, 0, 1], [1, 1, 0, 0, 0], [0, 0, 0, 1, 0]]
[[1, 1, 0, 0, 1], [1, 1, 0, 0, 0], [0, 0, 0, 1, 0]]
['doc1.txt', 'doc4.txt', 'doc2.txt']
```



```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
sample_data/doc/doc1.txt
sample_data/doc/doc4.txt
sample_data/doc/doc5.txt
sample_data/doc/doc3.txt
sample_data/doc/doc2.txt
Enter your query:Database Retrieval Storage
[]
database
retrieval
storage
[[1, 0, 1, 0, 0], [1, 1, 0, 0, 0], [0, 0, 1, 0, 1]]
[[1, 0, 1, 0, 0], [1, 1, 0, 0, 0], [0, 0, 1, 0, 1]]
['doc1.txt', 'doc5.txt']
```

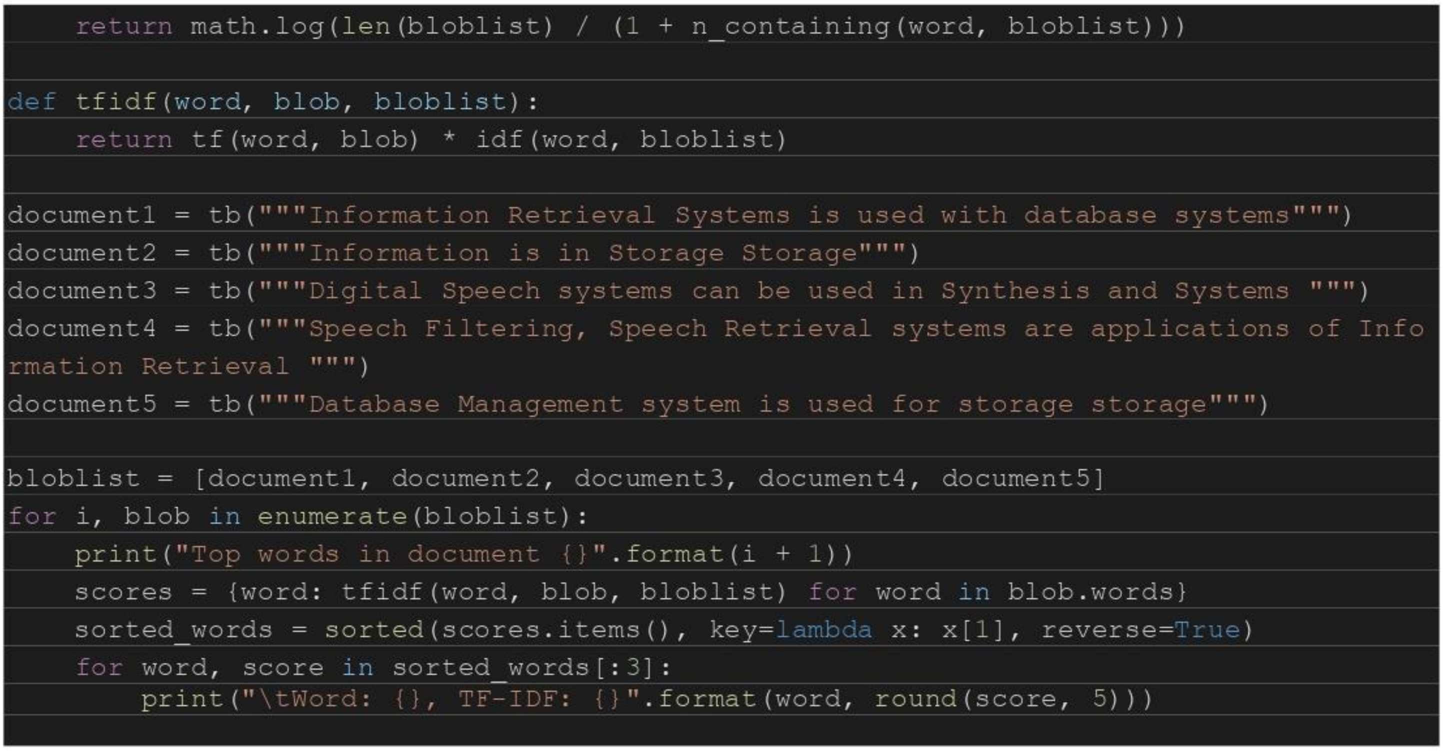


```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
sample_data/doc/doc1.txt
sample_data/doc/doc4.txt
sample_data/doc/doc5.txt
sample_data/doc/doc3.txt
sample_data/doc/doc2.txt
Enter your query:Database AND Retrieval AND Storage
['and', 'and']
database
retrieval
storage
[[1, 0, 1, 0, 0], [1, 1, 0, 0, 0], [0, 0, 1, 0, 1]]
[[0, 0, 0, 0, 0]]
[]
```

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

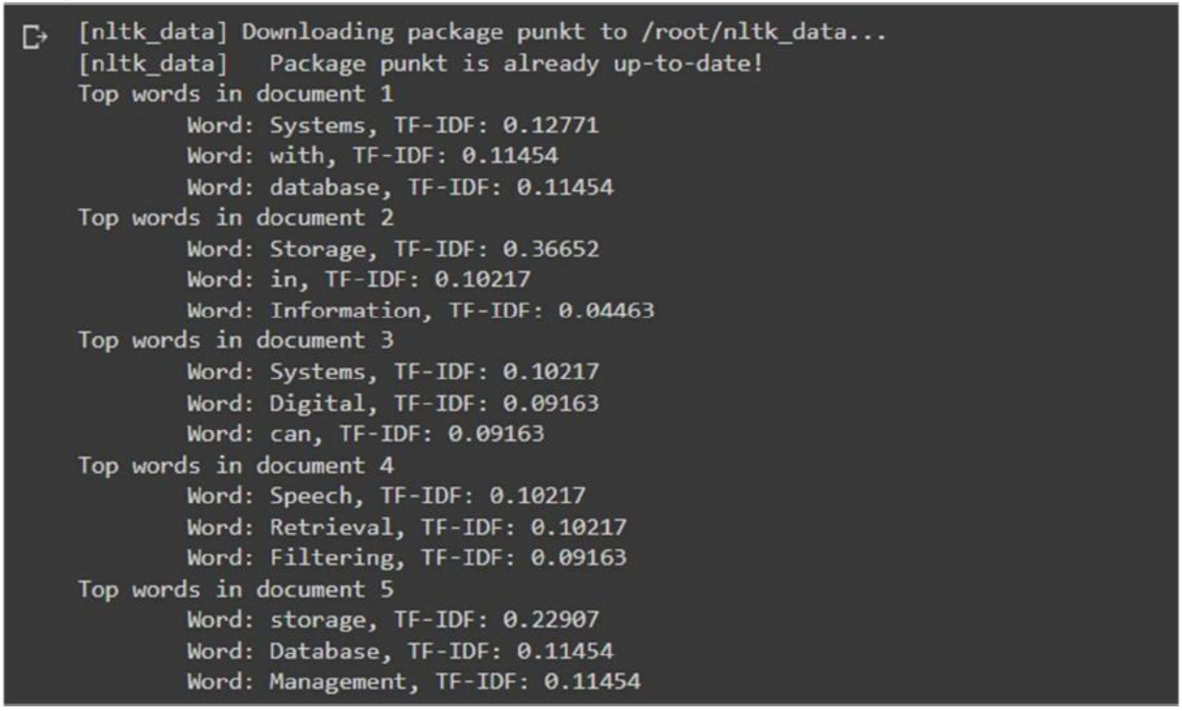
Code:

```
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:



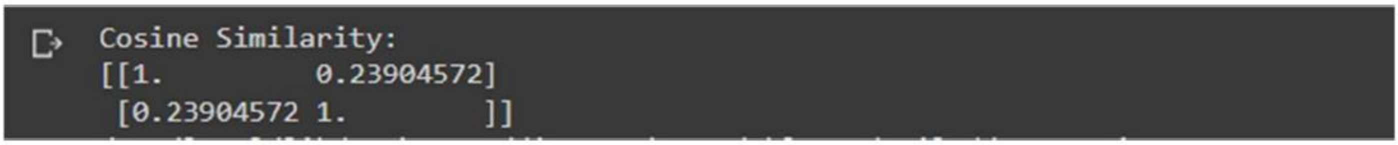
```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Top words in document 1
        Word: Systems, TF-IDF: 0.12771
        Word: with, TF-IDF: 0.11454
        Word: database, TF-IDF: 0.11454
Top words in document 2
        Word: Storage, TF-IDF: 0.36652
        Word: in, TF-IDF: 0.10217
        Word: Information, TF-IDF: 0.04463
Top words in document 3
        Word: Systems, TF-IDF: 0.10217
        Word: Digital, TF-IDF: 0.09163
        Word: can, TF-IDF: 0.09163
Top words in document 4
        Word: Speech, TF-IDF: 0.10217
        Word: Retrieval, TF-IDF: 0.10217
        Word: Filtering, TF-IDF: 0.09163
Top words in document 5
        Word: storage, TF-IDF: 0.22907
        Word: Database, TF-IDF: 0.11454
        Word: Management, TF-IDF: 0.11454
```

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

Code:

```
# Define the documents
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 fre
quencies.
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: ")
print(cosine_similarity(df, df))
```

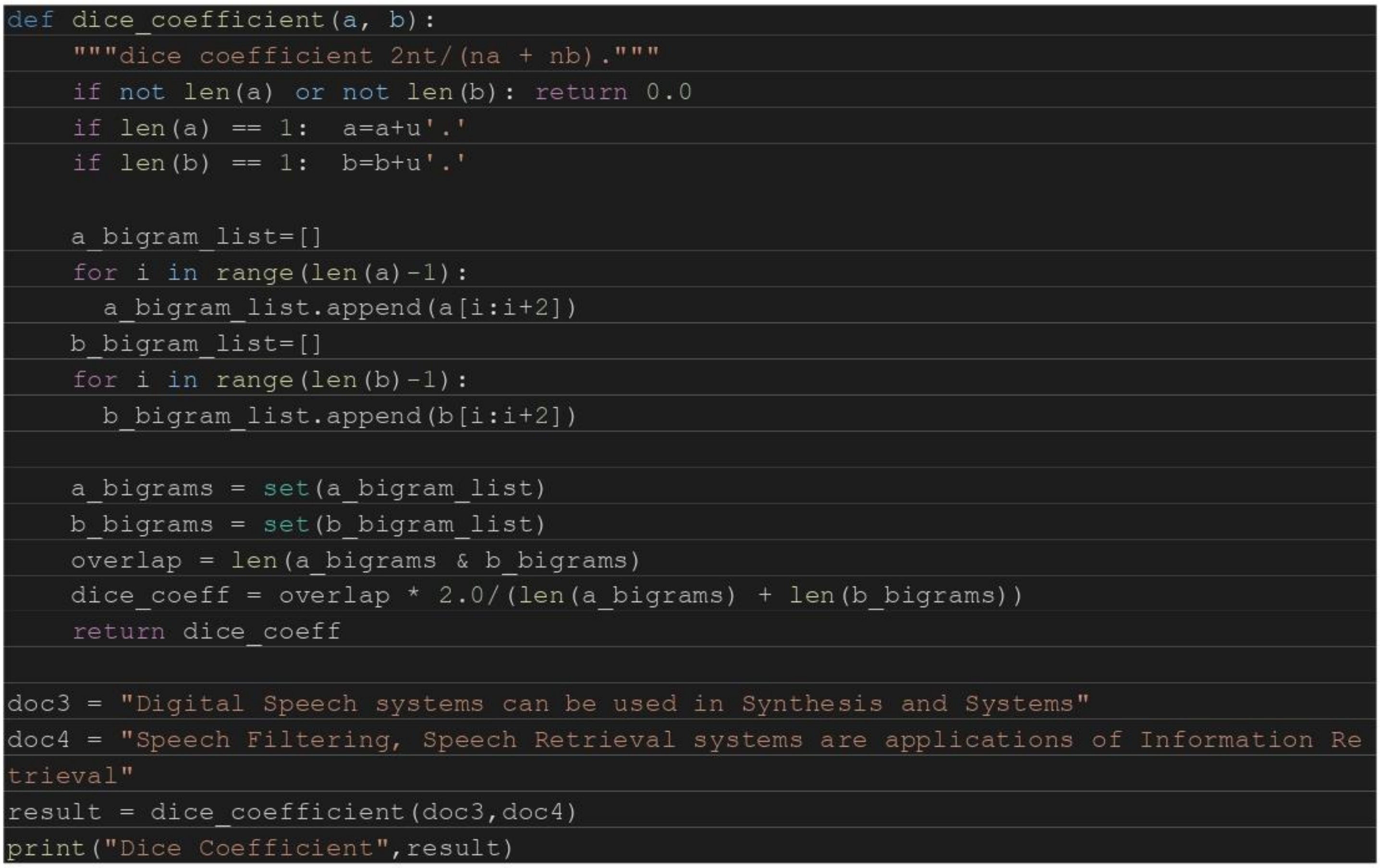
Output:



```
Cosine Similarity:
[[1.         0.23904572]
 [0.23904572 1.        ]]
```

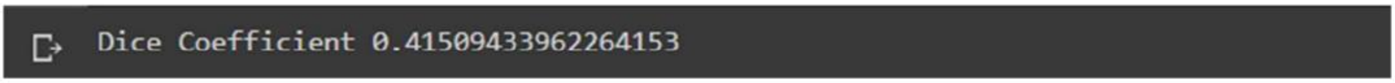
v. Compute Dice Co-efficient between docs 3 and docs 4.

Code:



```
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

doc3 = "Digital Speech systems can be used in Synthesis and Systems"
doc4 = "Speech Filtering, Speech Retrieval systems are applications of Information Re
trieval"
result = dice_coefficient(doc3,doc4)
print("Dice Coefficient",result)
```

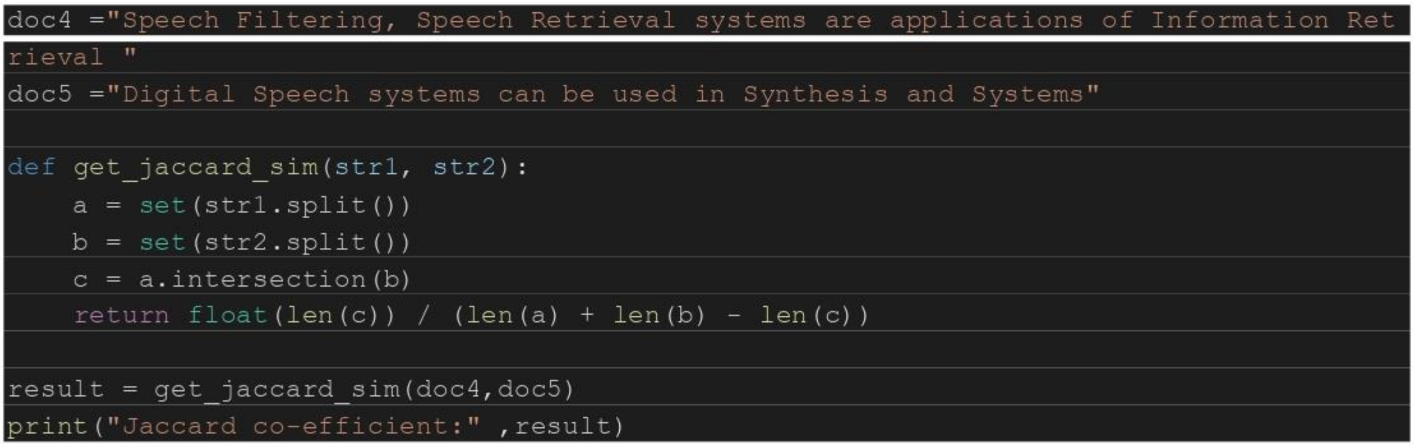
Output:



```
Dice Coefficient 0.41509433962264153
```

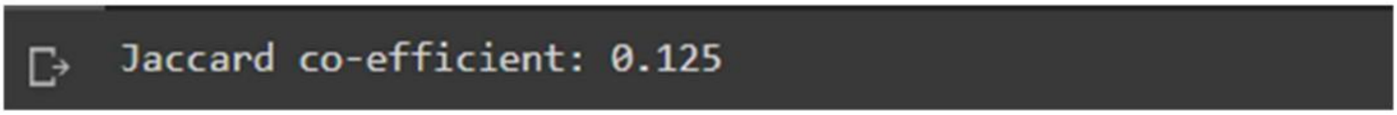
vi. Compute the Jaccard co-efficient between docs 4 and docs 5.

Code:



```
doc4 ="Speech Filtering, Speech Retrieval systems are applications of Information Retrieval "
doc5 ="Digital Speech systems can be used in Synthesis and Systems"

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(doc4,doc5)
print("Jaccard co-efficient:" ,result)
```

Output:



```
Jaccard co-efficient: 0.125
```