

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Programme** | : | B.Tech | **Semester** | : | WINTER SEM 21-22 |
| **Course** | : | Web Mining Lab | **Code** | : | CSE3024 |
| **Faculty** | : | Dr.Bhuvaneswari A | **Slot** | : | L7+L8 |
| **Time** | : | 8:00 PM to 9:30 PM | **Exam** | : | Challenging Task |
| **Date** | : | 05-04-2022 | **Marks** | : | 30 Marks |

**NAME: Dimish Pamnani**

**Reg No: 19BCE1407**

**Google Colab Link:** [**https://colab.research.google.com/drive/1Xpk27TH4R-xhVVXtWyHYKqw6sQCyD7R-?usp=sharing**](https://colab.research.google.com/drive/1Xpk27TH4R-xhVVXtWyHYKqw6sQCyD7R-?usp=sharing)

**CHALLENGING TASK - I**

**Answer any two questions (2 X 15 = 30 Marks)**

1. Write a Python program to classify the given twitter dataset describing tweets on U.S airlines into positive, neutral and negative classes. Use suitable Naïve Bayes classification technique. (dataset given).

* Apply Text pre-processing task
* Train and test the dataset
* Show the confusion matrix.
* Compute Accuracy, Precision, Recall and F-score.

**CODE:**

from enum import unique

import pandas as pd

import numpy as np

import re

all\_words = []

word\_size\_class = {}

all\_sents = []

word\_frequency\_label = {}

defload\_data(filename):

data = pd.read\_csv(filename)

test\_index = data['airline\_sentiment'] == '?'

training\_data = data[-test\_index]

testing\_data = data[test\_index]

testing\_data.reset\_index(inplace = True, drop = True)

return(training\_data,testing\_data)

deffind\_vocab\_size(training\_data):

for index,sent inenumerate(training\_data['text']):

ext\_words = re.findall(r"([a-z0-9]+)",sent)

label = training\_data['airline\_sentiment'][index]

word\_size\_class[label] = word\_size\_class.get(label,0) + len(ext\_words)

all\_words.extend(ext\_words)

all\_sents.append(ext\_words)

unique\_words\_count = len(set(all\_words))

all\_words\_count = len(all\_words)

global unique\_words

unique\_words = list(set(all\_words))

return(unique\_words\_count,all\_words\_count)

deffind\_prior\_probabilties(training\_data):

class\_prior = {}

labels = training\_data['airline\_sentiment'].unique()

total = len(training\_data)

for l in labels:

class\_prior[l] = sum(training\_data['airline\_sentiment'] == l) / total

return class\_prior

deffind\_word\_frequency\_class(training\_data):

for word in unique\_words:

for index,sent\_vec inenumerate(all\_sents):

if word in sent\_vec:

if word notin word\_frequency\_label:

word\_frequency\_label[word] = {}

label = training\_data['airline\_sentiment'][index]

word\_frequency\_label[word][label] = word\_frequency\_label[word].get(label,0) + sent\_vec.count(word)

defdisplay\_conditional\_prob(vocab\_size,labels):

i=0

for word in word\_frequency\_label:

for label in labels:

num = word\_frequency\_label[word].get(label,0) + 1

denom = word\_size\_class[label] + vocab\_size

space = ""

print(f"P({word}/{label}) = {num}/{denom}{space\*(14-len(word))}",end="\t")

print()

defdisplay\_test\_results(data,labels,vocab\_size,class\_prior):

for i,sent inenumerate(data['text']):

ext\_words = re.findall(r"([a-z0-9]+)",sent)

probs = []

for label in labels:

prob = 1

for word in ext\_words:

class\_dict = word\_frequency\_label.get(word)

num = 0

denom = (word\_size\_class[label]+vocab\_size)

if class\_dict == None:

num = 1

else:

num = class\_dict.get(label,0) + 1

prob \*= (num/denom)

prior = class\_prior[label]

probs.append(prior\*prob)

probs = np.array(probs, dtype=np.float32)

index = np.argmax(probs)

print(f"\n{sent} ===>{labels[index]}{probs}")

data['airline\_sentiment'][i] = labels[index]

print("\nFinal Result:")

print(f"{data}\n")

defmain():

filename = "./Tweets Dataset.csv"

train\_data,test\_data = load\_data(filename)

print("Training Data:")

print(train\_data)

labels = train\_data['airline\_sentiment'].unique()

class\_prior = find\_prior\_probabilties(train\_data)

print(f"\nPrior Probabilities: {class\_prior}\n")

vocab\_size, total\_word\_count = find\_vocab\_size(train\_data)

print(f"Vocab size: {vocab\_size}")

print(f"Total words in train data: {total\_word\_count}\n")

find\_word\_frequency\_class(train\_data)

print("Formed a dictionary of words with respect to their frequency and class\n")

print('\nDisplaying all the conditional Probabilities:')

display\_conditional\_prob(vocab\_size,labels)

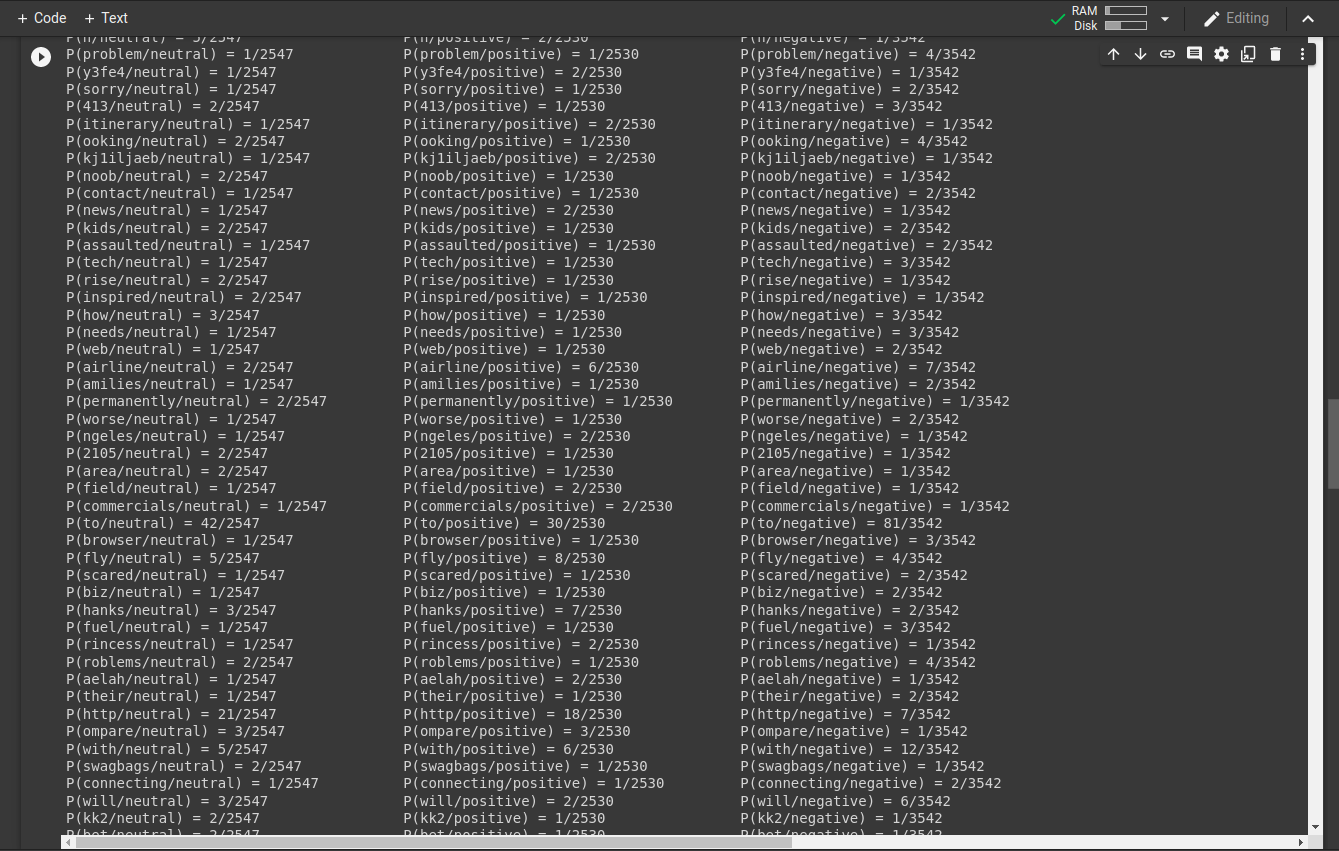
print("\nDisplaying the result on test sentences:")

display\_test\_results(test\_data,labels,vocab\_size,class\_prior)

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

main()

**OUTPUT:**

****

**CODE:**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Tweets Dataset.csv')

X = dataset.iloc[:,[2,9]].values

y = dataset.iloc[:,1].values

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Training the Naive Bayes model on the Training set

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

print(y\_pred)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score

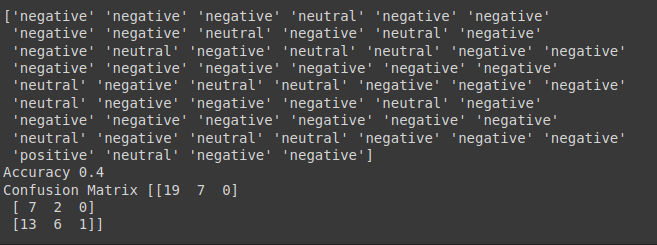
ac = accuracy\_score(y\_test,y\_pred)

cm = confusion\_matrix(y\_test, y\_pred)

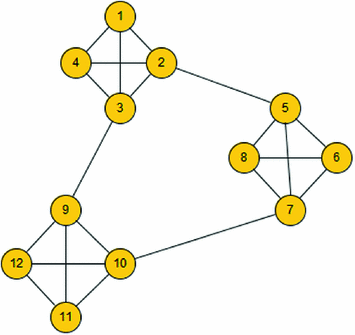
print("Accuracy",ac)

print("Confusion Matrix",cm)

**OUTPUT:**

****

1. Implement a suitable community detection algorithm for the detection and analysis of community structure for the following network graph datasets. Show the communities using graph coloring technique.



**CODE:**

import networkx as nx

from matplotlib import pyplot as plt

from networkx.algorithms.community.centrality import girvan\_newman

adjacency\_matrix=[

[0,1,1,1,0,0,0,0,0,0,0,0],

[1,0,1,1,1,0,0,0,0,0,0,0],

[1,1,0,1,1,0,0,0,1,0,0,0],

[1,1,1,0,0,0,0,0,0,0,0,0],

[0,1,0,0,0,1,1,1,0,0,0,0],

[0,0,0,0,1,0,1,1,0,0,0,0],

[0,0,0,0,1,1,0,1,0,1,0,0],

[0,0,0,0,1,1,1,0,0,0,0,0],

[0,0,1,0,0,0,0,0,0,1,1,1],

[0,0,0,0,0,0,1,0,1,0,1,1],

[0,0,0,0,0,0,0,0,1,1,0,1],

[0,0,0,0,0,0,0,0,1,1,1,0]

]

num\_vertices = 12

vertices\_list = ['1','2','3','4','5','6','7','8','9','10','11','12']

graph = nx.DiGraph()

# Load the nodes into the graph

graph.add\_nodes\_from(vertices\_list)

# Add the edges from the adjacency matrix

for i inrange(num\_vertices):

for j inrange(num\_vertices):

if adjacency\_matrix[i][j] == 1:

graph.add\_edge(vertices\_list[i], vertices\_list[j])

# Draw generated graph

nx.draw\_networkx(graph, pos=nx.circular\_layout(graph), arrows=True, with\_labels= True)

plt.show()

communities = girvan\_newman(graph)

node\_groups = []

for com innext(communities):

node\_groups.append(list(com))

print(node\_groups)

color\_map = []

for node in graph:

if node in node\_groups[0]:

color\_map.append('red')

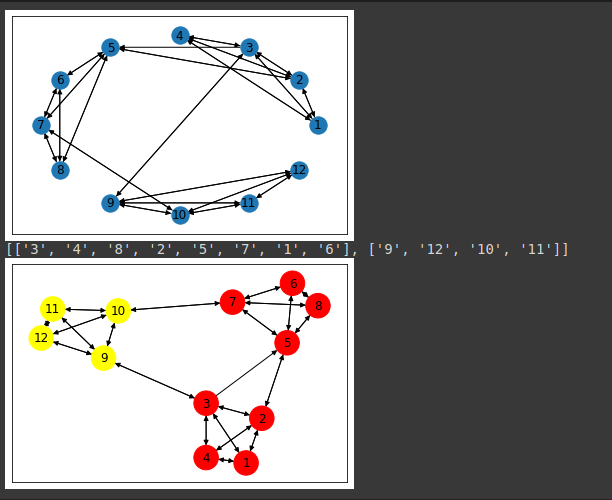
else:

color\_map.append('yellow')

graph = nx.draw\_networkx(graph,node\_size=600,node\_color=color\_map)

plt.show()

**OUTPUT:**

****