

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT
on

MACHINE LEARNING **(20CS6PCMAL)**

Submitted by

Sushma T P (1BM20CS416)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “**MACHINE LEARNING**” carried out by **Sushma T P (1BM20CS416)** who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Dr.K. Panimozhi
Assistant Professor
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Program 1 : FIND S

```
import csv

def updatehypothesis(x,h):
    if h == []:
        for i in range(0,len(x)):
            h.append("$")

    print("Initial State : ", h)
    return x

for i in range(0,len(x)):
    if x[i].upper() != h[i].upper() :
        h[i] = "?"

print("Most Specific Hypothesis : ", h)
return h

if __name__ == "__main__":
    data = []
    h = []

    with open('findsdataset.csv','r') as file :
        reader = csv.reader(file)
        print("Data Set : ")

        for row in reader:
            data.append(row)
```

```
print(row)
```

```
if data:
```

```
    for x in data:
```

```
        if x[-1].upper() == "YES":
```

```
            x.pop()
```

```
            h = updatehypothesis(x,h)
```

```
print("Maximally Specific Hypothesis : " , h)
```

1	sky	airtemp	humidity	wind	water	forecast	enjoysport
2	sunny	warm	normal	strong	warm	same	yes
3	sunny	warm	high	strong	warm	same	yes
4	rainy	cold	high	strong	warm	change	no
5	sunny	warm	high	strong	cool	change	yes

Data Set :

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
```

```
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
```

```
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
```

```
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']
```

```
Initial State : ['$' , '$' , '$' , '$' , '$' , '$']
```

```
Most Specific Hypothesis : ['sunny', 'warm', '?', 'strong', 'warm', 'same']
```

```
Most Specific Hypothesis : ['sunny', 'warm', '?', 'strong', '?', '?']
```

```
Maximally Specific Hypothesis : ['sunny', 'warm', '?', 'strong', '?', '?']
```

Program 2 : Candidate Elimination

```
import csv

a=[]

with open('../input/lab1find-s/Lab1 - Sheet1.csv','r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)

print(a)

print("")

num_attributes = len(a[0])-1
s=['0']*num_attributes

g=["?" for i in range(len(s))]

for i in range(0,len(a)):
    if a[i][num_attributes]=='yes':
        for j in range(0,num_attributes):
            if s[j]=='0' or s[j]==a[i][j]:
                s[j]=a[i][j]
            else:
                s[j]='?'
    else:
        for j in range(0,num_attributes):
            if(s[j] == a[i][j] or s[j] == '?'):
                g[j][j]='?'
            continue
```

```

else:
    g[j][j] = s[j]
for j in range(0,num_attributes):
    if s[j]!=g[j][j] or s[j]=='?':
        g[j][j]='?'

indices = [i for i, val in enumerate(g) if val == ['?', '?', '?', '?', '?', '?']]
for i in indices:
    g.remove(['?', '?', '?', '?', '?', '?'])
print("Specific hypothesis:",s)
print("General hypothesis:",g)

```

1	sky	airtemp	humidity	wind	water	forecast	enjoysport
2	sunny	warm	normal	strong	warm	same	yes
3	sunny	warm	high	strong	warm	same	yes
4	rainy	cold	high	strong	warm	change	no
5	sunny	warm	high	strong	cool	change	yes

```

[['sky', 'airtemp', 'humidity', 'wind', 'water', 'forecast', 'enjoysport'], ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'], ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]
Specific hypothesis: ['sunny', 'warm', '?', 'strong', '?', '?']
General hypothesis: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

```

Program 3 : ID3

```
import math

import csv

def load_csv(filename):

    lines=csv.reader(open(filename,"r"));

    dataset = list(lines)

    headers = dataset.pop(0)

    return dataset,headers


class Node:

    def __init__(self,attribute):

        self.attribute=attribute

        self.children=[]

        self.answer=""


def subtables(data,col,delete):

    dic={ }

    coldata=[row[col] for row in data]

    attr=list(set(coldata))

    counts=[0]*len(attr)

    r=len(data)

    c=len(data[0])

    for x in range(len(attr)):

        for y in range(r):

            if data[y][col]==attr[x]:

                counts[x]+=1
```

```

for x in range(len(attr)):
    dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
    pos=0
    for y in range(r):
        if data[y][col]==attr[x]:
            if delete:
                del data[y][col]
            dic[attr[x]][pos]=data[y]
            pos+=1
    return attr,dic

```

```

def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

```

```

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

```



```

total_size=len(data)
entropies=[0]*len(attr)
ratio=[0]*len(attr)

total_entropy=entropy([row[-1] for row in data])
for x in range(len(attr)):
    ratio[x]=len(dic[attr[x]])/(total_size*1.0)
    entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
    total_entropy-=ratio[x]*entropies[x]
return total_entropy

```

```

def build_tree(data,features):

```

```

    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

```

```

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]

```

```

    attr,dic=subtables(data,split,delete=True)

```

```
for x in range(len(attr)):
    child=build_tree(dic[attr[x]],fea)
    node.children.append((attr[x],child))
return node
```

```
def print_tree(node,level):
    if node.answer!="":
        print(" "*level,node.answer)
    return
```

```
print(" "*level,node.attribute)
for value,n in node.children:
    print(" "*(level+1),value)
    print_tree(n,level+2)
```

```
def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
```

```
dataset,features=load_csv("../input/playtennis/ID3.csv")
```

```
node1=build_tree(dataset,features)
```

```
print("The decision tree for the dataset using ID3 algorithm is")
```

```
print_tree(node1,0)
```

```
# testdata,features=load_csv("../input/playtennis/ID3.csv")
```

```
# print(features,"\n\n",testdata)
```

```
# for xtest in testdata:
```

```
#     print("The test instance:",xtest)
```

```
#     print("The label for test instance:",end=" ")
```

```
#     classify(node1,xtest,features)
```

1	Outlook	Temperature	Humidity	Wind	Answer
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no

Your Notebook is now running in the cloud.
Enter some code at the bottom of this console and press [Enter].

The decision tree for the dataset using ID3 algorithm is

Outlook

 sunny

 Humidity

 normal

 yes

 high

 no

 overcast

 yes

 rain

 Wind

 strong

 no

 weak

 yes

Program 4 : Navie Bayes

```
import pandas as pd

data = pd.read_csv('../input/playtennisnb/PlayTennis.csv')
data.head()

y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values

print(f'Target Values: {y}')
print(f'Features: \n{X}')

y_train = y[:8]
y_val = y[8:]

X_train = X[:8]
X_val = X[8:]

print(f'Number of instances in training set: {len(X_train)}")
print(f'Number of instances in testing set: {len(X_val)}")

class NaiveBayesClassifier:

    def __init__(self, X, y):

        self.X, self.y = X, y
```

```
self.N = len(self.X)
```

```
self.dim = len(self.X[0])
```

```
self.attrs = [[] for _ in range(self.dim)]
```

```
self.output_dom = { }
```

```
self.data = []
```

```
for i in range(len(self.X)):
```

```
    for j in range(self.dim):
```

```
        if not self.X[i][j] in self.attrs[j]:
```

```
            self.attrs[j].append(self.X[i][j])
```

```
    if not self.y[i] in self.output_dom.keys():
```

```
        self.output_dom[self.y[i]] = 1
```

```
    else:
```

```
        self.output_dom[self.y[i]] += 1
```

```
    self.data.append([self.X[i], self.y[i]])
```

```
def classify(self, entry):
```

```
    solve = None
```

```
    max_arg = -1
```

```
for y in self.output_dom.keys():
```

```
    prob = self.output_dom[y]/self.N
```

```
    for i in range(self.dim):
```

```
        cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
```

```
        n = len(cases)
```

```
        prob *= n/self.N
```

```
    if prob > max_arg:
```

```
        max_arg = prob
```

```
    solve = y
```

```
return solve
```

```
nbc = NaiveBayesClassifier(X_train, y_train)
```

```
total_cases = len(y_val)
```

```
good = 0
```

```
bad = 0
```

```
predictions = []
```

```
for i in range(total_cases):
```

```
    predict = nbc.classify(X_val[i])
```

```
    predictions.append(predict)
```

```
    if y_val[i] == predict:
```

```

        good += 1

    else:

        bad += 1

print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)

```

1	Outlook	Temperature	Humidity	Wind	Answer
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no

Your Notebook is now running in the cloud.

Enter some code at the bottom of this console and press [Enter].

```
PlayTennis Outlook Temperature Humidity Wind
0 No Sunny Hot High Weak
1 No Sunny Hot High Strong
2 Yes Overcast Hot High Weak
3 Yes Rain Mild High Weak
4 Yes Rain Cool Normal Weak
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
Number of instances in training set: 8
Number of instances in testing set: 6
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

Program 5 : Liner Regression

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('../input/salarydata/salaryData.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)

# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='green')
viz_train.plot(X_train, regressor.predict(X_train), color='black')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()
```

```
# Visualizing the Test set results
```

```
viz_test = plt
```

```
viz_test.scatter(X_test, y_test, color='green')
```

```
viz_test.plot(X_train, regressor.predict(X_train), color='black')
```

```
viz_test.title('Salary VS Experience (Test set)')
```

```
viz_test.xlabel('Year of Experience')
```

```
viz_test.ylabel('Salary')
```

```
viz_test.show()
```

	YearsExperience	Salary
1	1.1	39343
2	1.3	46205
3	1.5	37731
4	2.0	43525
5	2.2	39891
6	2.9	56642
7	3.0	60150
8	3.2	54445
9	3.2	64445
10	3.7	57189
11	3.9	63218
12	4.0	55794
13	4.0	56957
14	4.1	57081
15	4.5	61111
16	4.9	67938
17	5.1	66029
18	5.3	83088
19	5.9	81363
20	6.0	93940
21	6.0	91533



Program 6: K Means

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans

data = pd.read_csv("../input/K-Means/Dataset.csv")
data

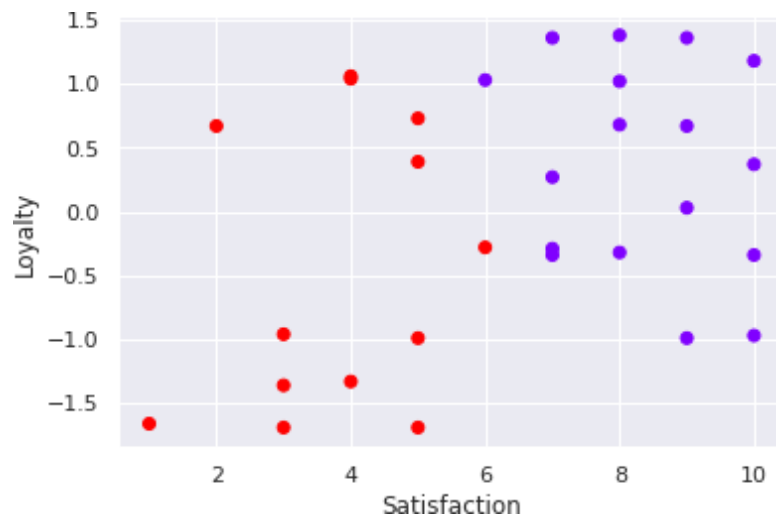
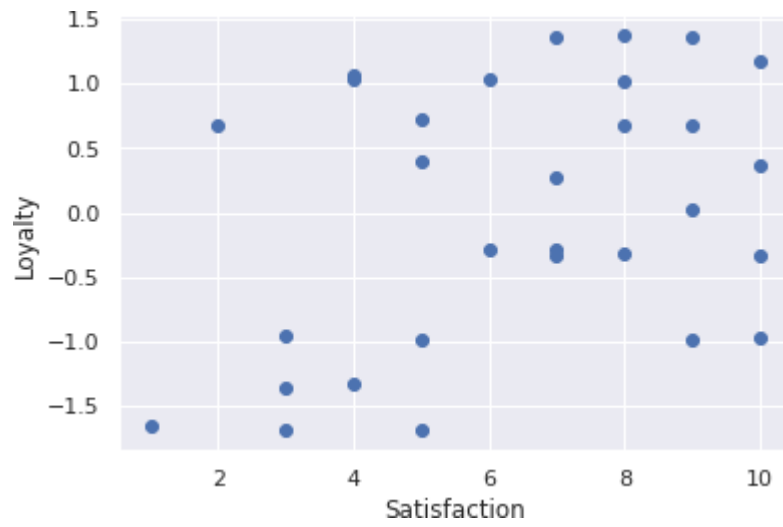
plt.scatter(data['Satisfaction'],data['Loyalty'])
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
plt.show()

x=data.copy()
kmean=KMeans(2)
kmean.fit(x)

clusters=x.copy()
clusters['cluster_pred']=kmean.fit_predict(x)

plt.scatter(clusters['Satisfaction'],clusters['Loyalty'],c=clusters['cluster_pred'],cmap='rainbow')
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
plt.ylabel('Loyalty')
plt.show()
```

Satisfaction	Loyalty
4	-1.33
6	-0.28
5	-0.99
7	-0.29
4	1.06
1	-1.66
10	-0.97
8	-0.32
8	1.02
8	0.68
10	-0.34
5	0.39
5	-1.69
2	0.67
7	0.27



Program 7: EM Algorithm

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)

plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print("The accuracy score of K-Mean: ", sm.accuracy_score(y, model.labels_))
print("The Confusion matrix of K-Mean:\n", sm.confusion_matrix(y, model.labels_))

from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
```

```

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)




y_gmm = gmm.predict(xs)

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

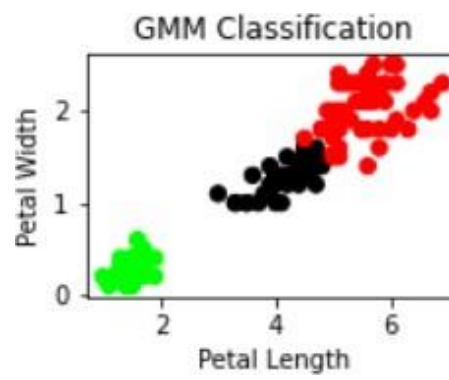
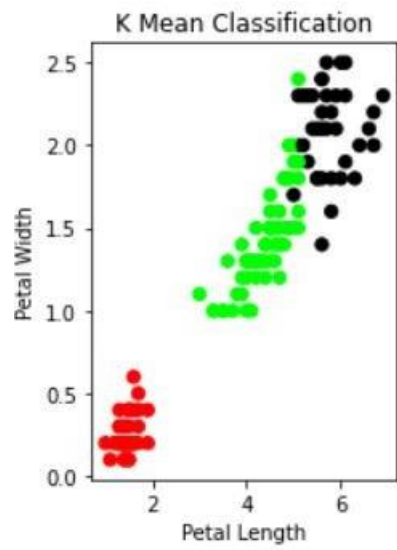
print('The accuracy score of EM: ', sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM:\n', sm.confusion_matrix(y, y_gmm))

```

```

<Figure size 1008x504 with 0 Axes>
Text(0, 0.5, 'Petal Width')

Text(0, 0.5, 'Petal Width')

The accuracy score of K-Mean: 0.8933333333333333
The Confusion matrix of K-Mean:
[[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
Text(0, 0.5, 'Petal Width')

The accuracy score of EM: 0.0
The Confusion matrix of EM:
[[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]

```

Program 8 : K Nearest Neighbor

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
Y = iris.target

print('sepal-length','sepal-width','petal-length','petal-width')
print(X)
print('target')
print(Y)

x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3)

classier = KNeighborsClassifier(n_neighbors=5)
classier.fit(x_train, y_train)

y_pred=classier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))

print('Accuracy')
print(classification_report(y_test,y_pred))
```

[illegible]

Confusion Matrix

[[19 0 0]]

$$\begin{bmatrix} 0 & 16 & 1 \end{bmatrix}$$

[0 0 9]]

Accuracy

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	0.94	0.97	17
2	0.90	1.00	0.95	9
accuracy			0.98	45
macro avg	0.97	0.98	0.97	45
weighted avg	0.98	0.98	0.98	45

Program 9: Bayesian Network

```
import numpy as np
import pandas as pd
import csv
import pgmpy
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination

#read Cleveland Heart Disease data
heartDisease = pd.read_csv('../input/heartdisease/heartDiseaseDataset.csv')
heartDisease = heartDisease.replace('?',np.nan)

#display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())

#display the Attributes names and datatypes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)

#Create Model-Bayesian Network
model =
BayesianNetwork([('age','heartDisease'),('sex','heartDisease'),('exang','heartDisease'),('cp','heartDisease'),('
restecg','heartDisease'),('heartDisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

#Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
heartDiseasetest_infer = VariableElimination(model)

#computing the Probability of heartDisease given restecg
print('\n 1.Probability of heartDisease given evidence= restecg :1')
q1=heartDiseasetest_infer.query(variables=['heartDisease'],evidence={'restecg':1})
print(q1)

#computing the Probability of heartDisease given cp
print('\n 2.Probability of heartDisease given evidence= cp:2 ')
q2=heartDiseasetest_infer.query(variables=['heartDisease'],evidence={'cp':2})
print(q2)
```

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartDisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
53	1	4	140	203	1	2	155	1	3.1	3	0	7	1
57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
44	1	2	120	263	0	0	173	0	0	1	0	7	0
52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
57	1	3	150	168	0	0	174	0	1.6	1	0	3	0
48	1	2	110	229	0	0	168	0	1	3	0	7	1
54	1	4	140	239	0	0	160	0	1.2	1	0	3	0
48	0	3	130	275	0	0	139	0	0.2	1	0	3	0
49	1	2	130	266	0	0	171	0	0.6	1	0	3	0
64	1	1	110	211	0	2	144	1	1.8	2	0	3	0
58	0	1	150	283	1	2	162	0	1	1	0	3	0

Sample instances from the dataset are given below

```

age sex cp trestbps chol fbs restecg thalach exang oldpeak slope \
0 63 1 1 145 233 1 2 150 0 2.3 3
1 67 1 4 160 286 0 2 108 1 1.5 2
2 67 1 4 120 229 0 2 129 1 2.6 2
3 37 1 3 130 250 0 0 187 0 3.5 3
4 41 0 2 130 204 0 2 172 0 1.4 1

```

```

ca thal heartDisease
0 0 6 0
1 3 3 2
2 2 7 1
3 0 3 0
4 0 3 0

```

Attributes and datatypes

```

age          int64
sex          int64
cp           int64
trestbps     int64
chol         int64
fbs          int64
restecg      int64
thalach      int64
exang        int64
oldpeak      float64
slope        int64
ca           int64
thal         int64
heartDisease int64
dtype: object

```

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of heartDisease given evidence= restecg :1

Finding Elimination Order: : 100%  4/4 [00:00<00:00, 81.79it/s]

Eliminating: exang: 100%  4/4 [00:00<00:00, 64.63it/s]

heartDisease	phi(heartDisease)
heartDisease(0)	0.1972
heartDisease(1)	0.1970
heartDisease(2)	0.1976
heartDisease(3)	0.1976
heartDisease(4)	0.2106

2. Probability of heartDisease given evidence= cp:2

Finding Elimination Order: : 100%  4/4 [00:00<00:00, 69.85it/s]

Eliminating: exang: 100%  4/4 [00:00<00:00, 89.18it/s]

heartDisease	phi(heartDisease)
heartDisease(0)	0.3138
heartDisease(1)	0.2150
heartDisease(2)	0.1552
heartDisease(3)	0.1633
heartDisease(4)	0.1527

Program 10: Locally Weighted Regression

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import numpy.linalg as np
from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,yamat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*yamat.T))
    return W

def localWeightRegression(xmat,yamat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,yamat,k)
    return ypred

#load data points
data = pd.read_csv('../input/tipsdata/tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)
# mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1] # print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
print(X)
```

```

#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)

xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='blue')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();

import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau):
    # add bias term
    x0 = np.r_[1, x0]
    # Add one to avoid the loss in information
    X = np.c_[np.ones(len(X)), X]
    # fit model: normal equations with kernel
    xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W
    beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product
    return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction

def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernel Bias Function

n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y:\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])

```



```

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

def plot_lwr(tau):
    # prediction through regression
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot

show(gridplot([[plot_lwr(10.), plot_lwr(1.)],
[plot_lwr(0.1), plot_lwr(0.01)]]))

from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)

```

```

    return ypred

# load data points
data = pd.read_csv('../input/tipsdata/tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,0.3)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();

```

total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2.0	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5.0	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3.0	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2
10.33	1.67	Female	No	Sun	Dinner	3
16.29	3.71	Male	No	Sun	Dinner	3
16.97	3.5	Female	No	Sun	Dinner	3
20.65	3.35	Male	No	Sat	Dinner	3
17.92	4.08	Male	No	Sat	Dinner	2

