**CIS-660 Lab 2**

**Submitted By:**

A person standing in front of a bridge

Description automatically generated with medium confidence

**Name: Mohamed Gani Mohamed Sulthan**

**CSU ID: 2811619**

**Grail Id: momohame@grail.eecs.csuohio.edu**

**Master’s in computer science**

**Part 1: Preprocessing to Build Document Vectors for Web Page Content Analysis**

**Process:**

Step 1: In the given document we need to access the six following links and scrapping the data from the website

Step 2: From the content, we are removing all the unnecessary symbols from the content

Step 3: In the content we need to find the term frequency of given words

Step 4: For the single word we can directly use the numpy to group the data

Step 5: For Bigram words, I used nltk(Natural Language Tool Kit) to access the content and made the group of near by words and stored it in to the stemmed list. From the list I found the count of the repeated words

Step 6: Print the word counts

**Code:**

**# To collect the term frequency**

'''

CIS-660 Lab 2 Assignment

Author: Mohamed Gani Mohamed Sulthan

CSU ID: 2811619

Major: Master’s in computer science

'''

# Import all the required modules

import requests

import string

import nltk

import re

import operator

import requests

import numpy as np

import pandas as pd

import os

from bs4 import BeautifulSoup

from collections import Counter

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from requests.adapters import HTTPAdapter

from urllib3.util.retry import Retry

from sklearn.metrics.pairwise import cosine\_similarity

from scipy import spatial

# Define the location of the directory

path =r"C:/Users/Gani/Desktop/Cleveland State University/Third Sem/Data Mining/Lab/Lab 2/"

# Change the directory

os.chdir(path)

list = []

wordsCountAll = []

stemWordList = []

def read\_files(file\_path):

with open(file\_path, 'r', encoding="utf8") as file:

contents = file.read()

# Breaking into lines and remove leading and trailing space and multi-headlines and blank lines

lines = (line.strip() for line in contents.splitlines())

chunks = (phrase.strip() for line in lines for phrase in line.split(" "))

text = '\n'.join(chunk for chunk in chunks if chunk)

# Removing hyper link and regular expressions

hyperlink\_removed\_text=re.sub(r'https?:\/\/.\*[\r\n]\*', '',text)

te=text.translate(str.maketrans('', '', string.punctuation))

pattern = r'[0-9]'

new\_string = re.sub(pattern, '', te)

new\_string =new\_string.lower()

# Making the bigrams of the word by combining it

bigram\_word1=re.sub('machine learning',' machinelearning ', new\_string)

bigram\_word2=re.sub('deep learning',' deeplearning ', bigram\_word1)

bigram\_word3=re.sub('data mining',' datamining ', bigram\_word2)

# Removing unnecessary special characters and Splitting the details and put into list

final= re.sub("\s\s+", " ", bigram\_word3)

final2= re.sub("–", "", final)

final3=re.sub('[^a-zA-Z0-9 \n\.]', ' ',final2)

words\_list = final3.split()

# Removing stopwords and putting into the filtered\_words

filtered\_words = [word for word in words\_list if word not in stopwords.words('english')]

# stemming

ps = PorterStemmer()

words = word\_tokenize(new\_text)

for w in words:

wordlist.append(w)

stemWordList.append(ps.stem(w))

# Counting the values using pandas and matcing with the documents

count = pd.value\_counts(np.array(filtered\_words))

words\_find = ['research','data','mining', 'analytics', 'datamining', 'machinelearning', 'deeplearning']

wordscount = []

doc = 0

for i in words\_find:

print(i, "appears", filtered\_words.count(i), "times" )

wordscount.append(filtered\_words.count(i))

wordsCountAll.append(wordscount.copy())

print("Total words count for 6 six files for seven words in an array", wordsCountAll)

if \_\_name\_\_ == "\_\_main\_\_":

# Iterate over all the files in the directory

for file in os.listdir():

if file.endswith('.txt'):

# Create the filepath of particular file

file\_path =f"{path}/{file}"

read\_files(file\_path)

**Output:**

Graphical user interface, text, application

Description automatically generated

**Part 2: Data Transformation for Topic Analysis of Documents (Webpages)**

**Process:**

Step 1: From the written output, calling the variable in to this program and created the DF

Step 2: In the DataFrame, called the values in rows and columns

Step 3: The row has the document details and column has the found text frequency

Step 4: Calling the sklearn.metrices.pairwise and importing the cosine similarity

Step 5: Printing the cosine similarity values in the list

**Code:**

df = pd.DataFrame(wordsCountAll, columns = ['research','data','mining', 'analytics', 'datamining', 'machinelearning', 'deeplearning'])

df.index = ['doc1','doc2','doc3','doc4','doc5','doc6']

df

cos\_df = pd.DataFrame(cosine\_similarity(df))

cos\_df.index = ['doc1','doc2','doc3','doc4','doc5','doc6']

cos\_df.columns = ['doc1','doc2','doc3','doc4','doc5','doc6']

display(cos\_df)

**Output:**

A picture containing graphical user interface

Description automatically generated

**Part3: Analysis and Discussion of Problems**

1. **Discuss briefly about your topic analysis with your cosine similarity matrix focusing on that: Whether each value (in Cosine Sim) of each pair of any two docs indicate the similarity correctly?**

**Solution:**

To compute the cosine similarity I used **from sklearn.metrics.pairwise import cosine\_similarity.** It’s performed for the dataframe consisting of seven given topics to the six documents. The six \* six document matrices consist of cosine similarity. All the document pairs have the same word.

Doc4 and Doc5 indicated the cosine similarity function more correctly.

1. **Which 2 docs are most similar in terms of 7 given topics?**

**Solution:**

Doc4 and Doc5 are the exact similar in terms of the seven given topics

Doc2 and Doc3 are the most similar in terms of the seven given topics

1. **The Topics of Doc6 is similar to the Topics of Doc 4 and 5? Explain Why or Why Not in terms of 7 TFs? If not, what are the reasons?**

**Solution:**

The topics in the doc6 is not exactly similar to the topics presented in the doc4 and doc5. But all the seven topics are presented in all the three documents. Some of the topics term frequencies are higher in doc4 and doc5. The least matching is deeplearning which is almost same to the doc4 and doc5.

**Extra Credit**

You can create a Term Dictionary with TF (and DF) in a Table format or in MongoDB with the scheme. Your Term Dictionary with TF and DF would look like either one of those tables below.

**Process:**

Step 1: Calling all the documents in the main and saving the contents in the string

Step 2: Splitting the documents in to the each word. So each words have their own string

Step 3: Joined the words to remove the common duplicate words

Step 4: Storing the words in the dictionary and visualize the dataframe

Step 5: Running the saved dictionary in to the term frequency function and and visualize it in the dataframe

Step 6: Removing the stopwords to compute IDF. Visualizing it in the dataframe after the computation of IDF and TFIDF

Step 7: Calling the TfidfVectorizer to find the vectorization for the document

**Code:**

'''

Credits: https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558

CIS-660 Lab 2 Assignment Extra Credit

Author: Mohamed Gani Mohamed Sulthan

CSU ID: 2811619

Major: Master’s in computer science

'''

import pandas as pd

import sklearn as sk

import math

import os

import re

import string

from nltk.corpus import stopwords

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

def document\_freq(file\_content):

contents = file\_content.read()

# Breaking into lines and remove leading and trailing space and multi-headlines and blank lines

lines = (line.strip() for line in contents.splitlines())

chunks = (phrase.strip() for line in lines for phrase in line.split(" "))

text = '\n'.join(chunk for chunk in chunks if chunk)

# Removing hyper link and regular expressions

hyperlink\_removed\_text=re.sub(r'https?:\/\/.\*[\r\n]\*', '',text)

te=text.translate(str.maketrans('', '', string.punctuation))

pattern = r'[0-9]'

new\_text = re.sub(pattern, '', te)

lowerCaseText =new\_string.lower()

# Making the bigrams of the word by combining it

bigram\_word1=re.sub('machine learning',' machinelearning ', lowerCaseText)

bigram\_word2=re.sub('deep learning',' deeplearning ', bigram\_word1)

bigram\_word3=re.sub('data mining',' datamining ', bigram\_word2)

# Removing unnecessary special characters and Splitting the details and put into list

final= re.sub("\s\s+", " ", bigram\_word3)

final2= re.sub("–", "", final)

final3=re.sub('[^a-zA-Z0-9 \n\.]', ' ',final2)

return final2

def computeTF(wordDict, doc):

tfDict = {}

corpusCount = len(doc)

for word, count in wordDict.items():

tfDict[word] = count/float(corpusCount)

return(tfDict)

def computeIDF(docList):

idfDict = {}

N = len(docList)

idfDict = dict.fromkeys(docList[0].keys(), 0)

for word, val in idfDict.items():

idfDict[word] = math.log10(N / (float(val) + 1))

return(idfDict)

def computeTFIDF(tfBow, idfs):

tfidf = {}

for word, val in tfBow.items():

tfidf[word] = val\*idfs[word]

return(tfidf)

if \_\_name\_\_ == "\_\_main\_\_":

doc1 = open(r"C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\website1.txt", "r", encoding="utf8")

first\_document = document\_freq(doc1)

doc2 = open(r"C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\website2.txt", "r", encoding="utf8")

second\_document = document\_freq(doc2)

doc3 = open(r"C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\website3.txt", "r", encoding="utf8")

third\_document = document\_freq(doc3)

doc4 = open(r"C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\website4.txt", "r", encoding="utf8")

fourth\_document = document\_freq(doc4)

doc5 = open(r"C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\website5.txt", "r", encoding="utf8")

fifth\_document = document\_freq(doc5)

doc6 = open(r"C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\website6.txt", "r", encoding="utf8")

sixth\_document = document\_freq(doc6)

#split so each word have their own string

first\_document\_split = first\_document.split(" ")

second\_document\_split = second\_document.split(" ")

third\_document\_split = third\_document.split(" ")

fourth\_document\_split = fourth\_document.split(" ")

fifth\_document\_split = fifth\_document.split(" ")

sixth\_document\_split = sixth\_document.split(" ")

#join them to remove common duplicate words

total= set(first\_document\_split).union(set(second\_document\_split), set(third\_document\_split), set(fourth\_document\_split),

set(fifth\_document\_split), set(sixth\_document\_split))

wordDictA = dict.fromkeys(total, 0)

wordDictB = dict.fromkeys(total, 0)

wordDictC = dict.fromkeys(total, 0)

wordDictD = dict.fromkeys(total, 0)

wordDictE = dict.fromkeys(total, 0)

wordDictF = dict.fromkeys(total, 0)

for word in first\_document\_split:

wordDictA[word]+=1

for word in second\_document\_split:

wordDictB[word]+=1

for word in third\_document\_split:

wordDictC[word]+=1

for word in fourth\_document\_split:

wordDictD[word]+=1

for word in fifth\_document\_split:

wordDictE[word]+=1

for word in sixth\_document\_split:

wordDictF[word]+=1

df = pd.DataFrame([wordDictA, wordDictB, wordDictC, wordDictD, wordDictE, wordDictF])

print("Dictionary key-value pairing visualization")

display(df)

df.to\_csv(r'C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\key\_value\_visualization.csv', encoding='utf-8')

#running our sentences through the tf function:

tfFirst = computeTF(wordDictA, first\_document\_split)

tfSecond = computeTF(wordDictB, second\_document\_split)

tfThird = computeTF(wordDictC, third\_document\_split)

tfFourth = computeTF(wordDictD, fourth\_document\_split)

tfFifth = computeTF(wordDictE, fifth\_document\_split)

tfSixth = computeTF(wordDictF, sixth\_document\_split)

#Converting to dataframe for visualization

tf = pd.DataFrame([tfFirst, tfSecond, tfThird, tfFourth, tfFifth, tfSixth])

print("Term Frequency visualization")

display(tf)

print("Seven Data Represntation:")

print(tf[['research','data', 'machinelearning']])

tf.to\_csv(r'C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\term\_frequency\_visualization.csv', encoding='utf-8')

# Doing stopwords and taking the word in tothe dictionary and matching with other elements

stop\_words = set(stopwords.words('english'))

filtered\_sentence = [w for w in wordDictA if not w in stop\_words]

#inputing our sentences in the log file

idfs = computeIDF([wordDictA, wordDictB, wordDictC, wordDictD, wordDictE, wordDictF])

idfFirst = computeTFIDF(tfFirst, idfs)

idfSecond = computeTFIDF(tfSecond, idfs)

idfThird = computeTFIDF(tfThird, idfs)

idfFourth = computeTFIDF(tfFourth, idfs)

idfFifth = computeTFIDF(tfFifth, idfs)

idfSixth = computeTFIDF(tfSixth, idfs)

#putting it in a dataframe

idf= pd.DataFrame([idfFirst, idfSecond, idfThird, idfFourth, idfFifth, idfSixth])

print("Visualization after computing IDF and TFIDF")

display(idf)

idf.to\_csv(r'C:\Users\Gani\Desktop\Cleveland State University\Third Sem\Data Mining\Lab\Lab 2\IDF\_TFIDF\_visualization.csv', encoding='utf-8')

#calling the TfidfVectorizer

vectorize= TfidfVectorizer()

firstV = first\_document.lower()

secondV = second\_document.lower()

thirdV = second\_document.lower()

fourthV = second\_document.lower()

fifthV = second\_document.lower()

sixthV = second\_document.lower()

#fitting the model and passing our document right away:

response= vectorize.fit\_transform([firstV, secondV, thirdV, fourthV, fifthV, sixthV])

print("After vectorization")

print(response)

**Output:**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated