ML LAB-1 REPORT

PROGRAM-1
Date-10/04/2022

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

```
import csv

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


if__name__== "__main__
    ": data = []
    h = []

    # reading csv file
    with open('data.csv', 'r') as file:
        reader = csv.reader(file)
        print("Data: ")
        for row in reader:
            data.append(row)
            print(row)

    if data:
        for x in data:
            if x[-1].upper()=="YES":
                x.pop() # removing last field
                h = updateHypothesis(x,h)

    print("\nHypothesis: ",h)
```

SCREENSHOTS

```
Data:
['sunny', 'yes', 'normal', 'yes']
['rainy', 'no', 'mild', 'no']
['overcast', 'yes', 'normal', 'yes']
['sunny', 'no', 'normal', 'yes']
['cloudy', 'no', 'mild', 'no']

Hypothesis:  ['?', '?', 'normal']
```

PROGRAM-2

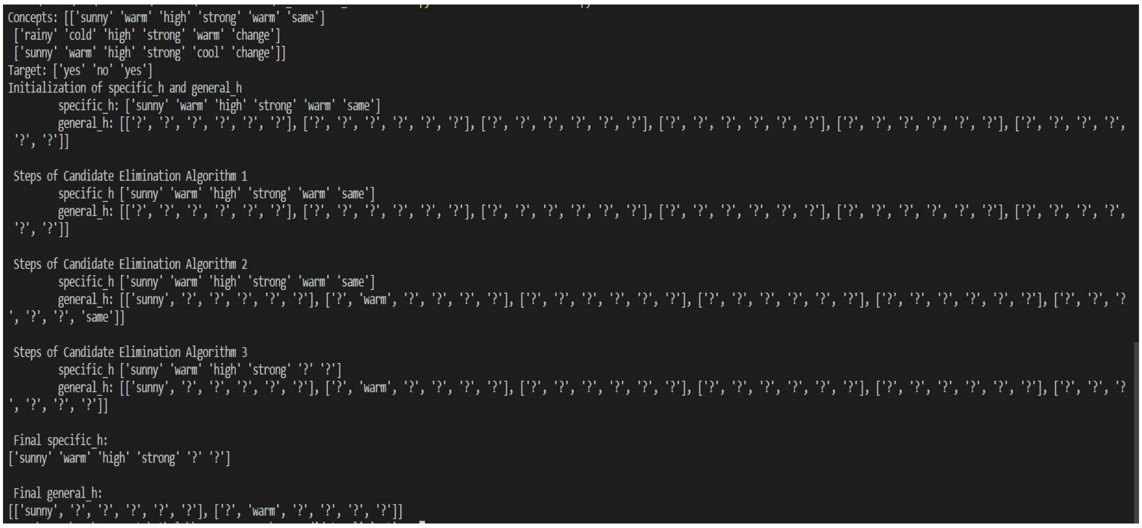
Date-24/03/2022

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.

```
import numpy as np
import pandas as pd

data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print('Concepts:', concepts)
target = np.array(data.iloc[:,-1])
print('Target:', target)

def learn(concepts, target):
    print("Initialization of specific_h and general_h")

    specific_h = concepts[0].copy()
    print('\t specific_h:', specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print('\t general_h:', general_h)


    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
        if target[i] == "no":
            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("\n Steps of Candidate Elimination Algorithm",i+1)
        print('\t specific_h', specific_h)
        print('\t general_h:', general_h)
```

```
    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("\n Final specific_h:", s_final, sep="\n")
print("\n Final general_h:", g_final, sep="\n")
```

SCREENSHOTS



```
Concepts: [['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
Target: ['yes' 'no' 'yes']
Initialization of specific_h and general_h
        specific_h: ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
        general_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
 '?', '?']]

 Steps of Candidate Elimination Algorithm 1
        specific_h ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
        general_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
 '?', '?']]

 Steps of Candidate Elimination Algorithm 2
        specific_h ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
        general_h: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', 'same']]

 Steps of Candidate Elimination Algorithm 3
        specific_h ['sunny' 'warm' 'high' 'strong' '?' '?']
        general_h: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?']]

 Final specific_h:
['sunny' 'warm' 'high' 'strong' '?' '?']

 Final general_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

PROGRAM-3

Date-31/03/2022

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.

```
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)

'''Main program'''
dataset,features=load_csv("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("id3_test.csv")

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

SCREENSHOTS

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   rain
     Wind
       strong
         no
       weak
         yes
   overcast
     yes
   sunny
     Humidity
       high
         no
       normal
         yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:   no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:   yes
```

PROGRAM-4

Date-21/04/2022

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

```
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("dataset.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)
```

SCREENSHOTS

```
<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]>

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

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

 Confusion matrix
[[28 10]
 [ 8 12]]

 Accuracy of the classifier is 0.6896551724137931

 The value of Precision 0.5454545454545454

 The value of Recall 0.6
Predicted Value for individual Test Data: [1]
```

```
 the total number of Training Data : (101, 1)

 the total number of Test Data : (44, 1)

 Confusion matrix
[[23  4]
 [ 6 11]]

 Accuracy of the classifier is 0.7727272727272727

 The value of Precision 0.7333333333333333

 The value of Recall 0.6470588235294118
Predicted Value for individual Test Data: [1]
```

```
 the total number of Training Data : (116, 1)

 the total number of Test Data : (29, 1)

 Confusion matrix
[[13  5]
 [ 3  8]]

 Accuracy of the classifier is 0.7241379310344828

 The value of Precision 0.6153846153846154

 The value of Recall 0.7272727272727273
Predicted Value for individual Test Data: [1]
```

PROGRAM-5

Date-28/04/2022

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

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


heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

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

print('\n Attributes and datatypes')
print(heartDisease.dtypes)

model=BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang',
'heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','c
hol')])

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

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

print('\n 1.Probability of HeartDisease given evidence=restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'re
stecg':1})
print(q1)


print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp
':2})
print(q2)
```

SCREENSHOTS

```
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
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 720.37it/s]
Eliminating: age: 100%|██████████| 5/5 [00:00<00:00, 66.59it/s]
 Inferencing with Bayesian Network:

 1.Probability of HeartDisease given evidence=restecg :1
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+
```

```
 2.Probability of HeartDisease given evidence= cp:2
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 839.60it/s]
Eliminating: age: 100%|██████████| 5/5 [00:00<00:00, 127.14it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

ML LAB-2 REPORT

PROGRAM-6
Date-02/06/2022

Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

```
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']

print(X.head())
print(y.head())

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

plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])

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')

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_))
```

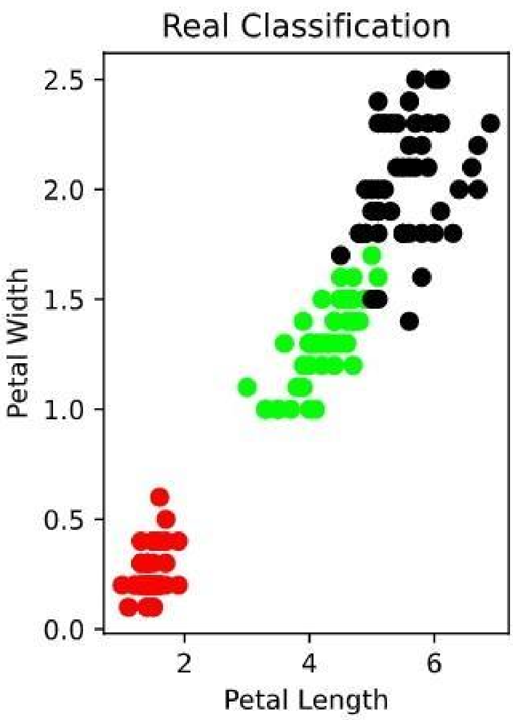
print('The Confusion matrixof K-Mean:\n ',sm.confusion_matrix(y, model.labels_))

SCREENSHOTS

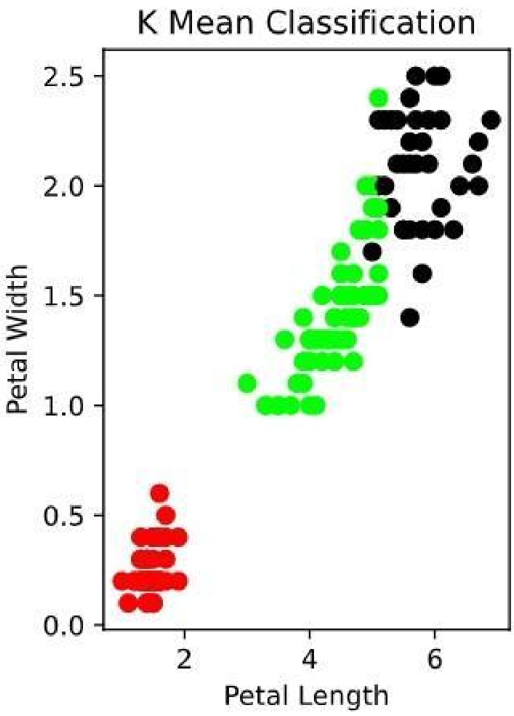
```
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
   Targets
0        0
1        0
2        0
3        0
4        0
```

```
KMeans(n_clusters=3)
```

```
Text(0, 0.5, 'Petal Width')
```



```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrixof K-Mean:
  [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
```



PROGRAM-7

Date-09/06/2022

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

```
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

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

from sklearn.mixture import GaussianMixture
model2=GaussianMixture(n_components=3,random_state=3425)
model2.fit(X)

Y_predict2= model2.predict(X)

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))
```

SCREENSHOTS

```
[[ 0 50  0]
 [48  0  2]
 [14  0 36]]
0.24

GaussianMixture(n_components=3, random_state=3425)

[[ 0 50  0]
 [45  0  5]
 [ 0  0 50]]
0.3333333333333333
```

PROGRAM-8

Date-09/06/2022

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

```
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))
```

SCREENSHOTS

```
confusion matrix
[[15  0  0]
 [ 0  7  2]
 [ 0  1 20]]

accuracy
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.88      0.78      0.82         9
           2       0.91      0.95      0.93        21

    accuracy                           0.93        45
   macro avg       0.93      0.91      0.92        45
weighted avg       0.93      0.93      0.93        45
```

PROGRAM-9

Date-09/06/2022

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

```
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)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

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()

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()
```

SCREENSHOTS

PROGRAM-10

Date-09/06/2022

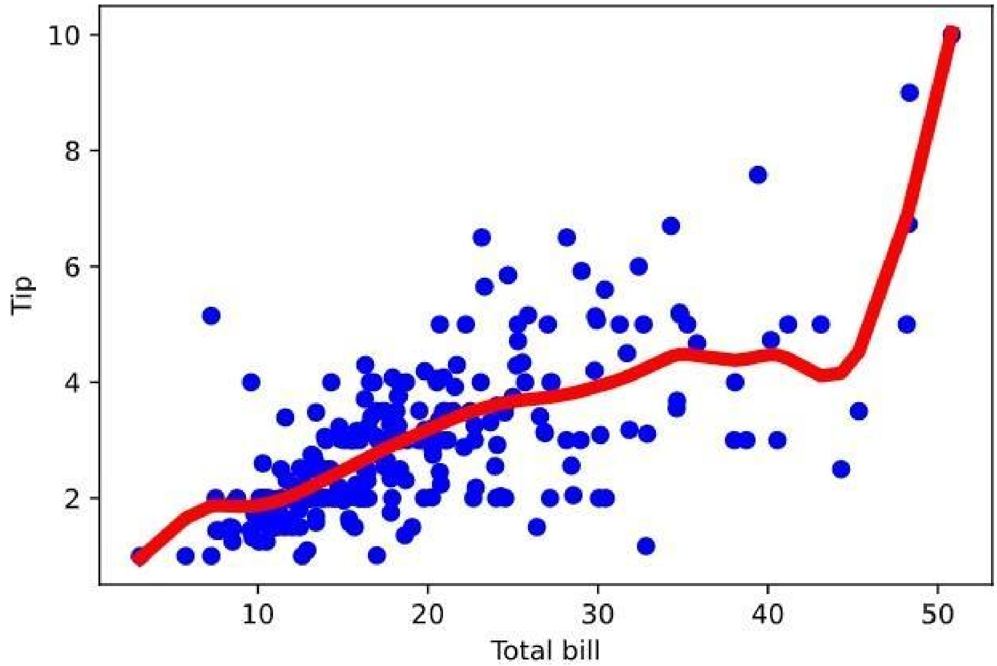
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
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

data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

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]
```

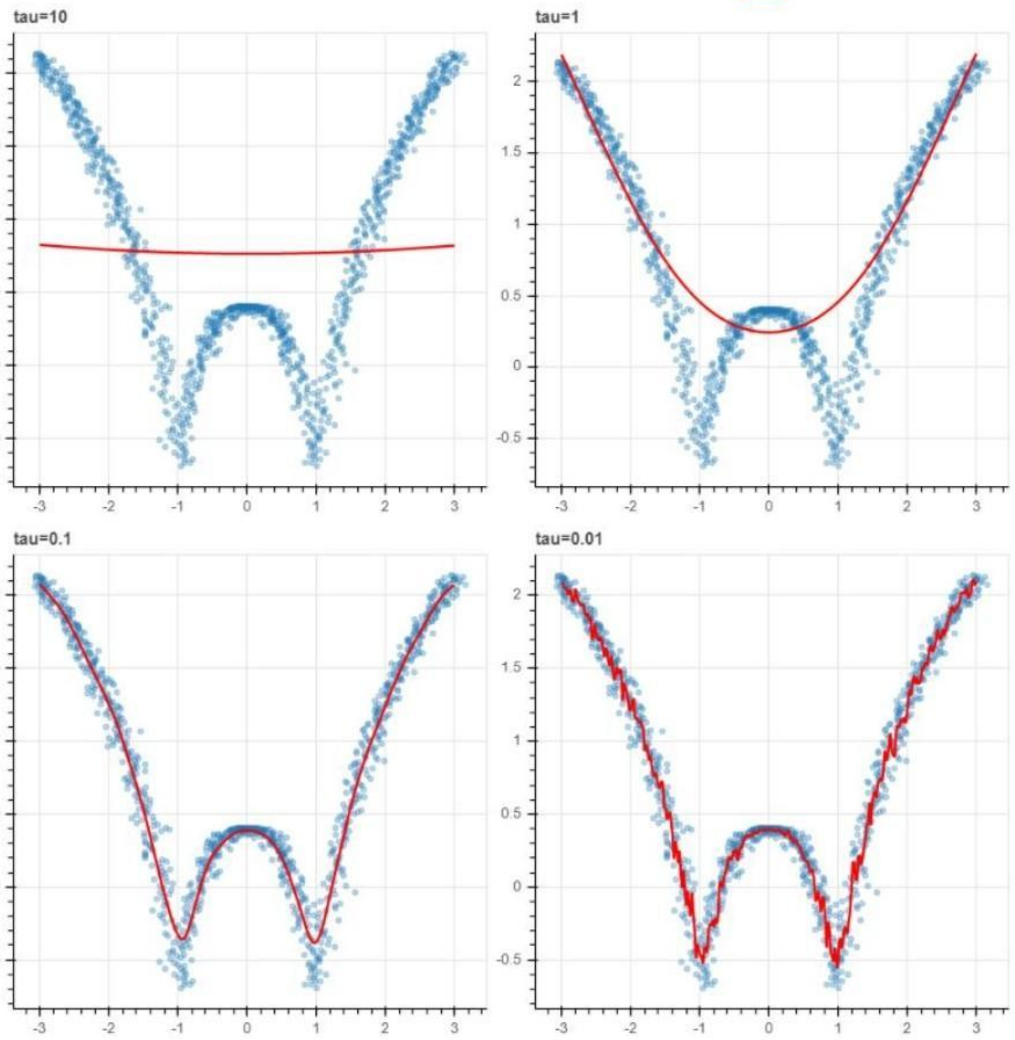
```
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE

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):
    x0 = np.r_[1, x0]
    X = np.c_[np.ones(len(X)), X]

    xw = X.T * radial_kernel(x0, X, tau)
    beta = np.linalg.pinv(xw @ X) @ xw @ Y
    return x0 @ beta

def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))


n = 1000
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])
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 = [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)]]))
```

SCREENSHOTS




```
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 :
 [-3.02807273 -2.87202266 -3.09630094 -3.18308318 -3.07358118 -3.01668872
 -3.03421482 -2.78784604 -2.99243688]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```


tau=10
tau=1
tau=0.1
tau=0.01