



# **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**

## Index

	Experiment	Date	Faculty Sign.
1.	To Find the most frequently occurring 5 words from a piece of text.	12-02	
2.	Print all the Arabic Stopwords. Omit a given list of stop words from the total stopwords list of English language.	5-02	
3.	Print the total number of male and female names in the names corpus. Then, Print the first 15 male and female names. Print the definition and examples of any one English language word using WordNet corpus. From the names corpus, combine all the labelled male and female names and print any 20.	24-02	
4.	To Find similarity of two pieces of text. Compute the similarity of two pieces of text using Levenshtein Edit Distance, Jaccard similarity and Cosine Similarity using both TF-IDF and count vectorizer	10-03	
5.	Implementation of the Lesk algorithm for Word Sense Disambiguation	17-3	
6.	Implement LDA with BoW Implement with TF-IDF Compare the accuracy and make the analysis	24-3	
7.	To implement of KNN, Naive Bayes and Multinomial Naive Bayes.	14-4	
8.	To implement of k means, k medoids and hierarchical clustering algorithms on Text data	21-4	
9.	To implement homophily for social media analysis.	28-4	

## EXPERIMENT NO. 1

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

**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 python 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:

```
print(fd.most_common(5))
```

```
[(',', 2), ('hello', 1), ('mr.', 1), ('smith', 1), ('today', 1)]
```

---

## EXPERIMENT NO. 2

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

### Objective: To

- Print all the Arabic Stopwords.
- 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)
```

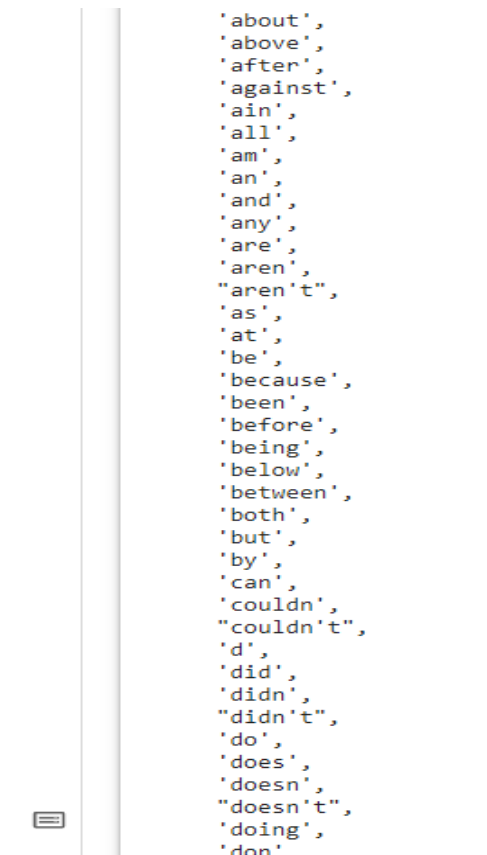
### Output

أكثر, إلا, التي, الذي, الذين, اللاتي, اللتان, اللتيا, اللتين, اللذان, اللذين, اللواتي, أم, أما, إما, أن, إن, إنا, أنا, أنت, أنتم, أنتما, أنتن, إنما, إنه, أني, أي, أه, ب, أي, أيا, إي, أين, أينما, إيه, يخ, يس, بعد, بعض, بك, بكم, بكما, بك, بنا, به, بها, بهم, بهما, بهن, بي, بين, بيد, تلك, تلكم, تلكما, ته, تي, تين, تينك, ذ, حيتما, حين, خلا, دون, ذا, ذات, ذاك, ذان, ذاك, ذلك, ذلكم, ذلكما, ذلكن, ذه, ذو, نك, ريت, سوف, سوى, شتان, عدا, عسى, عل, على, عليك, عليه, عما, عن, عند, غير, فيما, فيه, فيها, قد, كأن, كأنما, كأني, كذا, كذلك, كل, كلا, كلاهما, كلنا, كلما, كيت, كيف, كيفما, لا, لاسيما, لذي, لست, لستم, لستما, لستن, لسن, لسننا, لعل, لك, لا, لم, لما, لن, لنا, له, لها, لهم, لهما, لهن, لو, لولا, لوما, لي, لئن, ليت, ليس, ماذا, متى, مذ, مع, مما, ممن, من, منه, منها, منذ, مه, مهما, نحن, نحو, نعم, ها, هاهنا, هذا, هذان, هذه, هذي, هذين, هكذا, هل, هلا, هم, هما, هن, هنا, هناك, هنالك, هيا, هيت, هيهات, والذي, والذين, وإذ, وإذا, وإن, ولا, ولكن, ولو, وما, ومن, وهو, يا

## 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



```
'about',
'above',
'after',
'against',
'ain',
'all',
'am',
'an',
'and',
'any',
'are',
'aren',
"aren't",
'as',
'at',
'be',
'because',
'been',
'before',
'being',
'below',
'between',
'both',
'but',
'by',
'can',
'couldn't',
"couldn't",
'd',
'did',
'didn',
"didn't",
'do',
'does',
'doesn',
"doesn't",
'doing',
'don'
```

### EXPERIMENT NO. 3

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

**Objective:** To

- Print the total number of male and female names in the names corpus. Then, Print the first 15 male and female names.
- 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**

```
Number of male names:
2943

Number of female names:
5001

First 10 male names:
['Aamin', 'Aaron', 'Abbey', 'Abbie', 'Abbot', 'Abbott', 'Abby', 'Abdel', 'Abdul', 'Abdulkarim', 'Abdullah', 'Abe', 'Abel', 'Abelard', 'Abner']

First 10 female names:
['Abagael', 'Abigail', 'Abbe', 'Abbey', 'Abbi', 'Abbie', 'Abby', 'Abigael', 'Abigail', 'Abigale', 'Abra', 'Acacia', 'Ada', 'Adah', 'Adaline']
```

---

## 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

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
Defination of the said word:
the activities of educating or instructing; activities that impart knowledge or skill

Examples of the word in use::
['he received no formal education', 'our instruction was carefully programmed', 'good classroom teaching is seldom rewarded']
```



## EXPERIMENT NO. 4

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

**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))
```

## OUTPUT

```
# Jaccards Coefficient

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))
```

```
Jaccard Coefficient = 0.3333333333333333
```

## 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:

```
↳ Length of ref 6
   Edit Distance of word 0 = 1
   Edit Distance of word 1 = 0
   Edit Distance of word 2 = 0
   Edit Distance of word 3 = 0
   Edit Distance of word 4 = 2
   Edit Distance of word 5 = 3
   total correction = 6
   Average Correction Words= 1.0
```

## 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:

```
[nltk_data] Downloading package stopwords to /root/nltk_data...  
[nltk_data]   Package stopwords is already up-to-date!  
Fit Vectorizer to train set [[1 0 1 0]  
[0 1 0 1]]  
Transform Vectorizer to test set [[0 1 1 1]]  
[[0.70710678 0.          0.70710678 0.          ]  
[0.          0.70710678 0.          0.70710678]]  
[[0.          0.57735027 0.57735027 0.57735027]]
```

## EXPERIMENT NO. 5

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

**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:

```
import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')

[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
True
```

```
[12] from pywsd.lesk import simple_lesk
```

```
[16] sentences = ['I went to the bank to deposit my money','The river bank had a lot of fishes and crocodiles.']

#ambiguous word - Bank
```

## 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
```

```
[18] # Context 1 - Industrial plant
```

```
print ("Context-1:", new_sentences[0])
answer=simple_lesk(new_sentences[0]),
print ("Sense:",answer)
print("Definition : ", answerdefinition())
```

```
# Incorrect output - Industrial plant not
printed # Disambiguity occurred
```

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



## EXPERIMENT NO. 6

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

**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: {} \nWords: {}'.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:

```

Topic: 0
Words: 0.057*"australia" + 0.041*"trump" + 0.024*"australian" + 0.022*"china" + 0.019*"world" + 0.019*"sydney" + 0.017*"open" +
0.017*"coronavirus" + 0.015*"border" + 0.012*"win"
Topic: 1
Words: 0.023*"market" + 0.019*"year" + 0.016*"record" + 0.012*"care" + 0.012*"price" + 0.012*"years" + 0.012*"australian" + 0.011*"business" + 0.011*"country" + 0.010*"age"
Topic: 2
Words: 0.065*"coronavirus" + 0.032*"covid" + 0.029*"government" + 0.015*"rise" + 0.015*"restrictions" + 0.014*"water" + 0.012*"royal" + 0.012*"scott" + 0.011*"tasmanian" + 0.010*"commission"
Topic: 3
Words: 0.027*"kill" + 0.022*"die" + 0.019*"coast" + 0.018*"shoot" + 0.017*"miss" + 0.016*"crash" + 0.015*"attack" + 0.015*"golfd" + 0.015*"dead" + 0.014*"island"
Topic: 4
Words: 0.027*"kill" + 0.022*"die" + 0.019*"coast" + 0.018*"shoot" + 0.017*"miss" + 0.016*"crash" + 0.015*"attack" + 0.015*"golfd" + 0.015*"dead" + 0.014*"island"

```

## Testing both the models

```

# Bag Of Words
# Compute Perplexity
from gensim.models.coherencemodel import CoherenceModel
print('\nPerplexity: ', 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('\nCoherence Score: ', coherence_lda)

```

**Perplexity: -9.118709657723862**

**Coherence Score: 0.24535051024041796**

```
# TFIDF
# Compute Perplexity
from gensim.models.coherencemodel import CoherenceModel
print('\nPerplexity: ', 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('\nCoherence Score: ', coherence_lda)

#since tfidf has more coherence score therefore it is more effective than bow
```

Perplexity: -8.997436855159867

Coherence Score: 0.30983791499722557

## EXPERIMENT NO. 7

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

**Objective:** Implementation of KNN, Naive Bayes and Multinomial Naive Bayes.

**Outcome:** Students will be able to demonstrate KNN, Naive Bayes and Multinomial Naive Bayes.

**Problem Statement:** Select the documents and implement KNN, Naive Bayes and Multinomial 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_names, 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))

```

		precision	recall	f1-score	support
	0	0.87	0.90	0.88	167
	1	0.96	0.94	0.95	189
	2	0.88	0.84	0.86	172
	3	0.91	0.94	0.92	183
[[150	1	9	7]		
[ 4	177	3	5]		
[ 18	4	144	6]		
[ 1	2	8	172]]		
	accuracy			0.90	711
	macro avg	0.90	0.90	0.90	711
	weighted avg	0.90	0.90	0.90	711

## Multinomial naive bayes

```
from sklearn.naive_bayes import MultinomialNB
lr_model = MultinomialNB()
lr_model.fit(X_train, y_train)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
predictions = lr_model.predict(X_test)
```

```
from sklearn import metrics
print(metrics.confusion_matrix(y_test,predictions))
```

```
[[161  0  4  2]
 [ 0 187  1  1]
 [ 5  3 164  0]
 [ 3  2  3 175]]
```

```
print(metrics.classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	0.95	0.96	0.96	167
1	0.97	0.99	0.98	189
2	0.95	0.95	0.95	172
3	0.98	0.96	0.97	183
accuracy			0.97	711
macro avg	0.97	0.97	0.97	711
weighted avg	0.97	0.97	0.97	711

## EXPERIMENT NO. 8

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

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

**Output:**

### Data

```
In [2]: 1 categories = ['rec.motorcycles', 'sci.electronics',  
2             'comp.graphics', 'sci.med']  
3  
4 # sklearn provides us with subset data for training and testing  
5 train_data = fetch_20newsgroups(subset='train',  
6                               categories=categories, shuffle=True, random_state=42)  
7  
8 print(train_data.target_names)  
9  
10 print("\n".join(train_data.data[0].split("\n")[:3]))  
11 print(train_data.target_names[train_data.target[0]])  
12  
13 # Let's look at categories of our first ten training data  
14 for t in train_data.target[:10]:  
15     print(train_data.target_names[t])
```

['comp.graphics', 'rec.motorcycles', 'sci.electronics', 'sci.med']  
From: kreyling@lds.loral.com (Ed Kreyling 6966)  
Subject: Sun-os and 8bit ASCII graphics  
Organization: Loral Data Systems  
comp.graphics  
comp.graphics  
comp.graphics  
rec.motorcycles  
comp.graphics  
sci.med  
sci.electronics  
sci.electronics  
comp.graphics  
rec.motorcycles  
sci.electronics

## Converting text data into numerical data

```
1 count_vect = CountVectorizer()
2 X_train_counts = count_vect.fit_transform(train_data.data)
3
4 # transform a count matrix to a normalized tf-idf representation (tf-idf transformer)
5 tfidf_transformer = TfidfTransformer()
6 X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

## K-Means

```
1 kmeans = KMeans(
2     init="random",
3     n_clusters=3,
4     n_init=10,
5     max_iter=300,
6     random_state=42)
```

```
1 kmeans.fit(X_train_tfidf)
```

KMeans(init='random', n\_clusters=3, random\_state=42)

```
1 kmeans.inertia_
```

2221.3105438484026

```
1 kmeans.cluster_centers_
```

```
array([[3.18614231e-03, 1.47054910e-03, 1.82616520e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 1.82379509e-03, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.77607513e-03, 1.19837169e-03, 0.00000000e+00, ...,
        2.37351541e-04, 8.69520287e-05, 2.37351541e-04]])
```

```
1 kmeans.n_iter_
```

8

## K-Medoids

```
1 class k_medoids:
2     def __init__(self, k = 2, max_iter = 300, has_converged = False):
3         '''
4         Class constructor
5         Parameters
6         -----
7         - k: number of clusters.
8         - max_iter: number of times centroids will move
9         - has_converged: to check if the algorithm stop or not
10        '''
11        self.k = k
12        self.max_iter = max_iter
13        self.has_converged = has_converged
14        self.medoids_cost = []
15
16        def initMedoids(self, X):
17            '''
18            Parameters
19            -----
20            X: input data.
21            '''
22            self.medoids = []
```



```

90
91     for i in range(self.max_iter):
92         #Labels for this iteration
93         cur_labels = []
94         for medoid in range(0,self.k):
95             #Dissimilarity cost of the current cluster
96             self.medoids_cost[medoid] = 0
97             for k in range(len(X)):
98                 #Distances from a data point to each of the medoids
99                 d_list = []
100                 for j in range(0,self.k):
101                     d_list.append(euclideanDistance(self.medoids[j], X[k]))
102                 #Data points' Label is the medoid which has minimal distance to it
103                 cur_labels.append(d_list.index(min(d_list)))
104
105                 self.medoids_cost[medoid] += min(d_list)
106
107         self.updateMedoids(X, cur_labels)
108
109         if self.has_converged:
110             break
111
112     return np.array(self.medoids)
113
114
115 def predict(self,data):
116     """
117     Parameters
118     -----
119     data: input data.
120
121     Returns:
122     -----
123     pred: list cluster indexes of input data
124     """
125
126     pred = []
127
128     def predict(self,data):
129         """
130         Parameters
131         -----
132         data: input data.
133
134         Returns:
135         -----
136         pred: list cluster indexes of input data
137         """
138
139         pred = []
140         for i in range(len(data)):
141             #Distances from a data point to each of the medoids
142             d_list = []
143             for j in range(len(self.medoids)):
144                 d_list.append(euclideanDistance(self.medoids[j],data[i]))
145
146             pred.append(d_list.index(min(d_list)))
147
148         return np.array(pred)

```

In [ ]: 1

## Hierarchal Clustering

```

In [35]: 1 from sklearn.cluster import AgglomerativeClustering
2 cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
3 cluster.fit_predict(X_train_tfidf.toarray())

```

Out[35]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

## EXPERIMENT NO. 9

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

**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:

The screenshot shows a web browser window with the DataCamp interface. On the left, the 'Exercise 1' instructions are visible. On the right, a code editor shows the same Python code as above. Below the code editor, the output of the code is displayed: 0.555555555556. The interface includes a 'Course Outline' button, a 'Daily XP 700' indicator, and a 'Light Mode' toggle.

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:

The screenshot shows a DataCamp exercise interface for 'Exercise 2'. The left sidebar contains a list of topics: Network, homophily, occurs, when, nodes, that, share an, edge, share a, characteris, more, often, than, nodes. The main area displays a Jupyter Notebook with a script named 'script.py'. The notebook shows the following code and output:

```
df1 = df[df["village"]==1]
df1.head()
```

The output of the code is a DataFrame with the following columns: village, adjmatrix\_key, pid, hhid, resp\_id, resp\_gend, resp\_status, age, religion, caste, and shgparticipate. The first five rows of the output are:

	village	adjmatrix_key	pid	hhid	resp_id	resp_gend	resp_status	age	religion	caste	shgparticipate
0	1	5	100201	1002	1	1	Head of Household	38	HINDUISM	OBC	0
1	1	6	100202	1002	2	2	Spouse of Head of Household	27	HINDUISM	OBC	0
2	1	23	100601	1006	1	1	Head of Household	29	HINDUISM	OBC	0
3	1	24	100602	1006	2	2	Spouse of Head of Household	24	HINDUISM	OBC	0
4	1	27	100701	1007	1	1	Head of Household	58	HINDUISM	OBC	0

The bottom of the interface shows a Windows taskbar with the search bar and various application icons.

**Exercise 2**

Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge.

```

privategovt work_outside work_outside_freq shgparticipate shg_no \
0 PRIVATE BUSINESS Yes 0 No NaN
1 NaN NaN NaN No NaN
2 OTHER LAND No NaN No NaN
3 PRIVATE BUSINESS No NaN Yes 1
4 OTHER LAND No NaN No NaN

savings_no savings_no electioncard rationcard rationcard_colour
0 No NaN Yes Yes GREEN
1 No NaN Yes Yes GREEN
2 No NaN Yes Yes GREEN
3 Yes 1.0 Yes No
4 No NaN Yes Yes GREEN
[5 rows x 48 columns]

```

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:

The screenshot shows a web browser window with the DataCamp interface. On the left, the 'Instructions' panel for 'Exercise 4' is visible, containing the same text as the first block. The main area shows a code editor with the Python code from the 'CODE' block. Below the editor is an 'iPython Shell' displaying the output of the code. The output shows the chance homophily values for each attribute in both villages.

```
Village 1 chance of same caste: 0.674148850979
Village 1 chance of same religion: 0.980489698852
Village 2 chance of same sex: 0.500594530321
Village 2 chance of same caste: 0.425368244801
Village 2 chance of same religion: 1.0
```

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 increment
                            ed? 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:



`homophily()` and `chance_homophily()` are pre-loaded from previous exercises.

100 XP

- Use your `homophily()` function to compute the observed homophily for sex, caste, and religion in Villages 1 and 2. Print all six values.
- Use the `chance_homophily()` to compare these values to chance homophily. Are these values higher or lower than that expected by chance?

💡 Take Hint (-30 XP)

```
1 print("Village 1 observed proportion",
```

Downloaded from <http://ajph.org/> on November 10, 2015

```
Village 1 observed proportion of same sex: 0.5908629441624366
Village 1 observed proportion of same caste: 0.7959390862944162
Village 1 observed proportion of same religion: 0.9908629441624366
Village 2 observed proportion of same sex: 0.5658073270013568
Village 2 observed proportion of same caste: 0.8276797829036635
Village 2 observed proportion of same religion: 1.0
Village 1 chance of same sex: 0.502729986168
Village 1 chance of same caste: 0.674148850979
Village 1 chance of same religion: 0.980489698852
Village 2 chance of same sex: 0.500594530321
Village 2 chance of same caste: 0.425368244801
Village 2 chance of same religion: 1.0
```

In [1]: