AIML LAB PROGRAMS

Package installation:

1. Open Jupyter notebook
2. Create an ipynb file
3. Execute the following command

   pip install -i https://test.pypi.org/simple/ GeneralPack
   pip install numpy
   pip install pandas
   pip install scikit-learn
   pip install matplotlib

Reference GIT - https://github.com/thesloppyguy/AIML
Package Link - https://test.pypi.org/project/GeneralPack/

Program 1

```
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1),('H', 7)] ,
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],
}


def heuristic(n):
    H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
```

```
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node
    while len(open_set) > 0 :
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                if m not in open_set and m not in closed_set:
```

```
                open_set.add(m)
                parents[m] = n
                g[m] = g[n] + weight
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n
                    if m in closed_set:
                        closed_set.remove(m)
                        open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

aStarAlgo('A', 'J')
```

OUTPUT:

Path found: ['A', 'F', 'G', 'I', 'J']

Program 2:

```
from GeneralPack.AOStar import program
def aoStar(obj, v, backTracking):
       print("HEURISTIC VALUES :", obj.H)
       print("SOLUTION GRAPH :", obj.solutionGraph)
       print("PROCESSING NODE :", v)

print("-----------------------------------------------------------------------
---------------------------")
       if obj.getStatus(v) >= 0:
           minimumCost, childNodeList =
obj.computeMinimumCostChildNodes(v)
           obj.setHeuristicNodeValue(v, minimumCost)
           obj.setStatus(v,len(childNodeList))
           solved=True
           for childNode in childNodeList:
               obj.parent[childNode]=v
               if obj.getStatus(childNode)!=-1:
                   solved=solved & False
           if solved==True:
               obj.setStatus(v,-1)
               obj.solutionGraph[v]=childNodeList
           if v!=obj.start:
               aoStar(obj,obj.parent[v], True)
           if backTracking==False:
               for childNode in childNodeList:
                   obj.setStatus(childNode,0)
                   aoStar(obj,childNode, False)
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H':
7, 'I': 7, 'J':1, 'T': 3}
graph1 = {
   'A': [[('B', 1), ('C', 1)], [('D', 1)]],
```

```
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}
G1= program(graph1, h1, 'A')
aoStar(G1,'A',False)
print(G1.solutionGraph)
```

OUTPUT:

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : B

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : G

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : B

HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : I

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': []}

PROCESSING NODE : G

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I']}
PROCESSING NODE : B

HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : C

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : J

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE : C

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE : A

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

PROGRAM 3:

```
from GeneralPack.CE import program
import csv
def candidate_elimination(obj,examples):
   domains = obj.get_domains(examples)[:-1]

   G = set([obj.g_0(len(domains))])
   S = set([obj.s_0(len(domains))])
   i=0
   print("\n G[{0}]:".format(i),G)
   print("\n S[{0}]:".format(i),S)
   for xcx in examples:
       i=i+1
       x, cx = xcx[:-1], xcx[-1]
       if cx=='Yes':
           G = {g for g in G if obj.fulfills(x, g)}
           S = obj.generalize_S(x, G, S)
       else:
           S = {s for s in S if not obj.fulfills(x, s)}
           G = obj.specialize_G(x, domains, G, S)
       print("\n G[{0}]:".format(i),G)
       print("\n S[{0}]:".format(i),S)
   return
examples=[]
with
open(r'/Users/sahil/Documents/PROGRAMS/Python/AIML/Data/examples.csv'
) as csvFile:
   examples = [tuple(line) for line in csv.reader(csvFile)]
obj=program()
obj.get_domains(examples)
candidate_elimination(obj,examples)
```

OUTPUT:

```
G[0]: {('?', '?', '?', '?', '?', '?')}

 S[0]: {('0', '0', '0', '0', '0', '0')}

 G[1]: {('?', '?', '?', '?', 'Cool', '?'), ('?', '?', '?', 'Strong',
'?', '?'), ('Rainy', '?', '?', '?', '?', '?'), ('?', 'Warm', '?',
'?', '?', '?'), ('?', '?', 'High', '?', '?', '?'), ('?', '?',
'Normal', '?', '?', '?'), ('Sunny', '?', '?', '?', '?', '?'), ('?',
'?', '?', '?', '?', 'Same'), ('?', 'Cold', '?', '?', '?', '?'), ('?',
'?', '?', '?', '?', 'Change'), ('?', '?', '?', '?', 'Warm', '?')}

 S[1]: {('0', '0', '0', '0', '0', '0')}

 G[2]: {('?', '?', '?', 'Strong', '?', '?'), ('?', 'Warm', '?', '?',
'?', '?'), ('?', '?', '?', '?', '?', 'Same'), ('?', '?', 'Normal',
'?', '?', '?'), ('Sunny', '?', '?', '?', '?', '?'), ('?', '?', '?',
'?', 'Warm', '?')}

 S[2]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}

 G[3]: {('?', '?', '?', 'Strong', '?', '?'), ('?', 'Warm', '?', '?',
'?', '?'), ('?', '?', '?', '?', '?', 'Same'), ('Sunny', '?', '?',
'?', '?', '?'), ('?', '?', '?', '?', 'Warm', '?')}

 S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

 G[4]: {('?', '?', '?', '?', '?', 'Same'), ('Sunny', '?', '?', '?',
'?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

 S[4]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

 G[5]: {('?', 'Warm', '?', '?', '?', '?'), ('Sunny', '?', '?', '?',
'?', '?')}

 S[5]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}
```

PROGRAM 4:

```
from GeneralPack.ID32 import program
import csv

def decision_tree(obj,data,labels):
   classList=[rec[-1]for rec in data]
   if classList.count(classList[0])==len(classList):
       return classList[0]
   maxGainNode=obj.attribute_selection(data)
   treelabel=labels[maxGainNode]
   theTree={treelabel:{}}
   del(labels[maxGainNode])
   nodeValues=[rec[maxGainNode]for rec in data]
   uniqueValues=set(nodeValues)
   for value in uniqueValues:
       sublabels=labels[:]

theTree[treelabel][value]=obj.decision_tree(obj.dataset_split(data,ma
xGainNode,value),sublabels)
   return theTree


with
open('/Users/sahil/Documents/PROGRAMS/Python/AIML/Data/tennis.csv','r
') as csvfile:
   fdata = [line.strip() for line in csvfile]
   meta = fdata[0].split(',')
   data = [ x.split(',')  for x in fdata[1:]]


obj=program()
tree=decision_tree(obj,data,meta)
```

```python
obj.print_tree(tree)
```

OUTPUT:

```
Outlook
  overcast
    d= yes
  rain
    Wind
        strong
                d= no
        weak
                d= yes
  sunny
    Humidity
        high
                d= no
        normal
                d= yes
```

PROGRAM 5:

```
import numpy as np

def sigmoid(x):
   return 1/(1+np.exp(-x))

def sigmoid_derivative(x):
   return x * (1 - x)

X = np.array(([2, 9], [1, 5], [3, 6]))
y = np.array(([92], [86], [89]))
y = y/100

np.random.seed(1)
synapse_0 = 2*np.random.random((2,3)) - 1
synapse_1 = 2*np.random.random((3,1)) - 1

for iter in range(10000):
   # Forward propagation
   layer_0 = X
   layer_1 = sigmoid(np.dot(layer_0, synapse_0))
   layer_2 = sigmoid(np.dot(layer_1, synapse_1))
   # Forward propagation
   layer_2_error = y - layer_2
   layer_2_delta = layer_2_error * sigmoid_derivative(layer_2)
   # Forward propagation
   layer_1_error = layer_2_delta.dot(synapse_1.T)
   layer_1_delta = layer_1_error * sigmoid_derivative(layer_1)
   # Update weights
```

```
    synapse_1 += layer_1.T.dot(layer_2_delta)
    synapse_0 += layer_0.T.dot(layer_1_delta)
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,layer_2)
```

OUTPUT

```
Input:
[[2 9]
 [1 5]
 [3 6]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.9056794 ]
 [0.87185295]
 [0.89109416]]
```

PROGRAM 6:

```
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


data =
pd.read_csv('/Users/sahil/Documents/PROGRAMS/Python/AIML/Data/tennisd
ata.csv')
print("The first 5 Values of data is :\n", data.head())

data=data.apply(LabelEncoder().fit_transform)

X = data.iloc[:, :-1]
print("\nThe First 5 values of the train data is\n", X.head())

y = data.iloc[:, -1]
print("\nThe First 5 values of train output is\n", y.head())

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =
0.20,random_state=1)

classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```python
print("Accuracy is:", accuracy_score(classifier.predict(X_test),
y_test))
```

OUTPUT:

```
The first 5 Values of data is :
     Outlook Temperature Humidity   Windy PlayTennis
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

The First 5 values of the train data is
    Outlook  Temperature  Humidity  Windy
0         2            1         0      1
1         2            1         0      0
2         0            1         0      1
3         1            2         0      1
4         1            0         1      1

The First 5 values of train output is
 0    0
1    0
2    1
3    1
4    1
Name: PlayTennis, dtype: int64
Accuracy is: 0.6666666666666666
```

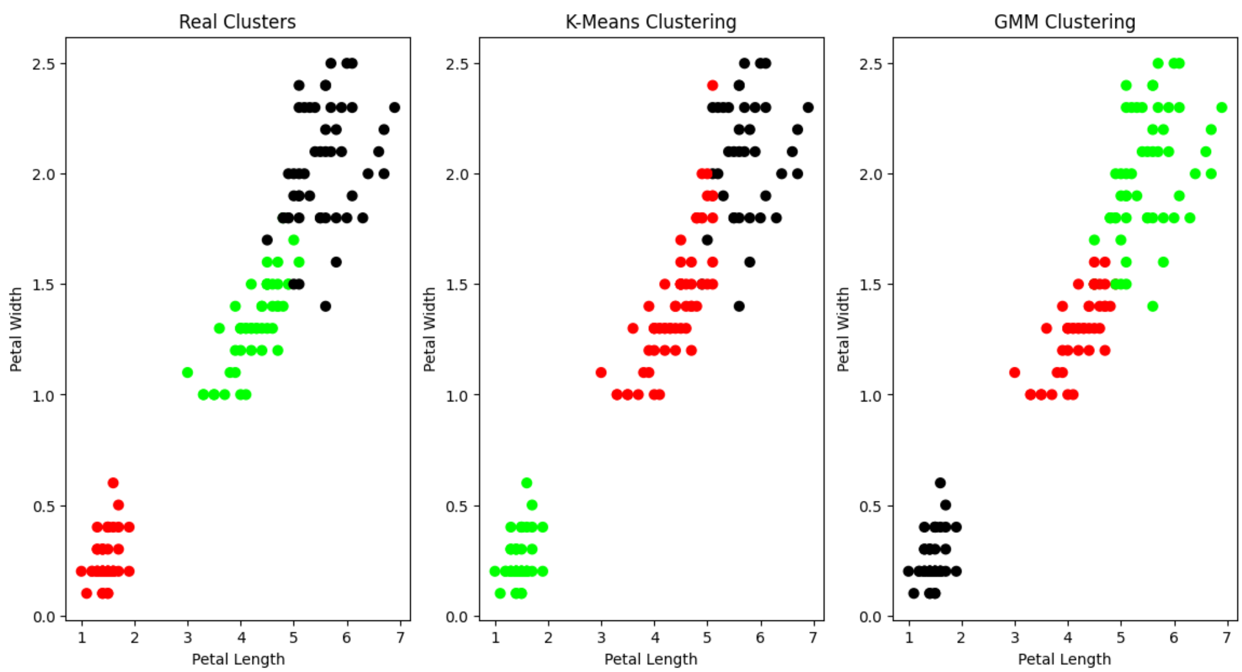
PROGRAM 7:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture


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']

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

plt.subplot(1, 3, 1)
```

```
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets],
s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

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

plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_],
s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm = gmm.predict(xs)

plt.subplot(1, 3, 3)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[y_cluster_gmm%3], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')
```

OUTPUT:



PROGRAM 8:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

iris=datasets.load_iris()
x_train, x_test, y_train, y_test =
train_test_split(iris.data,iris.target,test_size=0.1,random_state=1)
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)

print(iris.target_names)
for r in range(0,len(x_test)):
   print(" Sample:", str(x_test[r]), " Actual-label:",
str(y_test[r])," Predicted-label:", str(y_pred[r]))

print("Classification Accuracy :" , classifier.score(x_test,y_test))
```

OUTPUT:

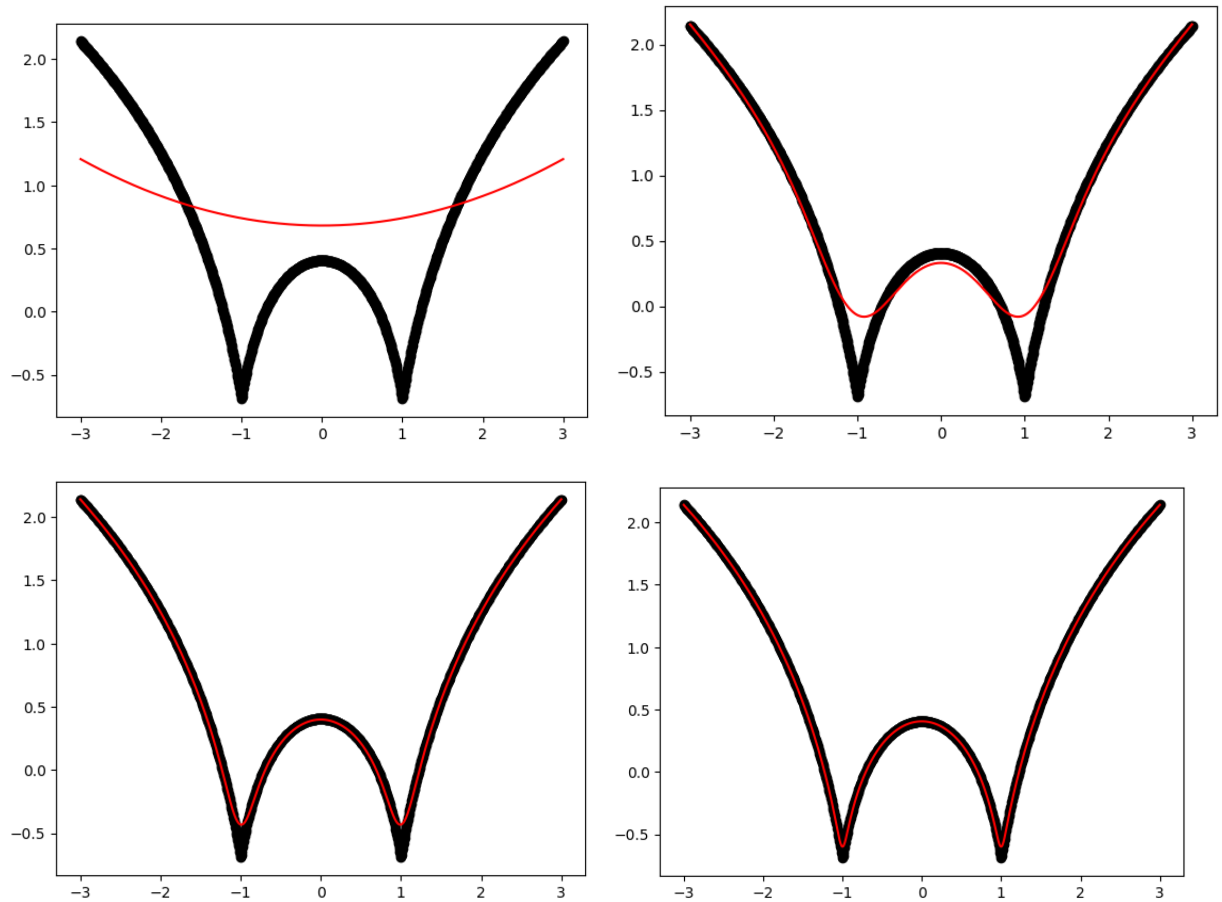
```
['setosa' 'versicolor' 'virginica']
 Sample: [5.8 4.  1.2 0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [5.1 2.5 3.  1.1]  Actual-label: 1  Predicted-label: 1
 Sample: [6.6 3.  4.4 1.4]  Actual-label: 1  Predicted-label: 1
 Sample: [5.4 3.9 1.3 0.4]  Actual-label: 0  Predicted-label: 0
 Sample: [7.9 3.8 6.4 2. ]  Actual-label: 2  Predicted-label: 2
```

```
 Sample: [6.3 3.3 4.7 1.6]  Actual-label: 1  Predicted-label: 1
 Sample: [6.9 3.1 5.1 2.3]  Actual-label: 2  Predicted-label: 2
 Sample: [5.1 3.8 1.9 0.4]  Actual-label: 0  Predicted-label: 0
 Sample: [4.7 3.2 1.6 0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [6.9 3.2 5.7 2.3]  Actual-label: 2  Predicted-label: 2
 Sample: [5.6 2.7 4.2 1.3]  Actual-label: 1  Predicted-label: 1
 Sample: [5.4 3.9 1.7 0.4]  Actual-label: 0  Predicted-label: 0
 Sample: [7.1 3.  5.9 2.1]  Actual-label: 2  Predicted-label: 2
 Sample: [6.4 3.2 4.5 1.5]  Actual-label: 1  Predicted-label: 1
 Sample: [6.  2.9 4.5 1.5]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 1.0
```

PROGRAM 9:

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

```python
def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    xw = (X.T) * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau))
    beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0
    return beta

def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.show()

X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)

draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```



2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3