

**TWIA**

**School of Engineering and**

**Technology Department of**

**Computer Science Engineering**

**LAB MANUAL**

Submitted By**: Rahul Dhawan**

Class: **VIII-C**

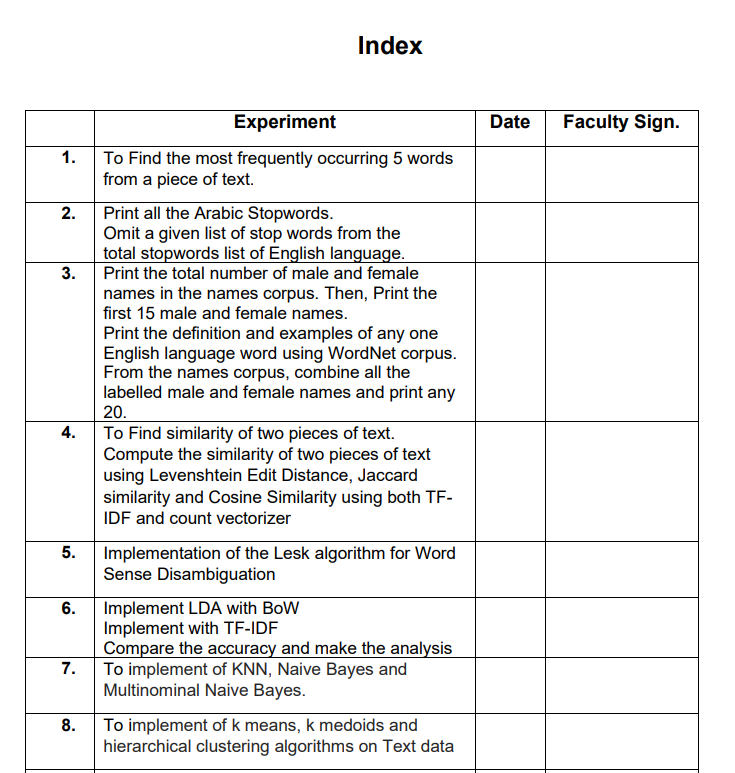
Roll Number: **17CSU146**

Taught By: **Ms. Vaishali Kalra**

Git-hub Link: **https://github.com/Crypto-Matrix/NLP-LAB**

**The Northcap University**

**Gurugram, Haryana**



28-4

21-4

14-4

24-3

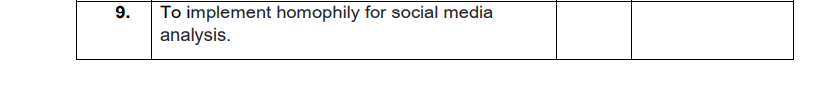
17-3

10-03

24-02

5-02

12-02





**TWIA Lab Manual (CSL 554)**

**2020-21**

**EXPERIMENT NO. 1**

|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 12 Feb 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective:** To Find the most frequently occurring 5 words from a piece of text. |
| **Outcome:** Students will be able to demonstrate how to find stopwords from corpus. |
| **Problem Statement:** Select the documents and find most commonly occurring 5 words.  **Code**  import nltk  from nltk.corpus import stopwords  # print(stopwords.words('english'))  from nltk.tokenize import word\_tokenize  txt="hello mr. smith, how are you doing today? the weather is great and pyhton is awesome. the sky is pinkish-blue, don't eat cardboard"  stop\_words = set(stopwords.words('english'))  word\_tokens = word\_tokenize(txt)  result=[]  for w in word\_tokens:  if w not in stop\_words:  result.append(w)    # print(word\_tokens)  # print('filtered sentence=', result)  fd = nltk.FreqDist(result)  print(fd.most\_common(5))  Output: |
|  |



**TWIA Lab Manual (CSL 554)**

**2020-21**

**EXPERIMENT NO. 2**

|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 5 Feb 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective:** To  a) Print all the Arabic Stopwords.  b) Omit a given list of stop words from the total stopwords list of English language. |
| **Outcome:** Students will be able to demonstrate how to print and omit stopwords. |
| **Problem Statement:** Select the documents and omit the given list of stopwords.  **Code (a)**  from nltk.corpus import stopwords  stopwords\_list = stopwords.words('arabic')  print(stopwords\_list)    **Output**    **Code (b)**  import nltk  nltk.download("stopwords")  from nltk.corpus import stopwords  english\_stopwords = set(stopwords.words('english'))  stop\_words = set(stopwords.words('english')) - set(['again', 'once', 'from'])  stop\_words  **Output** |
|  |



**TWIA Lab Manual (CSL 554)**

**2020-21**

**EXPERIMENT NO. 3**

|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 5 Feb 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective:** To  a) Print the total number of male and female names in the names corpus. Then, Print the first 15 male and female names.  b) Print the definition and examples of any one English language word using WordNet corpus. |
| **Outcome:** Students will be able to demonstrate how to use WordNet corpus. |
| **Problem Statement:** Select the documents and execute above statements.  **Code (a)**  from nltk.corpus import names  nltk.download('names')  print("\nNumber of male names:")  print (len(names.words('male.txt')))  print("\nNumber of female names:")  print (len(names.words('female.txt')))  male\_names = names.words('male.txt')  female\_names = names.words('female.txt')  print("\nFirst 10 male names:")  print (male\_names[0:15])  print("\nFirst 10 female names:")  print (female\_names[0:15])  **Output**    **Code (b)**  from nltk.corpus import wordnet  nltk.download('wordnet')  syns = wordnet.synsets("Education")  print("Defination of the said word:")  print(syns[0].definition())  print("\nExamples of the word in use::")  print(syns[0].examples())  **Output** |



**TWIA Lab Manual (CSL 554)**

**2020-21**

**EXPERIMENT NO. 4**

|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 12 Feb 2021** |
| **Faculty Signature:** |
| **Grade:** |

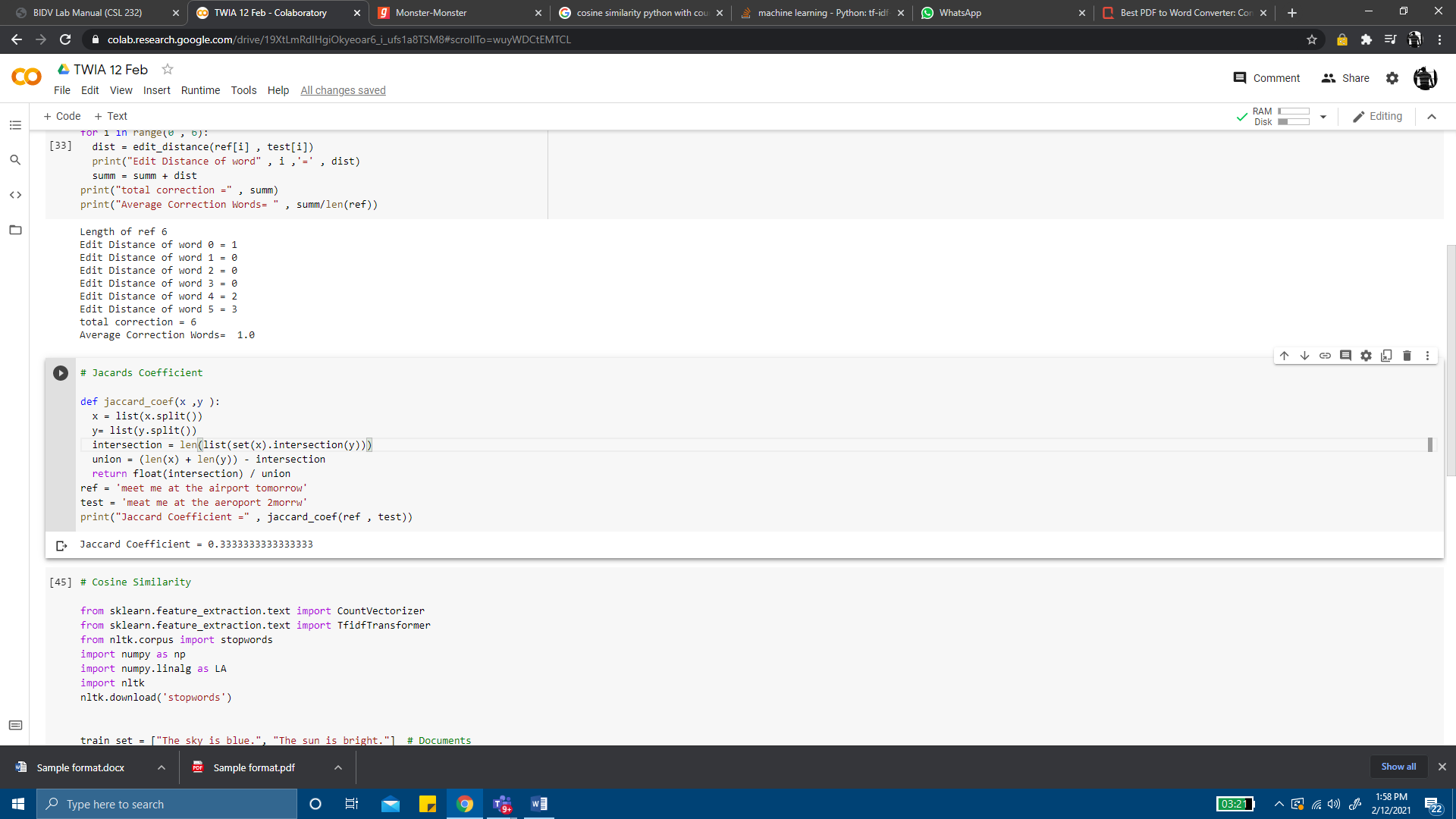
|  |
| --- |
| **Objective:** To implement Levenshtein Edit Distance , Jaccard similarity , Cosine Similarity using both TF-IDF and count vectorizer |
| **Outcome:** Students will be able to demonstrate Levenshtein Edit Distance , Jaccard similarity , Cosine Similarity using both TF-IDF and count vectorizer |
| **Problem Statement:** Select the documents and implement above similarities methods.  **Code (Jaccard Similarity)**  def jaccard\_coef(x ,y ):    x = list(x.split())    y= list(y.split())    intersection = len(list(set(x).intersection(y)))    union = (len(x) + len(y)) - intersection    return float(intersection) / union  ref = 'meet me at the airport tomorrow'  test = 'meat me at the aeroport 2morrw'  print("Jaccard Coefficient =" , jaccard\_coef(ref , test)) |



**TWIA Lab Manual (CSL 554)**

**2020-21**

**OUTPUT**



**CODE (Levenshtein Edit Distance)**

def edit\_distance(s1, s2):

    m=len(s1)+1

    n=len(s2)+1

    tbl = {}

    for i in range(m): tbl[i,0]=i

    for j in range(n): tbl[0,j]=j

    for i in range(1, m):

        for j in range(1, n):

            cost = 0 if s1[i-1] == s2[j-1] else 1

            tbl[i,j] = min(tbl[i, j-1]+1, tbl[i-1, j]+1, tbl[i-1, j-1]+cost)

    return (tbl[i,j])

ref = 'meet me at the airport tomorrow'

test = 'meat me at the aeroport 2morrw'

ref = ref.split()

test = test.split()

print("Length of ref" , len(ref))

summ=0

for i in range(0 , 6):

  dist = edit\_distance(ref[i] , test[i])

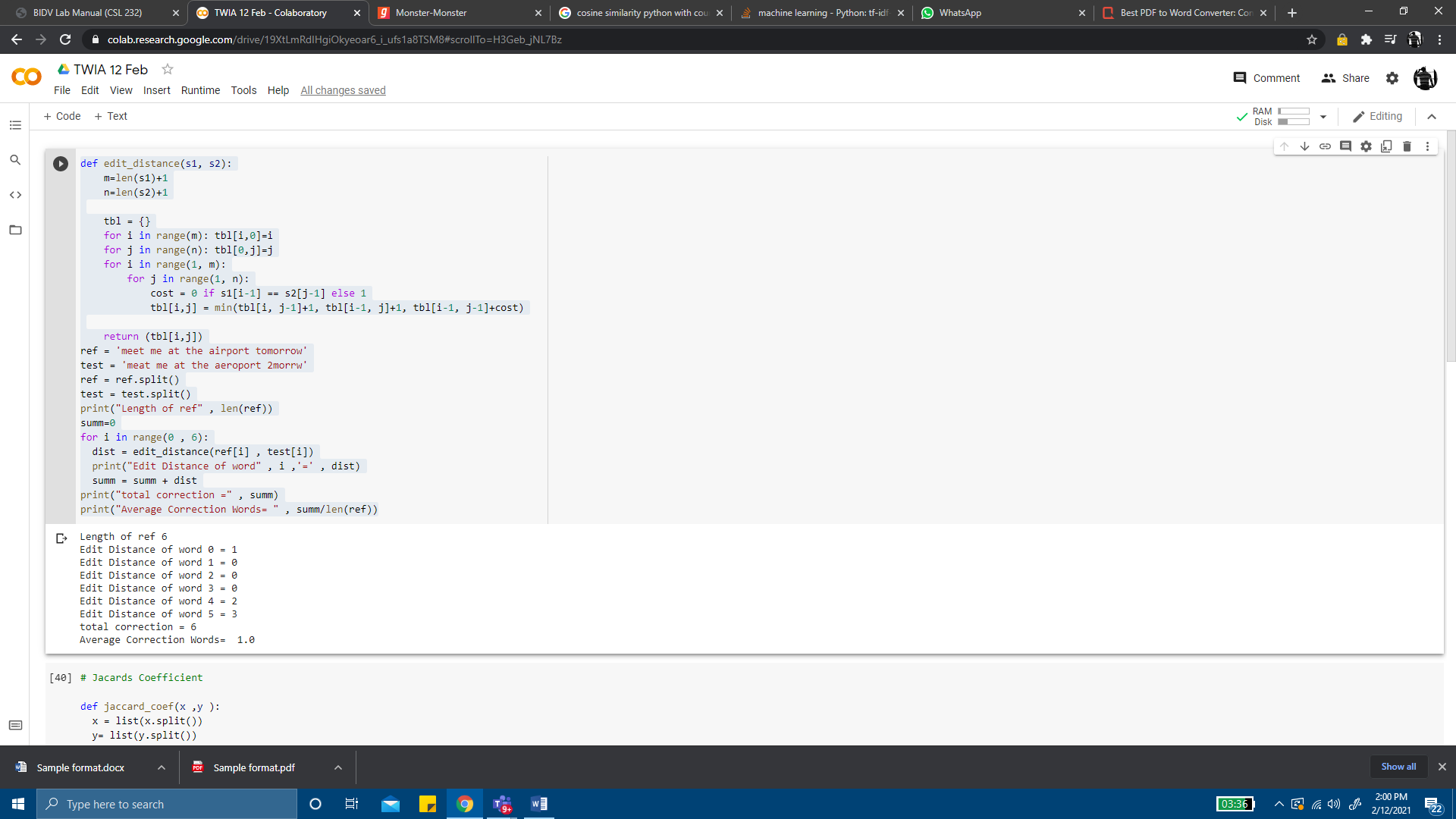
  print("Edit Distance of word" , i ,'=' , dist)

  summ = summ + dist

print("total correction =" , summ)

print("Average Correction Words= " , summ/len(ref))

**OUTPUT:**



**CODE(Cosine Similarity using both TF-IDF and count vectorizer)**

# Cosine Similarity

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from nltk.corpus import stopwords

import numpy as np

import numpy.linalg as LA

import nltk

nltk.download('stopwords')

train\_set = ["The sky is blue.", "The sun is bright."]  # Documents

test\_set = ["The sun in the sky is bright."]  # Query

stopWords = stopwords.words('english')

vectorizer = CountVectorizer(stop\_words = stopWords)

#print vectorizer

transformer = TfidfTransformer()

#print transformer

trainVectorizerArray = vectorizer.fit\_transform(train\_set).toarray()

testVectorizerArray = vectorizer.transform(test\_set).toarray()

print ('Fit Vectorizer to train set', trainVectorizerArray)

print ('Transform Vectorizer to test set', testVectorizerArray)

transformer.fit(trainVectorizerArray)

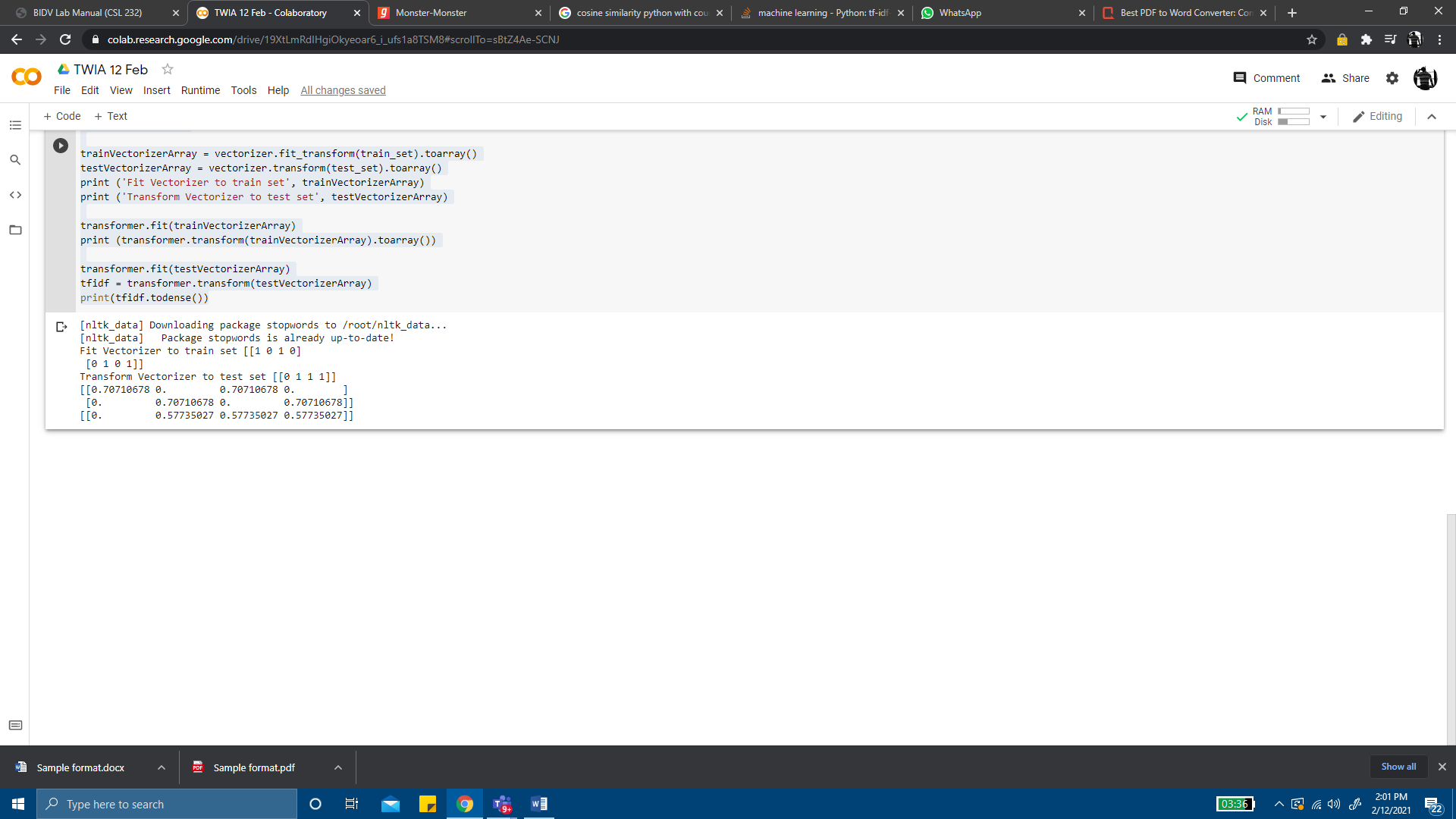
print (transformer.transform(trainVectorizerArray).toarray())

transformer.fit(testVectorizerArray)

tfidf = transformer.transform(testVectorizerArray)

print(tfidf.todense())

**OUTPUT:**





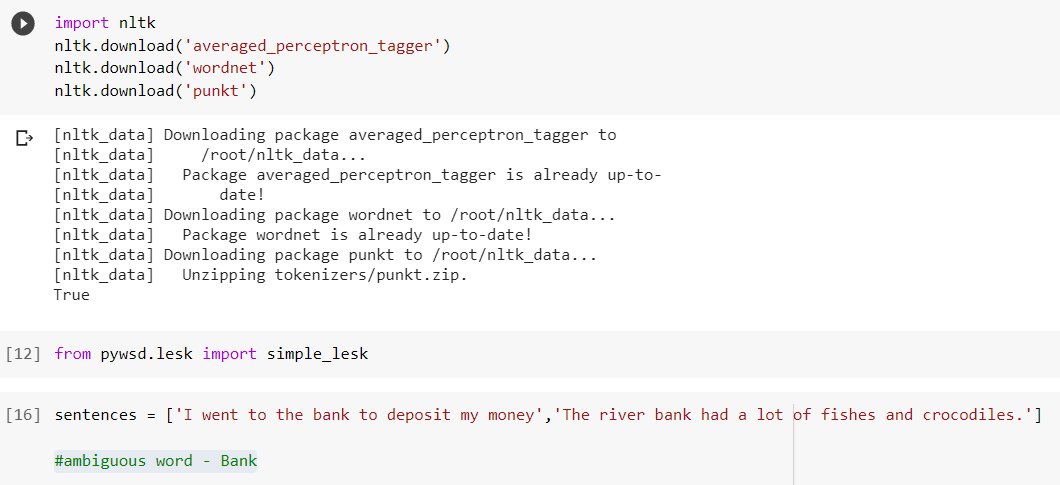
# EXPERIMENT NO. 5

**TWIA Lab Manual (CSL 554)**

**2020-21**

|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 6 March 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective:** Implementation of the Lesk algorithm for Word Sense Disambiguation. |
| **Outcome:** Students will be able to demonstrate how to Lesk algorithm works.  **Problem Statement:** Implement Lesk algorithm for Word Sense Disambiguation.  **CODE AND OUTPUT:** |



**LESK WORKS CORRECTLY**

}13] # Context 1 - Financial institution print (”Context-1: ”, sentences[0])

answer = simple lesk(sentences[0],'bank') print (”Sense:", answer)

print (”Definition ”, answer.definition())

# Correct Output - Financial Institution printed # No disambiguity

Context-1: I went to the bank to deposit my money Sense: Synset('depository\_financial\_institution.n.01')

Definition : a financial institution that accepts deposits and channels the money into lending activities

# Context 2 - River Bank

print (”Context-2: ", sentences[1]) answer = simple\_lesk(sentences[1],'bank') print (”Sense:", answer)

print (”Definition ”, answer.definition())

# Correct Output - River Bank or sloping land printed # No disambiguity

Context-2: The river bank had a lot of fishes and crocodiles.

Sense: Syn set( ' bank. n. 01' )

Definition ’ sloping land (especially the slope beside a body of water)

**LESK WORKS INCORRECTLY**

new\_sentences =[ ' the *workers* at the plant were overworked ' , ' The plant was no longer bearing flowers ' , ' the workers at the industrial plant were overworked ' ]

#ambiguous word °Plant

 *#* Context 1 - Industrial plant

print (“Context-1:“, new sentences[0]) answer = simple\_lesk(new sentences[0]), print (“Sense:“, answer)

print(“Definition : “, answer definition())

# Incorrect output - Industria1 plant not printed # Disambiguity occurred

Context -1: The no Akers at the plant were ove ruorked Sense : Synset ( ' plant. v.06 ' )

Definat ion put firmly in the cdna

[19] 4 Context 2 — TnBe/SeBdf1ng/Sapf1ng

print ("Context-2:’, new sentences[1j) answer = simple\_lesk(new sentences[1j,'plant’) print ("Sense:’, answer)

print ("Definition : ”, answer.definition())

# Correct output - Plant bearing flower sense printed

4 No disambiguity

Context-3: The plant was no longer bearing flowers Sense: Synset('plant.v.01’)

Definition : put or set (seeds, seedlings, or plants) into the ground

 # Context 3 - Industrial plant (Added the word industrial before plant in # Context 1)

print ("Context-3:’, new sentences[2j) answer = simple lesk(new sentences[2j,'plant’) print ("Sense:’, answer)

print ("Definition : ”, answer.definition())

# Correct output - Industrial plant printed

# (Disambiguity resolved by adding the word ’industrial’ in the sentence.)

 Context-3: The workers at the industrial plant were overworked Sense: Synset('plant.n.01’)

Definition : buildings for carrying on industrial labor



**TWIA Lab Manual (CSL 554)**

**2020-21**

**EXPERIMENT NO. 6**

|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 10 April 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective: Implement LDA with BoW and TF-IDF features and compare the results** |
| **Outcome:** Students will be able to demonstrate LDA with BoW and TF-IDF features and compare the results |
| **Problem Statement:** Select the documents and implement above similarities methods.  **Code:**  **import** **pandas** **as** **pd**  data = pd.read\_csv('abcnews-date-text.csv', error\_bad\_lines=**False**);  data\_text = data[['headline\_text']]  data\_text['index'] = data\_text.index  documents = data\_text  **import** **gensim**  **from** **gensim.utils** **import** simple\_preprocess  **from** **gensim.parsing.preprocessing** **import** STOPWORDS  **from** **nltk.stem** **import** WordNetLemmatizer  **from** **nltk.stem.porter** **import** \*  **import** **numpy** **as** **np**  np.random.seed(2018)  **import** **nltk**  *# not stemming as it will not provide valid results*  *#lemmatizing*  *#removing stopwords and words with len<3*  **def** lemmatize\_stemming(text):  **return** WordNetLemmatizer().lemmatize(text, pos='v')  **def** preprocess(text):  result = []  **for** token **in** gensim.utils.simple\_preprocess(text):  **if** token **not** **in** gensim.parsing.preprocessing.STOPWORDS **and** len(token) > 3:  result.append(lemmatize\_stemming(token))  **return** result  processed\_docs = documents['headline\_text'].map(preprocess)  processed\_docs[:10]  dictionary = gensim.corpora.Dictionary(processed\_docs)  dictionary.filter\_extremes(no\_below=15, no\_above=0.5, keep\_n=100000)  bow\_corpus = [dictionary.doc2bow(doc) **for** doc **in** processed\_docs]  **from** **gensim** **import** corpora, models  tfidf = models.TfidfModel(bow\_corpus)  corpus\_tfidf = tfidf[bow\_corpus]  **from** **pprint** **import** pprint  **for** doc **in** corpus\_tfidf:  pprint(doc)  **break**  lda\_model = gensim.models.LdaMulticore(bow\_corpus, num\_topics=10, id2word=dictionary, passes=2, workers=2)  **for** idx, topic **in** lda\_model.print\_topics(-1):  print('Topic: **{}** **\n**Words: **{}**'.format(idx, topic))  lda\_model\_tfidf = gensim.models.LdaMulticore(corpus\_tfidf, num\_topics=10, id2word=dictionary, passes=2, workers=4)  **for** idx, topic **in** lda\_model\_tfidf.print\_topics(-1):  print('Topic: **{}** Word: **{}**'.format(idx, topic))  **OUTPUT:**    **Testing both the models**  *# Bag Of Words*  *# Compute Perplexity*  **from** **gensim.models.coherencemodel** **import** CoherenceModel  print('**\n**Perplexity: ', lda\_model.log\_perplexity(bow\_corpus)) *# a measure of how good the model is. lower the better.*  *# Compute Coherence Score*  coherence\_model\_lda = CoherenceModel(model=lda\_model, texts=processed\_docs, dictionary=dictionary, coherence='c\_v')  coherence\_lda = coherence\_model\_lda.get\_coherence()  print('**\n**Coherence Score: ', coherence\_lda)    *# TFIDF*  *# Compute Perplexity*  **from** **gensim.models.coherencemodel** **import** CoherenceModel  print('**\n**Perplexity: ', lda\_model\_tfidf.log\_perplexity(bow\_corpus)) *# a measure of how good the model is. lower the better.*  *# Compute Coherence Score*  coherence\_model\_lda = CoherenceModel(model=lda\_model\_tfidf, texts=processed\_docs, dictionary=dictionary, coherence='c\_v')  coherence\_lda = coherence\_model\_lda.get\_coherence()  print('**\n**Coherence Score: ', coherence\_lda)  *#since tdidf has more coherence score therefore it is more effective than bow* |
|  |



**TWIA Lab Manual (CSL 554)**

**2020-21**

**EXPERIMENT NO. 7**

|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 20 April 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective: Implementation of KNN, Naive Bayes and Multinominal Naive Bayes.** |
| **Outcome:** Students will be able to demonstrate KNN, Naive Bayes and Multinominal Naive Bayes. |
| **Problem Statement:** Select the documents and implement KNN, Naive Bayes and Multinominal Naive Bayes.  **Code:**  *# We defined the categories which we want to classify*  categories = ['rec.motorcycles', 'sci.electronics',  'comp.graphics', 'sci.med']  *# sklearn provides us with subset data for training and testing*  train\_data = fetch\_20newsgroups(subset='train',  categories=categories, shuffle=**True**, random\_state=42)  print(train\_data.target\_names)  print("**\n**".join(train\_data.data[0].split("**\n**")[:3]))  print(train\_data.target\_names[train\_data.target[0]])  *# Let's look at categories of our first ten training data*  **for** t **in** train\_data.target[:10]:  print(train\_data.target\_names[t])  knn = KNeighborsClassifier(n\_neighbors=7)  *# training our classifier ; train\_data.target will be having numbers assigned for each category in train data*  clf = knn.fit(X\_train\_tfidf, train\_data.target)  *# Input Data to predict their classes of the given categories*  docs\_new = ['I have a Harley Davidson and Yamaha.', 'I have a GTX 1050 GPU']  *# building up feature vector of our input*  X\_new\_counts = count\_vect.transform(docs\_new)  *# We call transform instead of fit\_transform because it's already been fit*  X\_new\_tfidf = tfidf\_transformer.transform(X\_new\_counts)  predicted = clf.predict(X\_new\_tfidf)  **for** doc, category **in** zip(docs\_new, predicted):  print('**%r** => **%s**' % (doc, train\_data.target\_names[category]))  **from** **sklearn.model\_selection** **import** train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_train\_tfidf, train\_data.target, test\_size=0.30, random\_state=42)  print('Training Data Shape:', X\_train.shape)  print('Testing Data Shape: ', X\_test.shape)  Naive Bayes  **from** **sklearn.naive\_bayes** **import** GaussianNB  lr\_model = GaussianNB()  lr\_model.fit(y1, y\_train)  predictions = lr\_model.predict(y2)  **from** **sklearn** **import** metrics  print(metrics.confusion\_matrix(y\_test,predictions))  print(metrics.classification\_report(y\_test,predictions))  print(metrics.accuracy\_score(y\_test,predictions))    Multinomial naive bayes |
|  |
|  |



**TWIA Lab Manual (CSL 554)**

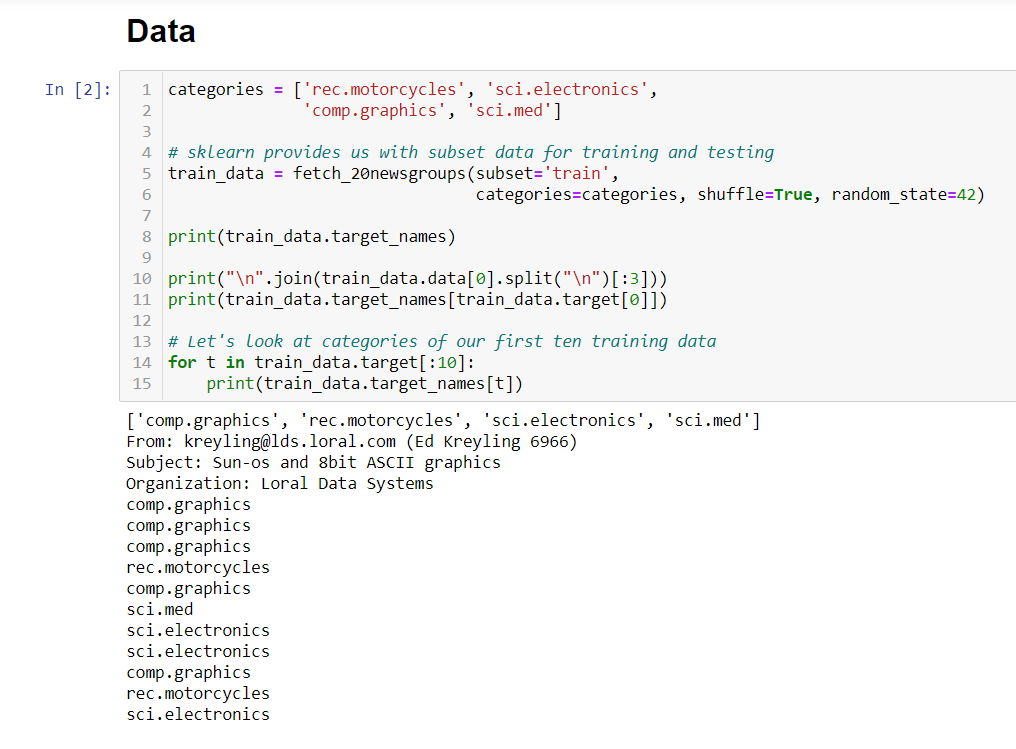
**2020-21**

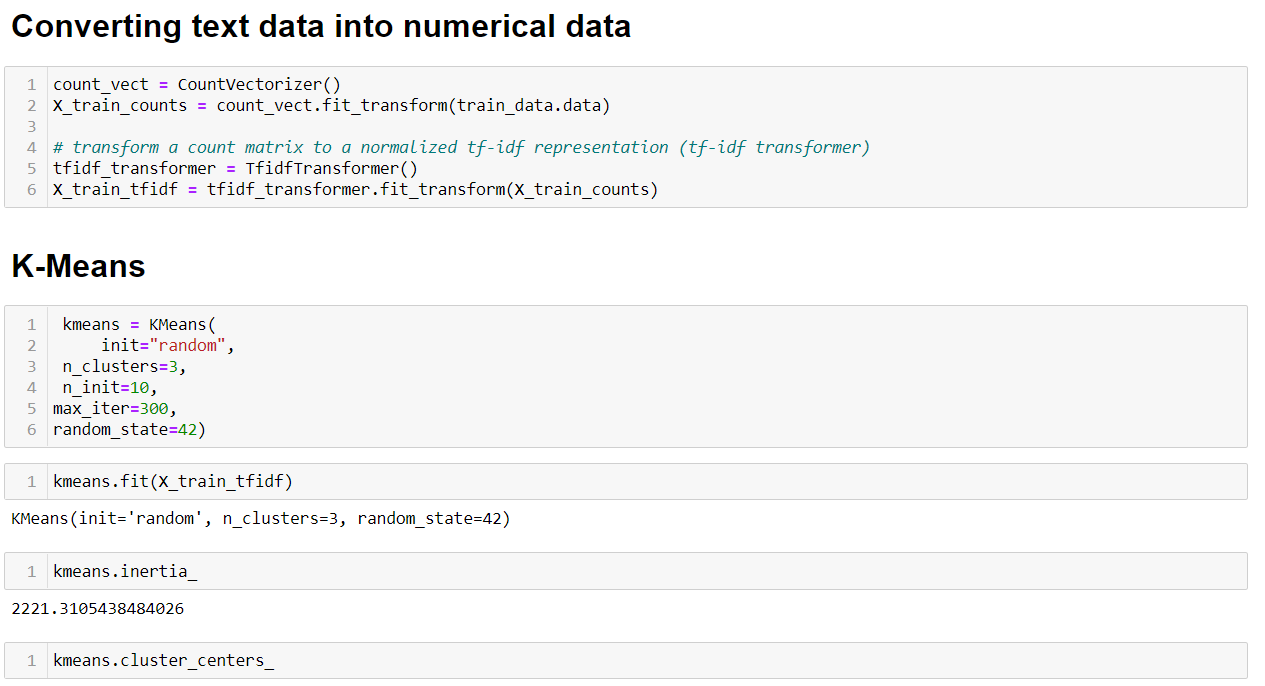
**EXPERIMENT NO. 8**

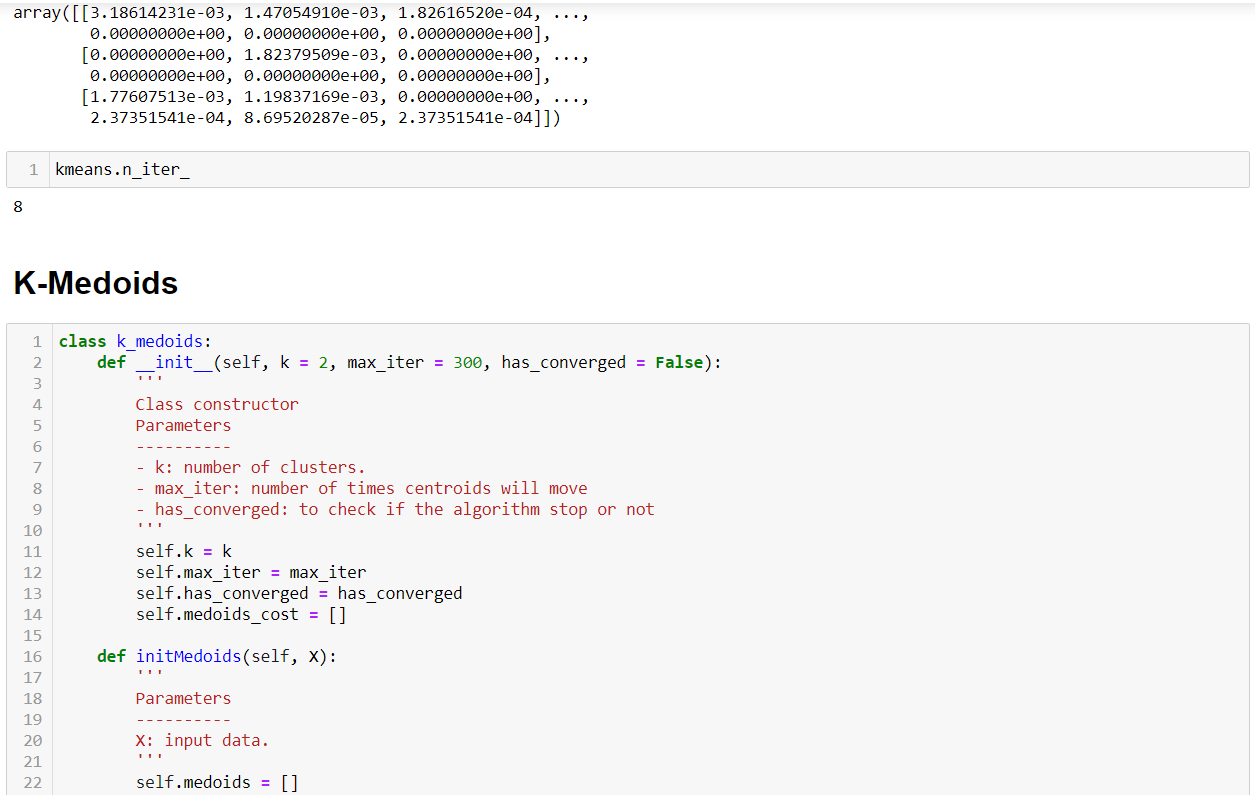
|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 20 April 2021** |
| **Faculty Signature:** |
| **Grade:** |

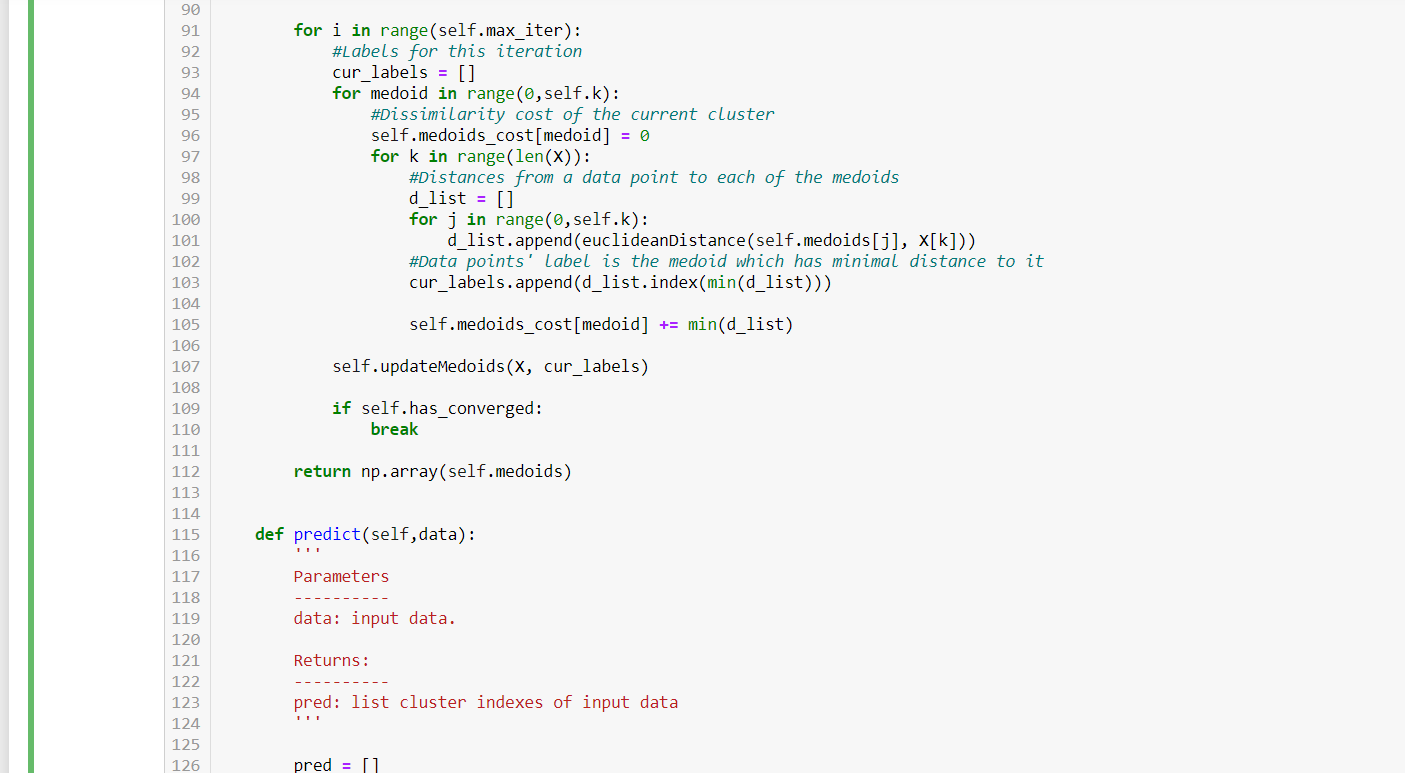
|  |
| --- |
| **Objective: Implementation of k means, k medoids and hierarchical clustering algorithms on Text data.** |

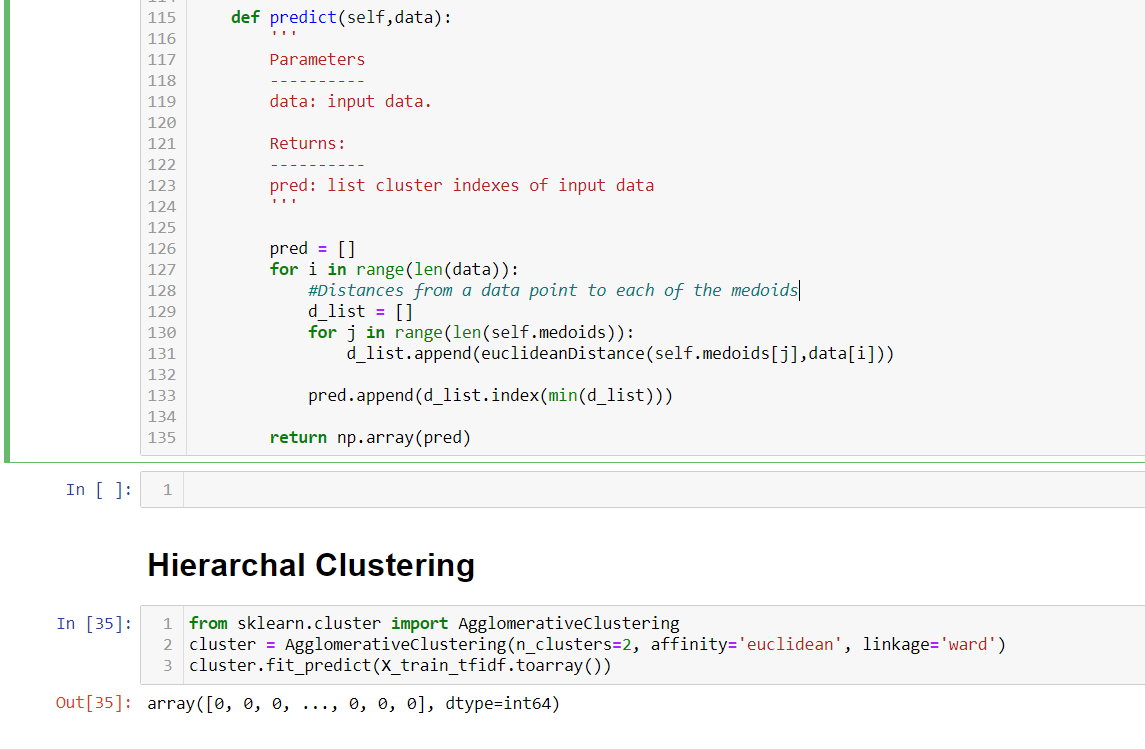
**Output:**













**TWIA Lab Manual (CSL 554)**

**2020-21**

**EXPERIMENT NO. 9**

|  |
| --- |
| **Student Name and Roll Number: RAHUL DHAWAN (17CSU146)** |
| **Semester /Section: 8-C** |
| **Date: 20 April 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective: In this case study, you will investigate homophily of several characteristics of individuals connected in social networks in rural India.** |

**DATA CAMP CASE STUDY 6**

Q1. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will calculate the chance homophily for an arbitrary characteristic. Homophily is the proportion of edges in the network whose constituent nodes share that characteristic. How much homophily do we expect by chance? If characteristics are distributed completely randomly, the probability that two nodes x and y share characteristic a is the probability both nodes have characteristic a, which is the frequency of a squared. The total probability that nodes x and y share their characteristic is therefore the sum of the frequency of each characteristic in the network. For example, in the dictionary favorite\_colors provided, the frequency of red and blue is 1/3 and 2/3 respectively, so the chance homophily is (1/3)^2+(2/3)^2 = 5/9.

**100 XP**

* Create a function that takes a dictionary chars with personal IDs as keys and characteristics as values, and returns a dictionary with characteristics as keys, and the frequency of their occurrence as values.
* Create a function chance\_homophily(chars) that takes a dictionary chars defined as above and computes the chance homophily for that characteristic.
* A sample of three peoples' favorite colors is given in favorite\_colors. Use your function to compute the chance homophily in this group, and store as color\_homophily.
* Print color\_homophily.

CODE:

from collections import Counter

def chance\_homophily(chars):

    """

    Computes the chance homophily of a characteristic,

    specified as a dictionary, chars.

    """

    #enter your code here

    chars\_counts\_dict = Counter(chars.values())

    chars\_counts = np.array(list(chars\_counts\_dict.values()))

    chars\_props  = chars\_counts / sum(chars\_counts)

    return sum(chars\_props\*\*2)

favorite\_colors = {

    "ankit":  "red",

    "xiaoyu": "blue",

    "mary":   "blue"

}

color\_homophily = chance\_homophily(favorite\_colors)

print(color\_homophily)

OUTPUT:

Graphical user interface, text

Description automatically generated

Q2. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In the remaining exercises, we will calculate and compare the actual homophily in these village to chance. In this exercise, we subset the data into individual villages and store them.

**Instructions**

**100 XP**

* individual\_characteristics.dta contains several characteristics for each individual in the dataset such as age, religion, and caste. Use the pandas library to read in and store these characteristics as a dataframe called df.
* Store separate datasets for individuals belonging to Villages 1 and 2 as df1 and df2, respectively.
  + Note that some attributes may be missing for some individuals. In this case study, we will ignore rows of data where some column information is missing.
* Use the head method to display the first few entries of df1.

CODE :

import pandas as pd

df  = pd.read\_stata(data\_filepath + "individual\_characteristics.dta")

df1 = df[df["village"]==1]# Enter code here!

df2 = df[df["village"]==2]# Enter code here!

# Enter code here!

df1.head()

OUTPUT:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Q3. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we define a few dictionaries that enable us to look up the sex, caste, and religion of members of each village by personal ID. For Villages 1 and 2, their personal IDs are stored as pid.

**Instructions**

**100 XP**

* Define dictionaries with personal IDs as keys and a given covariate for that individual as values. Complete this for the sex, caste, and religion covariates, for Villages 1 and 2.
* For Village 1, store these dictionaries into variables named sex1, caste1, and religion1.
* For Village 2, store these dictionaries into variables named sex2, caste2, and religion2.

CODE:

sex1      = df1.set\_index("pid")["resp\_gend"].to\_dict()

caste1    = df1.set\_index("pid")["caste"].to\_dict()

religion1 = df1.set\_index("pid")["religion"].to\_dict()

sex2      = df2.set\_index("pid")["resp\_gend"].to\_dict()

caste2    = df2.set\_index("pid")["caste"].to\_dict()

religion2 = df2.set\_index("pid")["religion"].to\_dict()

Q4.Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will print the chance homophily of several characteristics of Villages 1 and 2. The function chance\_homophily is still defined from Exercise 1.

**Instructions**

**100 XP**

* sex1, caste1, religion1, sex2, caste2, and religion2 are already defined from previous exercises. Use chance\_homophily to compute the chance homophily for sex, caste, and religion In Villages 1 and 2. Is the chance homophily for any attribute very high for either village?

CODE :

print("Village 1 chance of same sex:", chance\_homophily(sex1))

# Enter your code here.

print("Village 1 chance of same caste:", chance\_homophily(caste1))

print("Village 1 chance of same religion:", chance\_homophily(religion1))

print("Village 2 chance of same sex:", chance\_homophily(sex2))

print("Village 2 chance of same caste:", chance\_homophily(caste2))

print("Village 2 chance of same religion:", chance\_homophily(religion2))

OUTPUT:

Graphical user interface, text, chat or text message

Description automatically generated

Q5. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will create a function that computes the observed homophily given a village and characteristic.

**Instructions**

**100 XP**

* Complete the function homophily(), which takes a network G, a dictionary of characteristics chars, and node IDs IDs. For each node pair, determine whether a tie exists between them, as well as whether they share a characteristic. The total count of these is num\_same\_ties and num\_ties respectively, and their ratio is the homophily of chars in G. Complete the function by choosing where to increment num\_same\_ties and num\_ties.

CODE:

def homophily(G, chars, IDs):

    num\_same\_ties, num\_ties = 0, 0

    for n1 in G.nodes():

        for n2 in G.nodes():

            if n1 > n2:   # do not double-count edges!

                if IDs[n1] in chars and IDs[n2] in chars:

                    if G.has\_edge(n1, n2):

                        num\_ties += 1# Should `num\_ties` be incremented?  What about `num\_same\_ties`?

                        if chars[IDs[n1]] == chars[IDs[n2]]:

                            num\_same\_ties += 1# Should `num\_ties` be incremented?  What about `num\_same\_ties`?

    return (num\_same\_ties / num\_ties)

Q6. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will obtain the personal IDs for Villages 1 and 2. These will be used in the next exercise to calculate homophily for these villages.

**Instructions**

**100 XP**

* In this dataset, each individual has a personal ID, or PID, stored in key\_vilno\_1.csv and key\_vilno\_2.csv for villages 1 and 2, respectively. data\_filepath contains the base URL to the datasets used in this exercise. Use pd.read\_csv to read in and store key\_vilno\_1.csv and key\_vilno\_2.csv as pid1 and pid2 respectively. The csv files have no headers, so make sure to include the parameter header = None.

CODE:

pid1 = pd.read\_csv(data\_filepath + "key\_vilno\_1.csv", dtype=int, header = None)

pid2 = pd.read\_csv(data\_filepath + "key\_vilno\_2.csv", dtype=int, header = None)

Q7. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will compute the homophily of several network characteristics for Villages 1 and 2, and compare this to chance homophily. The networks for these villages have been stored as networkx graph objects G1 and G2. homophily() and chance\_homophily() are pre-loaded from previous exercises.

CODE :

print("Village 1 observed proportion of same sex:", homophily(G1, sex1, pid1))

# Enter your code here!

print("Village 1 observed proportion of same caste:", homophily(G1, caste1, pid1))

print("Village 1 observed proportion of same religion:", homophily(G1, religion1, pid1))

print("Village 2 observed proportion of same sex:", homophily(G2, sex2, pid2))

print("Village 2 observed proportion of same caste:", homophily(G2, caste2, pid2))

print("Village 2 observed proportion of same religion:", homophily(G2, religion2, pid2))

print("Village 1 chance of same sex:", chance\_homophily(sex1))

print("Village 1 chance of same caste:", chance\_homophily(caste1))

print("Village 1 chance of same religion:", chance\_homophily(religion1))

print("Village 2 chance of same sex:", chance\_homophily(sex2))

print("Village 2 chance of same caste:", chance\_homophily(caste2))

print("Village 2 chance of same religion:", chance\_homophily(religion2))

OUTPUT:

Graphical user interface, text, chat or text message

Description automatically generated