VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT on

MACHINE LEARNING

Submitted by

SAQUIB NAUSHAD (1BM19CS144)

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

BENGALURU-560019 May-2022 to July-2022

(Autonomous Institution under VTU)

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 **SAQUIB NAUSHAD (1BM19CS144), who is bonafide student of B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in ComputerScience 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.

Name of the Lab-In charge:

Prof. Saritha A.N.
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course Outcome

EXPERIMENT 1

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

CODE:

```
import csv import pandas

as pd import numpy as

np


data = pd.read_csv("Desktop/data.csv") print(data,"\n")


#array of all the attributes d = np.array(data)[:,:-1]

print("\n

The attributes are: ",d)


target = np.array(data)[:,-1] print("\n

The target is: ",target)


def findS(c,t):     for i, val in

enumerate(t):

        if val == "Yes":

            specific_hypothesis = c[i].copy()

            break


    for i, val in enumerate(c):

        if t[i] == "Yes":            for x in

range(len(specific_hypothesis)):              if val[x]

!= specific_hypothesis[x]:
```

```
            specific_hypothesis[x] = '?'
        else:
pass

    return specific_hypothesis

print("\n The final hypothesis is:",findS(d,target))
```

OUTPUT:

```
  Weather Temperature Humidity    Wind Goes
0   Sunny        Warm     Mild  Strong  Yes
1   Rainy        Cold     Mild  Normal   No
2   Sunny    Moderate   Normal  Normal  Yes
3   Sunny        Cold     High  Strong  Yes


 The attributes are:  [['Sunny' 'Warm' 'Mild' 'Strong']
 ['Rainy' 'Cold' 'Mild' 'Normal']
 ['Sunny' 'Moderate' 'Normal' 'Normal']
 ['Sunny' 'Cold' 'High' 'Strong']]

 The target is:  ['Yes' 'No' 'Yes' 'Yes']

 The final hypothesis is: ['Sunny' '?' '?' '?']
```

EXPERIMENT 2

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

CODE:

```
import numpy as np  import

pandas as pd


data = pd.read_csv('Desktop/shape.csv') concepts

= np.array(data.iloc[:,0:-1]) print("\nInstances

are:\n",concepts) target

= np.array(data.iloc[:,-1]) print("\nTarget

Values are: ",target)


def learn(concepts, target):

    specific_h = concepts[0].copy()     print("\nInitialization of specific_h and

genearal_h")     print("\nSpecific Boundary: ", specific_h)     general_h = [["?" for

i in range(len(specific_h))] for i in range(len(specific_h))]     print("\nGeneric

Boundary: ",general_h)


    for i, h in enumerate(concepts):

print("\nInstance", i+1 , "is ", h)        if target[i]

== "yes":

           print("Instance is Positive ")

for x in range(len(specific_h)):                if

h[x]!= specific_h[x]:

specific_h[x] ='?'
```

```
general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
for x in range(len(specific_h)):                  if
h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n") print("Final
General_h: ", g_final, sep="\n")
```

OUTPUT:

```
In [3]: data = pd.read_csv('Desktop/shape.csv')
        concepts = np.array(data.iloc[:,0:-1])
        print("\nInstances are:\n",concepts)
        target = np.array(data.iloc[:,-1])
        print("\nTarget Values are: ",target)
```

```
Instances are:
 [['big' 'red' 'circle']
 ['small' 'red' 'triangle']
 ['small' 'red' 'circle']
 ['big' 'blue' 'circle']
 ['small' 'blue' 'circle']]

Target Values are:  ['no' 'no' 'yes' 'no' 'yes']
```

```
Initialization of specific_h and genearal_h

Specific Boundary:  ['big' 'red' 'circle']

Generic Boundary:  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', '?']]

Instance 1 is  ['big' 'red' 'circle']
Instance is Negative
Specific Bundary after  1 Instance is  ['big' 'red' 'circle']
Generic Boundary after  1 Instance is  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', '?']]



Instance 2 is  ['small' 'red' 'triangle']
Instance is Negative
Specific Bundary after  2 Instance is  ['big' 'red' 'circle']
Generic Boundary after  2 Instance is  [['big', '?', '?'], ['?', '?', '?'], ['?', '?', 'circle']]



Instance 3 is  ['small' 'red' 'circle']
Instance is Positive
Specific Bundary after  3 Instance is  ['?' 'red' 'circle']
Generic Boundary after  3 Instance is  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', 'circle']]



Instance 4 is  ['big' 'blue' 'circle']
Instance is Negative
Specific Bundary after  4 Instance is  ['?' 'red' 'circle']
Generic Boundary after  4 Instance is  [['?', '?', '?'], ['?', 'red', '?'], ['?', '?', '?']]



Instance 5 is  ['small' 'blue' 'circle']
Instance is Positive
Specific Bundary after  5 Instance is  ['?' '?' 'circle']
Generic Boundary after  5 Instance is  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', '?']]


Final Specific_h:
['?' '?' 'circle']
Final General_h:
[['?', '?', '?'], ['?', '?', '?'], ['?', '?', '?']]
```

EXPERIMENT 3

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

CODE:

WITHOUT ALGO:

```
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)

'''Main program''' dataset,features=load_csv("data.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("test.csv")

for xtest in testdata:
   print("The test instance:",xtest)     print("The
label for test instance:",end=" ")
classify(node1,xtest,features)
```

WITH ALGO:

```
import numpy as np
import pandas as pd import
math
```

```
data = pd.DataFrame(data=pd.read_csv('data.csv')) print(data)

def countPosNeg(data):
    pos = data.iloc[:,-1:].value_counts()['yes']     neg
= len(data) - pos     return pos, neg

def calcEntropy(pos, neg):
    entropy = -(pos/(pos+neg))*math.log2(pos/(pos+neg)) -(neg/(pos+neg))*math.log2(neg/(pos+neg))

    return entropy

def calcAverageInformation(data):     #
iterate through each attribute (col)     attribs
= data.iloc[:0,:-1].columns.values
print(attribs)

    for attrib in attribs:         # get possible
values         values =
data[attrib].unique()
valueEntropies = pd.DataFrame(0,
columns=['p','n','entropy'],
index=values)
        print()
print(attrib)
print(valueEntropies)

        # iterate through whole dataframe
        for i in data.index:
            print(data['Answer'][i])
```

```
if data['Answer'][i] == 'yes':
            valueEntropies[data[attrib]]['p'] += 1            elif
data['Answer'][i] == 'no':
            valueEntropies[data[attrib]]['n'] += 1


      for value in valueEntropies:
         value['entropy'] = calcEntropy(value['p'], value['n'])


      print(valueEntropies)


   return 10

calcAverageInformation(data)


def calcGain(entropy, avg_info):
return entropy - avg_info


# data for the total dataset

tot_pos, tot_neg = countPosNeg(data) tot_entropy
= calcEntropy(tot_pos, tot_neg) print(tot_entropy)


# iterate through dataset and calc pos, neg and entropy vals for each column
```

OUTPUT:

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   sunny
     Humidity
       normal
         yes
       high
         no
   rain
     Wind
       weak
         yes
       strong
         no
   overcast
     yes
```

```
import numpy as np
import pandas as pd
import math

data = pd.DataFrame(data=pd.read_csv('Desktop/data.csv'))
print(data)

# print(data['Answer'])
```

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

```python
            valueEntropies[data[attrib]]['p'] += 1
        elif data['Answer'][i] == 'no':
            valueEntropies[data[attrib]]['n'] += 1

    for value in valueEntropies:
        value['entropy'] = calcEntropy(value['p'], value['r

    print(valueEntropies)





# print(data['Outlook'].unique())

return 10

calcAverageInformation(data)
```

```
['Outlook' 'Temperature' 'Humidity' 'Wind']

Outlook
          p  n  entropy
sunny     0  0        0
overcast  0  0        0
rain      0  0        0
no
```

```python
def calcGain(entropy, avg_info):
    return entropy - avg_info
```

```python
# data for the total dataset

tot_pos, tot_neg = countPosNeg(data)
tot_entropy = calcEntropy(tot_pos, tot_neg)
print(tot_entropy)

# iterate through dataset and calc pos, neg and entropy vals f
```

```
Answer
yes      0.940286
dtype: float64
```

EXPERIMENT 4

Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

CODE:

```
import numpy as np import

matplotlib.pyplot as plt import pandas

as pd


dataset = pd.read_csv('salary_data.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='red') viz_train.plot(X_train,

regressor.predict(X_train), color='blue')
```

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

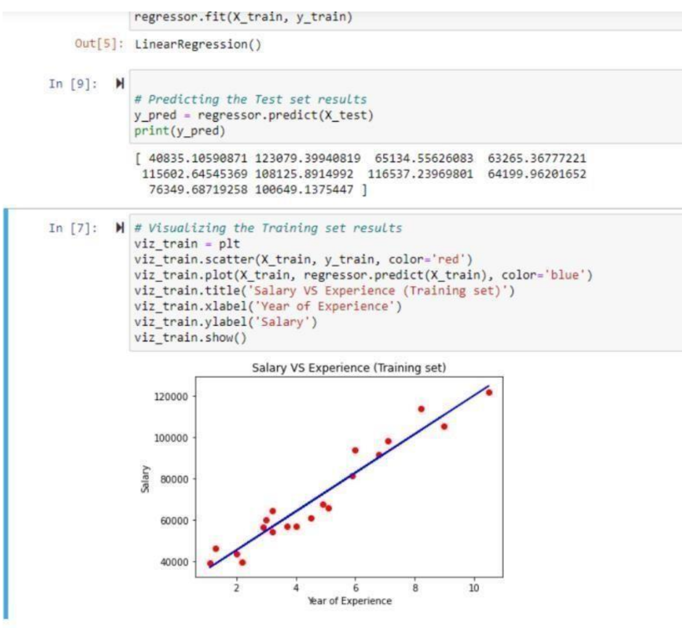
viz_test.scatter(X_test, y_test, color='red') viz_test.plot(X_train,

regressor.predict(X_train), color='blue') viz_test.title('Salary VS

Experience (Test set)') viz_test.xlabel('Year of Experience')

viz_test.ylabel('Salary') viz_test.show()

OUTPUT:

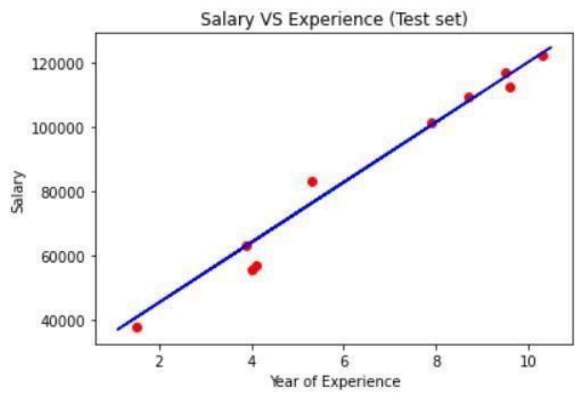


```
regressor.fit(X_train, y_train)

Out[5]: LinearRegression()

In [9]:
# Predicting the Test set results
y_pred = regressor.predict(X_test)
print(y_pred)

[ 40835.10590871 123079.39940819  65134.55626083  63265.36777221
 115602.64545369 108125.8914992  116537.23969801  64199.96201652
  76349.68719258 100649.1375447 ]

In [7]:
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()
```

Year of Experience

In [8]:

```
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```



EXPERIMENT 5

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

CODE:

```
import pandas as pd from sklearn.model_selection

import train_test_split from sklearn.naive_bayes import

GaussianNB from sklearn import metrics


df = pd.read_csv("Downloads/data.csv")

feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred',
'age']

predicted_class_names = ['diabetes']


X = df[feature_col_names].values  y = df[predicted_class_names].values


print(df.head) xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.40)


print ('\n the total number of Training Data :',ytrain.shape) print

('\n the total number of Test Data :',ytest.shape)


clf = GaussianNB().fit(xtrain,ytrain.ravel()) predicted

= clf.predict(xtest) predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])


print('\n Confusion matrix') print(metrics.confusion_matrix(ytest,predicted))
```

```
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

OUTPUT:

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  thickness  insulin   bmi  \
0           6           148            72         35        0  33.6
1           1            85            66         29        0  26.6
2           8           183            64          0        0  23.3
3           1            89            66         23       94  28.1
4           0           137            40         35      168  43.1
..        ...           ...           ...        ...      ...   ...
140         3           128            78          0        0  21.1
141         5           106            82         30        0  39.5
142         2           108            52         26       63  32.5
143        10           108            66          0        0  32.4
144         4           154            62         31      284  32.8

     diab_pred  age  diabetes
0        0.627   50         1
1        0.351   31         0
2        0.672   32         1
3        0.167   21         0
4        2.288   33         1
..         ...  ...       ...
140      0.268   55         0
141      0.286   38         0
142      0.318   22         0
143      0.272   42         1
144      0.237   23         0

[145 rows x 9 columns]>
```

```
print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)
```

```
 the total number of Training Data : (87, 1)

 the total number of Test Data : (58, 1)
```

```
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
```

In [7]:

```
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

```
 Confusion matrix
[[32  6]
 [12  8]]
```

In [8]:

```
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

```
 Accuracy of the classifier is 0.6896551724137931

 The value of Precision 0.5714285714285714

 The value of Recall 0.4
Predicted Value for individual Test Data: [1]
```

Experiment 6

Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

CODE:

```
import numpy as np import pandas as pd

import csv import pgmpy from

pgmpy.estimators import

MaximumLikelihoodEstimator from pgmpy.models import

BayesianModel from pgmpy.inference import

VariableElimination


#read Cleveland Heart Disease data heartDisease

= pd.read_csv('Downloads/data.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
=
BayesianModel([('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)
```

OUTPUT:

```
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: : 0% 0/4 [00:00<?, ?it/s]

Eliminating: cp: 100% 4/4 [00:00<00:00, 41.78it/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: : 0% 0/4 [00:00<?, ?it/s]

Eliminating: restecg: 100% 4/4 [00:00<00:00, 72.92it/s]

```
+-----------------+---------------------+
| heartDisease    |   phi(heartDisease) |
+=================+=====================+
| heartDisease(0) |              0.3138 |
+-----------------+---------------------+
| heartDisease(1) |              0.2150 |
+-----------------+---------------------+
| heartDisease(2) |              0.1552 |
+-----------------+---------------------+
| heartDisease(3) |              0.1633 |
+-----------------+---------------------+
| heartDisease(4) |              0.1527 |
+-----------------+---------------------+
```

Experiment 7

Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

CODE:

```
import matplotlib import

matplotlib.pyplot as plt import

seaborn as sns; sns.set() import numpy

as np



from sklearn.datasets import make_blobs X, y_true =

make_blobs(n_samples=300, centers=4,

                    cluster_std=0.60, random_state=0) plt.scatter(X[:,

0], X[:, 1], s=50)


from sklearn.cluster import KMeans kmeans

= KMeans(n_clusters=4) kmeans.fit(X)

y_kmeans = kmeans.predict(X)


plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')


centers = kmeans.cluster_centers_

plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)


import pandas as pd import numpy

as np heartDisease
```

=

pd.read_csv('Downloads/d

ata.csv') heartDisease = heartDisease.replace('?',np.

nan)

heartDisease.head()

trestbpsX = heartDisease.loc[:,'trestbps'] cholY

= heartDisease.loc[:,'chol'] plt.scatter(trestbpsX,

cholY, s=50)

kmeans2 = KMeans(n_clusters=2)

combined_list = list(zip(trestbpsX, cholY)) kmeans2.fit(combined_list)

y_kmeans2 = kmeans2.predict(combined_list)

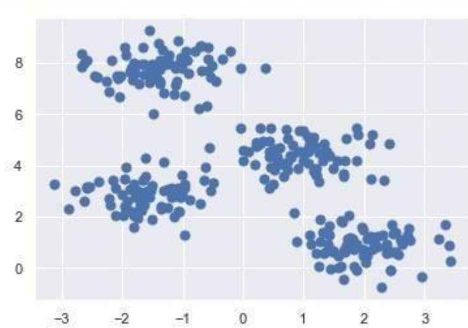
plt.scatter(trestbpsX, cholY, c=y_kmeans2, s=50, cmap='viridis')

centers = kmeans2.cluster_centers_

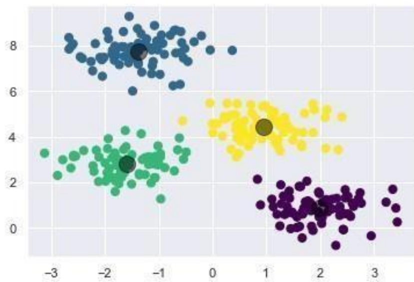
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)

OUTPUT:

```
Out[2]: <matplotlib.collections.PathCollection at 0x2006b964490>
```



Out[4]: <matplotlib.collections.PathCollection at 0x2006bc88610>

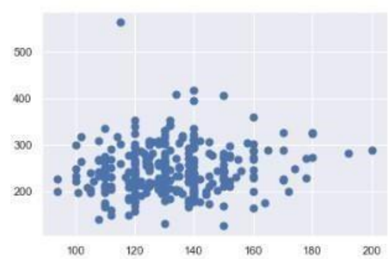


n [6]:
```python
import pandas as pd
import numpy as np
heartDisease = pd.read_csv('Downloads/data.csv')
heartDisease = heartDisease.replace('?',np.nan)

heartDisease.head()
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartDisease
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

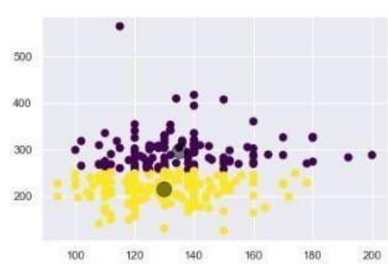
Out[8]: <matplotlib.collections.PathCollection at 0x2006c47ac40>



In [9]:
```python
kmeans2 = KMeans(n_clusters=2)
combined_list = list(zip(trestbpsX, cholY))
kmeans2.fit(combined_list)
y_kmeans2 = kmeans2.predict(combined_list)
```

In [10]:
```python
plt.scatter(trestbpsX, cholY, c=y_kmeans2, s=50, cmap='viridis')

centers = kmeans2.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
```

Out[10]: <matplotlib.collections.PathCollection at 0x2006c4d7d00>



Experiment 8

Write a program to implement k-Means algorithm to classify the iris data set. Print both correct and wrong predictions.

CODE:

```
from sklearn import datasets from

sklearn.cluster import KMeans from

sklearn.utils import shuffle import numpy

as np import pandas as pd


iris=datasets.load_iris()

X=iris.data

Y=iris.target


#Shuffle of Data

X,Y = shuffle(X,Y)


model=KMeans(n_clusters=3,init='k-means++',max_iter=10,n_init=1,random_state=3425)


#Training of the model model.fit(X)


# This is what KMeans thought (Prediction)

Y_Pred=model.labels_


from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Y,Y_Pred) print(cm)
```

```
from sklearn.metrics import accuracy_score

print(accuracy_score(Y,Y_Pred))

#Defining EM Model from sklearn.mixture import GaussianMixture
model2=GaussianMixture(n_components=3,random_state=3425)

#Training of the model model2.fit(X)

#Predicting classes for our data
Y_predict2= model2.predict(X)

#Accuracy of EM Model from sklearn.metrics import
confusion_matrix

cm=confusion_matrix(Y,Y_predict2) print(cm)

from sklearn.metrics import accuracy_score

print(accuracy_score(Y,Y_predict2))
```

OUTPUT:

```
[[ 0 50  0]
 [ 3  0 47]
 [36  0 14]]
0.09333333333333334
```

```
In [17]:
#Defining EM Model
from sklearn.mixture import GaussianMixture
model2=GaussianMixture(n_components=3,random_state=3425)

#Training of the model
model2.fit(X)
```

```
Out[17]:
GaussianMixture
GaussianMixture(n_components=3, random_state=3425)
```

```
In [18]:
#Predicting classes for our data
Y_predict2= model2.predict(X)

#Accuracy of EM Model
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Y,Y_predict2)
print(cm)

from sklearn.metrics import accuracy_score

print(accuracy_score(Y,Y_predict2))
```

```
[[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]
0.0
```

Experiment 9

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

CODE:

```
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))
```

OUTPUT:

```
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
```

```
In [21]: x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3)
```

```
In [22]: classier = KNeighborsClassifier(n_neighbors=5)
         classier.fit(x_train, y_train)

Out[22]: ▾ KNeighborsClassifier
         KNeighborsClassifier()
```

```
In [23]: y_pred=classier.predict(x_test)
```

```
In [24]: print('confusion matrix')
         print(confusion_matrix(y_test,y_pred))

confusion matrix
[[15  0  0]
 [ 0 17  2]
 [ 0  0 11]]
```

```
In [25]: print('accuracy')
         print(classification_report(y_test,y_pred))

accuracy
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      0.89      0.94        19
           2       0.85      1.00      0.92        11

    accuracy                           0.96        45
   macro avg       0.95      0.96      0.95        45
weighted avg       0.96      0.96      0.96        45
```

Experiment 10

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

CODE:

```
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('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

    # predict value   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 Kernal 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])

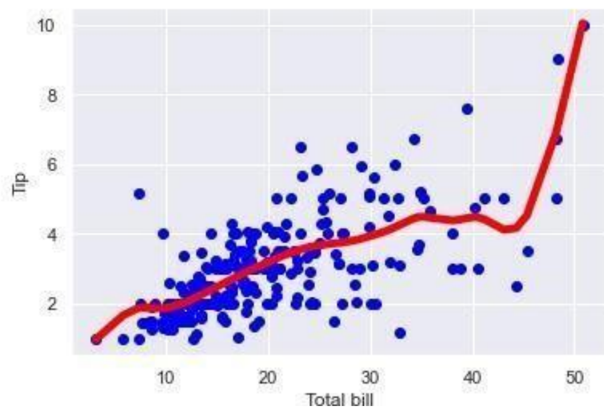
def plot_lwr(tau): # prediction through regression   prediction = [local_regression(x0, X, Y, tau) for x0 in domain]   plot = figure(plot_width=400, plot_height=400)

plot.title.text='tau=%g' % tau   plot.scatter(X, Y, alpha=.3)

plot.line(domain, prediction, line_width=2, color='red') return plot

OUTPUT:

```
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()
```




```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-2.7984698  -3.00877009 -3.05888439 -2.95096415 -2.94588394 -2.97666794
 -3.01995    -3.08887995 -2.92471686]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```

```
def plot_lwr(tau):
```