TEXT AND WEB ANALYTICS
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

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 Multinominal Naive Bayes.	14-4	
8.	To implement of k means, k medoids and hierarchical clustering algorithms on Text data	21-4	

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:

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

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

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

```
'أكثر', 'ألا', 'إلا', 'التي', 'الذي', 'الذين', 'اللاتي', 'اللائي', 'اللتان', 'اللتيا', 'اللتين', 'اللذان', 'اللذين', 'اللواتي', '[
'أم', 'أما', 'أما', 'إما', 'أن', 'إن', 'إنا', 'أنا', 'أنت', 'أنتم', 'أنتما', 'أنتن', 'إنما', 'إنه', 'أنى', 'أنى', 'آه',
'أي', 'أيها', 'إي', 'أين', 'أين', 'أينما', 'إيه', 'بخ', 'بس', 'بعد', 'بعض', 'بك', 'بكم', 'بكم', 'بكما', 'بك
'بنا', 'به', 'بها', 'بهم', 'بهما', 'بهن', 'بي', 'بين', 'بيد', 'تلك', 'تلكم', 'تلكما', 'ته', 'تي', 'تين', 'تينك',
'حيثما', 'حين', 'خلا', 'دون', 'ذا', 'ذات', 'ذاك', 'ذان', 'ذانك', 'ذلك', 'ذلكم', 'ذلكما', 'ذلكن', 'ذه', 'ذو',
'ريث', 'سوف', 'سوى', 'شتان', 'عدا', 'عسى', 'عل', 'على', 'عليك', 'عليه', 'عما', 'عن', 'عند', 'غير',
'فيما', 'فيه', 'فيها', 'قد', 'كأن', 'كأنما', 'كأي', 'كأين', 'كذا', 'كذلك', 'كل', 'كلا', 'كلاهما', 'كلتا', 'كلما',
'كيت', 'كيف', 'كيفما', 'لا', 'لاسيما', 'لدى', 'لست', 'لستم', 'لستما', 'لستن', 'لسن', 'لسنا', 'لعل', 'لك',
'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',
"couldn't",
'd',
'did',
'didn',
"didn't",
'do',
'does',
'doesn',
"doesn't",
'doing',
'don'
```

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

```
Number of male names:
2943

Number of female names:
5001

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

First 10 female names:
['Abagael', 'Abagail', '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

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))
```

OUTPUT

```
# Jacards 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

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:

```
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(newsentences[0]),
print ("Sense:",answer)
print("Definition : ", answerdefinition())
```

Incorrect output - Industria1 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

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
```

```python
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=dicti
onary, 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, id2wo
rd=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" + 0.0
11*"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*"gol
d" + 0.015*"dead" + 0.014*"island"
Topic: 4
```

Testing both the models

```python
# Bag Of Words
# Compute Perplexity
from gensim.models.coherencemodel import CoherenceModel
print('\nPerplexity: ', lda_model.log_perplexity(bow_corpus))  # a measure of h
ow good the model is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=processed_docs, dic
tionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -9.118709657723862

Coherence Score:  0.24535051024041796
```

```python
# TFIDF
# Compute Perplexity
from gensim.models.coherencemodel import CoherenceModel
print('\nPerplexity: ', lda_model_tfidf.log_perplexity(bow_corpus))  # a measur
e of how good the model is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model_tfidf, texts=processed_doc
s, dictionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)

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

```
Perplexity:  -8.997436855159867

Coherence Score:  0.30983791499722557
```

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_sta
te=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 f
or 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.t
arget, 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))
```

```
[[150   1   9   7]
 [  4 177   3   5]
 [ 18   4 144   6]
 [  1   2   8 172]]
```

```
              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

    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

```python
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)
```

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

```python
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]]
```

```python
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

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:

Data

In [2]:

```
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])
```

```
['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

```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(train_data.data)

# transform a count matrix to a normalized tf-idf representation (tf-idf transformer)
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

K-Means

```
 kmeans = KMeans(
     init="random",
 n_clusters=3,
 n_init=10,
max_iter=300,
random_state=42)
```

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

```
KMeans(init='random', n_clusters=3, random_state=42)
```

```
kmeans.inertia_
```

```
2221.3105438484026
```

```
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]])
```

```
kmeans.n_iter_
```

```
8
```

K-Medoids

```
class k_medoids:
    def __init__(self, k = 2, max_iter = 300, has_converged = False):
        '''
        Class constructor
        Parameters
        ----------
        - k: number of clusters.
        - max_iter: number of times centroids will move
        - has_converged: to check if the algorithm stop or not
        '''
        self.k = k
        self.max_iter = max_iter
        self.has_converged = has_converged
        self.medoids_cost = []

    def initMedoids(self, X):
        '''
        Parameters
        ----------
        X: input data.
        '''
        self.medoids = []
```

```

        for i in range(self.max_iter):
            #Labels for this iteration
            cur_labels = []
            for medoid in range(0,self.k):
                #Dissimilarity cost of the current cluster
                self.medoids_cost[medoid] = 0
                for k in range(len(X)):
                    #Distances from a data point to each of the medoids
                    d_list = []
                    for j in range(0,self.k):
                        d_list.append(euclideanDistance(self.medoids[j], X[k]))
                    #Data points' label is the medoid which has minimal distance to it
                    cur_labels.append(d_list.index(min(d_list)))

                    self.medoids_cost[medoid] += min(d_list)

            self.updateMedoids(X, cur_labels)

            if self.has_converged:
                break

        return np.array(self.medoids)


    def predict(self,data):
        '''
        Parameters
        ----------
        data: input data.

        Returns:
        ----------
        pred: list cluster indexes of input data
        '''

        pred = []
```

```
    def predict(self,data):
        '''
        Parameters
        ----------
        data: input data.

        Returns:
        ----------
        pred: list cluster indexes of input data
        '''

        pred = []
        for i in range(len(data)):
            #Distances from a data point to each of the medoids
            d_list = []
            for j in range(len(self.medoids)):
                d_list.append(euclideanDistance(self.medoids[j],data[i]))

            pred.append(d_list.index(min(d_list)))

        return np.array(pred)
```

In [ ]:

```

```

Hierarchal Clustering

In [35]:

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
cluster.fit_predict(X_train_tfidf.toarray())
```

Out[35]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```