ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)

MANUAL

FOR VII SEMESTER B.E

(As per Visveswaraya Technological University Syllabus)

2023-24

Compiled By:

Prof.D.Saritha
Assistant Professor
` Dept.of CSE

Prof.Soumya George
Asst.Professor
Dept.of CSE

Name:____________

USN:_____________

A P S COLLEGE OF ENGINEERING
Anantha Gnana Gangothri Campus, Somanahalli,
Kanakapura Road, Bengaluru – 5600082
Department of Computer Science and Engineering

Program 1

Implement A* Algorithm

Find the most cost effective path to read from start state A to final state J using A* Algorithm.

Step 1: Place the starting node into OPEN and find its f (n) value.

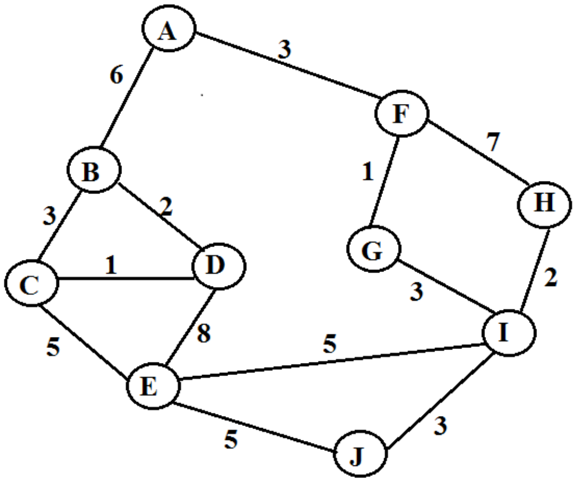
Step 2: Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

Step 5: Go to Step-2.

Step 6: Exit.



Heuristic Values

A	B	C	D	E	F	G	H	I	J
10	8	5	7	3	6	5	3	1	

```
defaStarAlgo(start_node, stop_node):

open_set = set(start_node)
closed_set = set()
        g = {} #store distance from starting node
parents = {}# parents contains an adjacency map of all nodes

        #ditance of starting node from itself is zero
g[start_node] = 0
        #start_node is root node i.e it has no parent nodes
        #so start_node is set to its own parent node
parents[start_node] = start_node


whilelen(open_set) > 0:
            n = None

            #node with lowest f() is found
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):
                   #nodes 'm' not in first and last set are added to first
                    #n is set its parent
if m not in open_set and m not in closed_set:
open_set.add(m)
parents[m] = n
g[m] = g[n] + weight


                    #for each node m,compare its distance from start i.e g(m) to the
                    #from start through n node
else:
if g[m] > g[n] + weight:
                            #update g(m)
```

```
g[m] = g[n] + weight
                   #change parent of m to n
parents[m] = n

                   #if m in closed set,remove and add to open
if m in closed_set:
closed_set.remove(m)
open_set.add(m)

if n == None:
print('Path does not exist!')
return None

        # if the current node is the stop_node
        # then we begin reconstructin the path from it to the start_node
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



        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
open_set.remove(n)
closed_set.add(n)

print('Path does not exist!')
return None

#define fuction to return neighbor and its distance
#from the passed node
```

```
defget_neighbors(v):
if v in Graph_nodes:
returnGraph_nodes[v]
else:
return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
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
        }

returnH_dist[n]

#Describe your graph here
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)],

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

Output

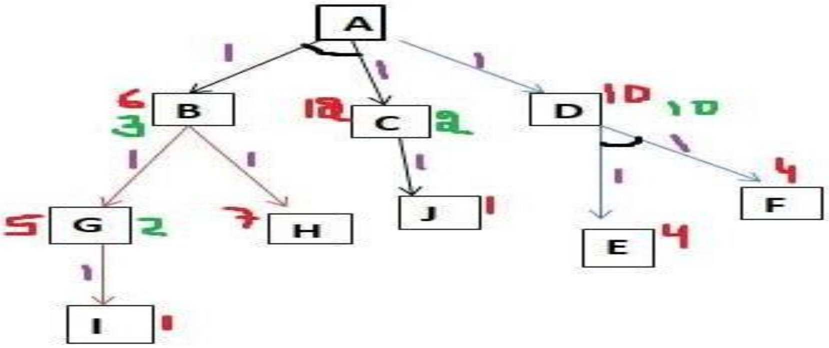
```
Path found: ['A', 'F', 'G', 'I', 'J']
```

Program 2

Implement AO* Algorithm

Algorithm

1. It is an informed search and works as Best First Search.
2. AO* algorithm is based on problem decomposition.
3. It represents an AND-OR graph algorithm that is used to find more than one solution.
4. It is an efficient method to explore a solution path.



```
class Graph:
def __init__(self, graph, heuristicNodeList, startNode):  #instantiate graph object
with graph topology, heuristic values, start node

self.graph = graph
self.H=heuristicNodeList
self.start=startNode
self.parent={}
self.status={}
self.solutionGraph={}

defapplyAOStar(self):         # starts a recursive AO* algorithm
self.aoStar(self.start, False)
```

```
defgetNeighbors(self, v):     # gets the Neighbors of a given node
returnself.graph.get(v,'')

defgetStatus(self,v):         # return the status of a given node
returnself.status.get(v,0)

defsetStatus(self,v, val):    # set the status of a given node
self.status[v]=val

defgetHeuristicNodeValue(self, n):
returnself.H.get(n,0)     # always return the heuristic value of a given node

defsetHeuristicNodeValue(self, n, value):
self.H[n]=value           # set the revised heuristic value of a given node


defprintSolution(self):
print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START
NODE:",self.start)
print("---------------------------------------------------------")
print(self.solutionGraph)
print("---------------------------------------------------------")

def computeMinimumCostChildNodes(self, v):  # Computes the Minimum Cost of
child nodes of a given node v
minimumCost=0
costToChildNodeListDict={}
costToChildNodeListDict[minimumCost]=[]
flag=True
for nodeInfoTupleList in self.getNeighbors(v):  # iterate over all the set of child
node/s
cost=0
nodeList=[]
for c, weight in nodeInfoTupleList:
```

```
cost=cost+self.getHeuristicNodeValue(c)+weight
nodeList.append(c)

if flag==True:                    # initialize Minimum Cost with the cost of first set of
child node/s
minimumCost=cost
costToChildNodeListDict[minimumCost]=nodeList      # set the Minimum Cost
child node/s
flag=False
else:                             # checking the Minimum Cost nodes with the current
Minimum Cost
ifminimumCost>cost:
minimumCost=cost
costToChildNodeListDict[minimumCost]=nodeList  # set the Minimum Cost child
node/s


returnminimumCost, costToChildNodeListDict[minimumCost]   # return Minimum
Cost and Minimum Cost child node/s


defaoStar(self, v, backTracking):     # AO* algorithm for a start node and
backTracking status flag

print("HEURISTIC VALUES  :", self.H)
print("SOLUTION GRAPH    :", self.solutionGraph)
print("PROCESSING NODE   :", v)
print("-----------------------------------------------------------------------------------------")

ifself.getStatus(v) >= 0:        # if status node v >= 0, compute Minimum Cost nodes
of v
minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
self.setHeuristicNodeValue(v, minimumCost)
self.setStatus(v,len(childNodeList))
```

```
solved=True                  # check the Minimum Cost nodes of v are solved
forchildNode in childNodeList:
self.parent[childNode]=v
ifself.getStatus(childNode)!=-1:
solved=solved & False

if solved==True:             # if the Minimum Cost nodes of v are solved, set the
current node status as solved(-1)
self.setStatus(v,-1)
self.solutionGraph[v]=childNodeList # update the solution graph with the solved
nodes which may be a part of solution



if v!=self.start:           # check the current node is the start node for backtracking the
current node value
self.aoStar(self.parent[v], True)   # backtracking the current node value with
backtracking status set to true

ifbackTracking==False:     # check the current call is not for backtracking
forchildNode in childNodeList:   # for each Minimum Cost child node
self.setStatus(childNode,0)   # set the status of child node to 0(needs exploration)
self.aoStar(childNode, False) # Minimum Cost child node is further explored with
backtracking status as 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= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

```
h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}  # Heuristic values of
Nodes
graph2 = {                                    # Graph of Nodes and Edges
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],      # Neighbors of Node 'A', B, C & D with
repective weights
    'B': [[('G', 1)], [('H', 1)]],               # Neighbors are included in a list of lists
    'D': [[('E', 1), ('F', 1)]]                # Eachsublist indicate a "OR" node or "AND"
nodes
}

G2 = Graph(graph2, h2, 'A')                      # Instantiate Graph object with graph,
heuristic values and start Node
G2.applyAOStar()                               # Run the AO* algorithm
G2.printSolution()                            # Print the solution graph as output of the
AO* algorithm search
```

AOStar(Recursive).py
Open with
Displaying AOStar(Recursive).py.

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
-----------------------------------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START
NODE: A
------------------------------------------------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
------------------------------------------------------------
HEURISTIC VALUES  : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : E
```

```
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : F
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': [], 'F': []}
PROCESSING NODE   : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
------------------------------------------------------------
```

Program 3

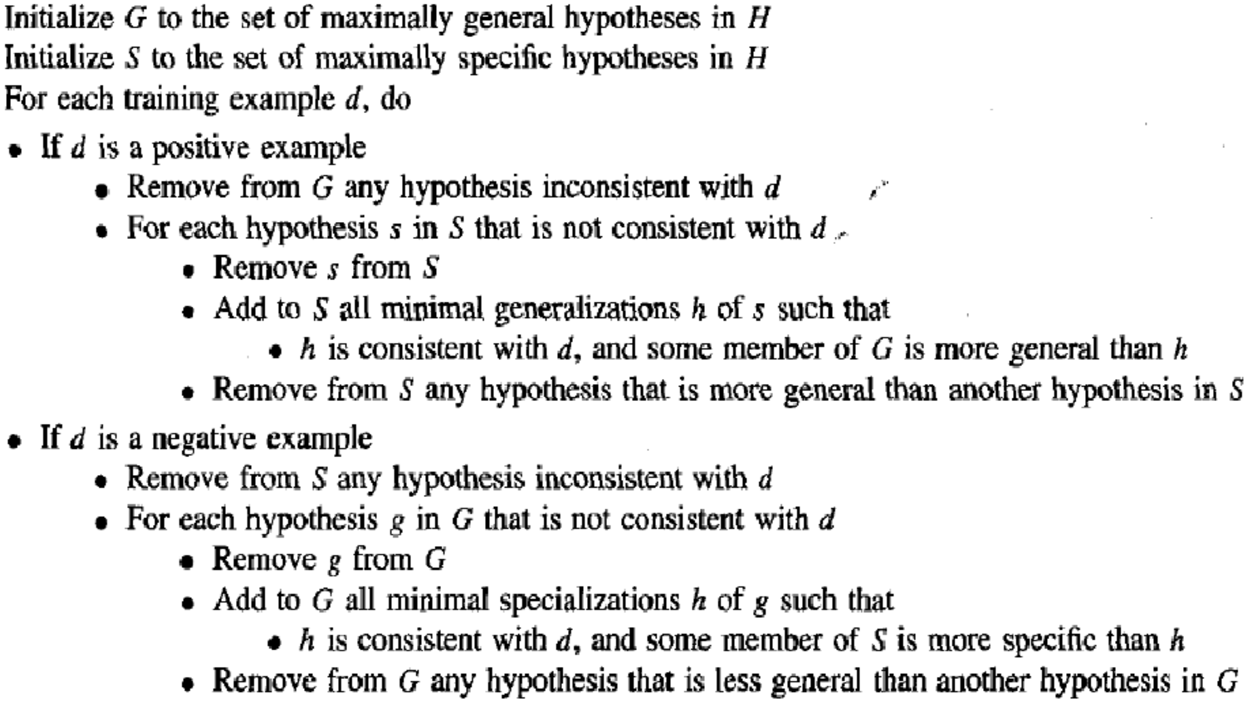
Foragivensetoftrainingdataexamplesstoredina. CSVfile,implement an demonstrate the Candidate Eliminationalgorithmtooutputadescriptionofthese tofallhypothesesconsistent with the trainingexamples.

Task: The CANDIDATE-ELIMINATION algorithm computes the version space containingall hypotheses from H that are consistent with an observed sequence of trainingexamples.

Dataset: Enjoy Sports Training Examples:

Example	**Sky**	**AirTemp**	**Humidity**	**Wind**	**Water**	**Forecast**	**EnjoySport**
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Candidate Elimination Algorithm:



```
Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do
• If d is a positive example
    • Remove from G any hypothesis inconsistent with d
    • For each hypothesis s in S that is not consistent with d
        • Remove s from S
        • Add to S all minimal generalizations h of s such that
            • h is consistent with d, and some member of G is more general than h
        • Remove from S any hypothesis that is more general than another hypothesis in S
• If d is a negative example
    • Remove from S any hypothesis inconsistent with d
    • For each hypothesis g in G that is not consistent with d
        • Remove g from G
        • Add to G all minimal specializations h of g such that
            • h is consistent with d, and some member of S is more specific than h
        • Remove from G any hypothesis that is less general than another hypothesis in G
```

Candidate Elimination Program:

```
importnumpy as np
import pandas as pd

data = pd.DataFrame(data=pd.read_csv('set.csv'))
concepts = np.array(data.iloc[:,0:-1])
print("\n Instances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\n Target values are:", target)

def learn(concepts, target):
specific_h = concepts[0].copy()
print("\n Initilization of specific_h and generic_h")
print("\n Specific Boundary", specific_h)
general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
print("\n Generic Boundary ",general_h)

fori, h in enumerate(concepts):
print("\n Instance", 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 Boundary 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 == ['?', '?', '?', '?', '?', '?']]
fori in indices:
general_h.remove(['?', '?', '?', '?', '?', '?'])
returnspecific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final S:", s_final, sep="\n")
print("Final G:", g_final, sep="\n")
```

out put

```
Instances are:
 [['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
 Target values are: ['Y' 'N' 'Y']

Initilization of specific_h and generic_h

 Specific Boundary ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

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

 Instance 1 is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Specific Boundary after  1 Instance is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Generic Boundary after  1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



 Instance 2 is ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
Specific Boundary after  2 Instance is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Generic Boundary after  2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Instance 3 is ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']
Specific Boundary after  3 Instance is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Generic Boundary after  3 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final S:
['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Final G:
[]
```

Program 4

Writeaprogramtodemonstratetheworkingofthedecisiontreebased ID3algorithm.Usean appropriatedatasetforbuildingthedecisiontreeandapplythisknowledgetoclassifyanew sample.

Task: ID3 determines the information gain for each candidate attribute, then Selects the one with highest information gain as the root node of the tree. The information gain values for all four attributes are calculated using the following formula:

$$\text{Entropy}(S)=\sum -P(I).\log_2 P(I)$$

$$\text{Gain}(S,A)=\text{Entropy}(S)-\sum[P(S/A).\text{Entropy}(S/A)]$$

Dataset: pima-indians-diabetes.csv

ID3Algorithm:

ID3(*Examples, Target_attribute,Attributes*)

Examples are the training examples. Target_attribute isthe attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifiesthe givenExamples.

```
o Create a Root node for thetree
o If all Examples are positive, Return the single-node tree Root, with
  label =+
o If all Examples are negative, Return the single-node tree Root, with
  label =-
o If Attributes is empty, Return the single-node tree Root, with label
  =most common value of Target_attributeinExamples
  o Otherwise Begin
❖ A ← the attribute from Attributes that best* classifiesExamples
❖ The decision attribute for Root←A
❖ For each possible value, υi, ofA,
  • Add a new tree branch below Root, corresponding to the test A =υi
  • Let Examplesif,be the subset of Examples that have value υiforA
  • If Examplesυi, is empty
      ❖ Then below this new branch add a leaf node with label=most
        common value of Target attributeinExamples
      ❖ Else below this new branch add thesubtree
            ID3(Examplesυi,Target_attribute,Attributes–{A}))
          End
          Return Root
```

ID3 Program:

```
import pandas as pd
importnumpy as np
#Import the dataset and define the feature as well as the target datasets /
columns#
dataset = pd.read_csv('playtennis.csv',
names=['outlook','temperature','humidity','wind','class'])
#Import all columns omitting the fist which consists the names of the animals
#We drop the animal names since this is not a good feature to split the data on

attributes =('Outlook','Temperature','Humidity','Wind','PlayTennis')
def entropy(target_col):
    """
    Calculate the entropy of a dataset.
    The only parameter of this function is the target_col parameter which specifies
the target column
    """
elements,counts = np.unique(target_col,return_counts = True)
    #print(elements,counts)
    #print(counts)
entropy = np.sum([(-
counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in
range(len(elements))])
    #print('Entropy =', entropy)
return entropy

defInfoGain(data,split_attribute_name,target_name="class"):
    #Calculate the entropy of the total dataset
total_entropy = entropy(data[target_name])

    ##Calculate the entropy of the dataset

    #Calculate the values and the corresponding counts for the split attribute
vals,counts= np.unique(data[split_attribute_name],return_counts=True)

    #Calculate the weighted entropy
Weighted_Entropy =
np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_nam
e]==vals[i]).dropna()[target_name]) for i in range(len(vals))])

    #Calculate the information gain
```

```
Information_Gain = total_entropy - Weighted_Entropy
returnInformation_Gain



def
ID3(data,originaldata,features,target_attribute_name="class",parent_node_class
= None):    #Define the stopping criteria --> If one of this is satisfied, we want to
return a leaf node#

   #If all target_values have the same value, return this value

iflen(np.unique(data[target_attribute_name])) <= 1:
returnnp.unique(data[target_attribute_name])[0]

   #If the dataset is empty, return the mode target feature value in the original
dataset
eliflen(data)==0:
return
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldat
a[target_attribute_name],return_counts=True)[1])]

eliflen(features) ==0:
returnparent_node_class

   #If none of the above holds true, grow the tree!

else:
      #Set the default value for this node --> The mode target feature value of the
current node
parent_node_class =
np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribut
e_name],return_counts=True)[1])]

      #Select the feature which best splits the dataset
item_values = [InfoGain(data,feature,target_attribute_name) for feature in
features] #Return the information gain values for the features in the dataset
best_feature_index = np.argmax(item_values)
best_feature = features[best_feature_index]

        #Create the tree structure. The root gets the name of the feature
(best_feature) with the maximum information
        #gain in the first run
tree = {best_feature:{}}



        #Remove the feature with the best inforamtion gain from the feature space
features = [i for i in features if i != best_feature]

        #Grow a branch under the root node for each possible value of the root node
feature

for value in np.unique(data[best_feature]):
value = value
            #Split the dataset along the value of the feature with the largest
information gain and therwith create sub_datasets
sub_data = data.where(data[best_feature] == value).dropna()

            #Call the ID3 algorithm for each of those sub_datasets with the new
parameters --> Here the recursion comes in!
subtree =
ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)

            #Add the sub tree, grown from the sub_dataset to the tree under the root
node
tree[best_feature][value] = subtree

return(tree)

def predict(query,tree,default = 1):
    #1.
for key in list(query.keys()):
if key in list(tree.keys()):
            #2.
try:
result = tree[key][query[key]]
except:
return default

            #3.
result = tree[key][query[key]]
            #4.
ifisinstance(result,dict):
return predict(query,result)
else:
return result

deftrain_test_split(dataset):
training_data = dataset.iloc[:14].reset_index(drop=True)
    #We drop the index respectively relabel the index
    #starting form 0, because we do not want to run into errors regarding the row
labels / indexes
    #testing_data = dataset.iloc[10:].reset_index(drop=True)
returntraining_data  #,testing_data

def test(data,tree):
    #Create new query instances by simply removing the target feature column
from the original dataset and
    #convert it to a dictionary
queries = data.iloc[:,:-1].to_dict(orient = "records")

    #Create a empty DataFrame in whose columns the prediction of the tree are
stored
predicted = pd.DataFrame(columns=["predicted"])
```

```
    #Calculate the prediction accuracy
  fori in range(len(data)):
  predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)

  print('The prediction accuracy is: ',(np.sum(predicted["predicted"] ==
  data["class"])/len(data))*100,'%')

  """
  Train the tree, Print the tree and predict the accuracy
  """
  XX = train_test_split(dataset)
  training_data=XX
  #testing_data=XX[1]
  tree = ID3(training_data,training_data,training_data.columns[:-1])
  print(' Display Tree',tree)
  print('len=',len(training_data))
  test(training_data,tree)
```

Output:

```
Display Tree {'outlook': {'Overcast': 'Yes', 'Rainy': {'wind': {'Strong': 'No',
'Weak': 'Yes'}}, 'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}}}
Len= 14
The prediction accuracy is:  100.0 %
```

Program5:

Build an Artificial Neural Network by implementing the Back propagation algorithmand test the same using appropriate dataset

The stochastic gradient descent version of the BACKPROPAGATION algorithm for feed forward networks containing two layers of sigmoid units.

Step 1: Begins by constructing a network with the desired number of hidden and output units and initializing all network weights to small random values. . For each training example, it applies the network to the example, calculates the error of the network output for this example, computes the gradient with respect to the error on this example, then updates all weights in the network. This gradient descent step is iterated (often thousands of times, using the same training examples multiple times) until the network performs acceptably well.

Step 2: The gradient descent weight-update rule is similar to the delta training rule The only difference is that the error (t - o) in the delta rule is replaced by a more complex error term aj.

Step 3: updates weights incrementally, following the Presentation of each training example. This corresponds to a stochastic approximation to gradient descent. To obtain the true gradient of E one would sum the Sj, xji values over all training examples before altering weight values.

Step 4: The weight-update loop in BACKPROPAGATION may be iterated thousands of times in a typical application. A variety of termination conditions can be used to halt the procedure.

One may choose to halt after a fixed number of iterations through the loop, or once the error on the training examples falls below some threshold.

Dataset:

ANN Program:

```
importnumpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
```

```
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100

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

#Derivative of Sigmoid Function
defderivatives_sigmoid(x):
return x * (1 - x)

#Variable initialization
epoch=5000          #Setting training iterations
lr=0.1              #Setting learning rate
inputlayer_neurons = 2          #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1              #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))


#draws a random range of numbers uniformly of dim x*y
fori in range(epoch):

#Forward Propogation
    hinp1=np.dot(X,wh)
hinp=hinp1 + bh
hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
outinp= outinp1+ bout
output = sigmoid(outinp)
```

```
#Backpropagation
    EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO* outgrad
    EH = d_output.dot(wout.T)

#how much hidden layer wts contributed to error
hiddengrad = derivatives_sigmoid(hlayer_act)
d_hiddenlayer = EH * hiddengrad

# dotproduct of nextlayererror and currentlayerop
wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hiddenlayer) *lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output

```
runfile('C:/Users/CSE/Desktop/meena/AI _ ML Lab Programs/PGM5/5-ann.py',
wdir='C:/Users/CSE/Desktop/meena/AI _ ML Lab Programs/PGM5')

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.85572459]
 [0.83716918]
 [0.86030558]]
```

Program 6

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.

Task:ItisaclassificationtechniquebasedonBayes’Theoremwithanassumptionofindependenceamongpredictors.Insimpleterms,aNaiveBayesclassifierassumesthatthepresenceofaparticularfeatureinaclassisunrelatedtothepresenceofanyotherfeature.Forexample,afruitmaybeconsideredtobeanappleifitisred,round,andabout3inchesindiameter.Evenifthesefeaturesdependoneachotheroruponthe existenceoftheotherfeatures,allofthesepropertiesindependently contribute to the probability that this fruit is an apple and that is why it is known as‘Naive’.

Dataset : Pima-indians-diabetes.csv

It is a classification technique based on Bayes‟ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as „Naive‟.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

1) Handling Of Data:

- Load the data from the CSV file and split in to training and test data set.
- Training data set can be used to by Naïve Bayes to make predictions.
- And Test data set can be used to evaluate the accuracy of the model.

2) Summarize Data:

The summary of the training data collected involves the mean and the standard deviation for each attribute, by class value.

- These are required when making predictions to calculate the probability of specific attribute values belonging to each class value.
- Summary data can be break down into the following sub-tasks:

- **Separate Data By Class**: The first task is to separate the training dataset instances by class value so that we can calculate statistics for each class. We can do that by creating a map of each class value to a list of instances that belong to that class and sort the entire dataset of instances into the appropriate lists.
- **Calculate Mean**:We need to calculate the mean of each attribute for a class value. The mean is the central middle or central tendency of the data, and we will use it as the middle of our gaussian distribution when calculating probabilities.
- **Calculate Standard Deviation**: We also need to calculate the standard deviation of each attribute for a class value. The standard deviation describes the variation of spread of the data, and we will use it to characterize the expected spread of each attribute in our Gaussian distribution when calculating probabilities.
- **Summarize Dataset**: For a given list of instances (for a class value) we can calculate the mean and the standard deviation for each attribute.
- The zip function groups the values for each attribute across our data instances into their own lists so that we can compute the mean and standard deviation values for the attribute.
- **Summarize Attributes By Class**: We can pull it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

3) Make Predictions:

- ❖ Making predictions involves calculating the probability that a given data instance belongs to each class,
- ❖ then selecting the class with the largest probability as the prediction.
- ❖ Finally, estimation of the accuracy of the model by making predictions for each data instance in the test dataset.

4) Evaluate Accuracy: The predictions can be compared to the class values in the test dataset and a classification\ accuracy can be calculated as an accuracy ratio between 0& and 100%.

Naïve Bayes Program:

```
print("\nNaive Bayes Classifier for concept learning problem")
import csv
import random
import math
import operator

defsafe_div(x,y):
if y == 0:
return 0
return x/y

# 1.Data Handling
# 1.1 Loading the Data from csv file of ConceptLearning dataset.
defloadCsv(filename):
lines = csv.reader(open(filename))
dataset = list(lines)
fori in range(len(dataset)):
dataset[i] = [float(x) for x in dataset[i]]
return dataset

#1.2 Splitting the Data set into Training Set
defsplitDataset(dataset, splitRatio):
trainSize = int(len(dataset) * splitRatio)
trainSet = []
copy = list(dataset)
i=0
whilelen(trainSet) <trainSize:
  #index = random.randrange(len(copy))
trainSet.append(copy.pop(i))
return [trainSet, copy]

#2.Summarize Data
#The naive bayes model is comprised of a
```

```
#summary of the data in the training dataset.
#This summary is then used when making predictions.
#involves the mean and the standard deviation for each attribute, by class value

#2.1: Separate Data By Class
#Function to categorize the dataset in terms of classes
#The function assumes that the last attribute (-1) is the class value.
#The function returns a map of class values to lists of data instances.

defseparateByClass(dataset):
separated = {}
fori in range(len(dataset)):
vector = dataset[i]
if (vector[-1] not in separated):
separated[vector[-1]] = []
separated[vector[-1]].append(vector)
return separated

#The mean is the central middle or central tendency of the data,
# and we will use it as the middle of our gaussian distribution
# when calculating probabilities

#2.2 : Calculate Mean
def mean(numbers):
returnsafe_div(sum(numbers),float(len(numbers)))

#The standard deviation describes the variation of spread of the data,
#and we will use it to characterize the expected spread of each attribute
#in our Gaussian distribution when calculating probabilities.

#2.3 : Calculate Standard Deviation
defstdev(numbers):
avg = mean(numbers)
variance = safe_div(sum([pow(x-avg,2) for x in numbers]),float(len(numbers)-1))
returnmath.sqrt(variance)
```

```
#2.4 : Summarize Dataset
#Summarize Data Set for a list of instances (for a class value)
#The zip function groups the values for each attribute across our data instances
#into their own lists so that we can compute the mean and standard deviation
values
#for the attribute.

def summarize(dataset):
summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
del summaries[-1]
return summaries

#2.5 : Summarize Attributes By Class
#We can pull it all together by first separating our training dataset into
#instances grouped by class.Then calculate the summaries for each attribute.

defsummarizeByClass(dataset):
separated = separateByClass(dataset)
summaries = {}
forclassValue, instances in separated.items():
summaries[classValue] = summarize(instances)
print("Summarize Attributes By Class")
print(summaries)
print(" ")
return summaries

#3.Make Prediction
#3.1 Calculate Probaility Density Function
defcalculateProbability(x, mean, stdev):
exponent = math.exp(-safe_div(math.pow(x-mean,2),(2*math.pow(stdev,2))))
final = safe_div(1 , (math.sqrt(2*math.pi) * stdev)) * exponent
return final

#3.2 Calculate Class Probabilities
defcalculateClassProbabilities(summaries, inputVector):
probabilities = {}
forclassValue, classSummaries in summaries.items():
```

```
probabilities[classValue] = 1
fori in range(len(classSummaries)):
mean, stdev = classSummaries[i]
    x = inputVector[i]
probabilities[classValue] *= calculateProbability(x, mean, stdev)
return probabilities

#3.3 Prediction : look for the largest probability and return the associated class
def predict(summaries, inputVector):
probabilities = calculateClassProbabilities(summaries, inputVector)
bestLabel, bestProb = None, -1
forclassValue, probability in probabilities.items():
ifbestLabel is None or probability >bestProb:
bestProb = probability
bestLabel = classValue
returnbestLabel

#4.Make Predictions
# Function which return predictions for list of predictions
# For each instance
defgetPredictions(summaries, testSet):
predictions = []
fori in range(len(testSet)):
result = predict(summaries, testSet[i])
predictions.append(result)
return predictions

#5. Computing Accuracy
defgetAccuracy(testSet, predictions):
correct = 0
fori in range(len(testSet)):
iftestSet[i][-1] == predictions[i]:
correct += 1
accuracy = safe_div(correct,float(len(testSet))) * 100.0
return accuracy

def main():
    filename = 'ConceptLearning.csv'
    splitRatio = 0.9
    dataset = loadCsv(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('Split {0} rows into'.format(len(dataset)))
    print('Number of Training data: ' + (repr(len(trainingSet))))
    print('Number of Test Data: ' + (repr(len(testSet))))
    print("\nThe values assumed for the concept learning attributes are\n")
    print("OUTLOOK=> Sunny=1 Overcast=2 Rain=3\nTEMPERATURE=> Hot=1 Mild=2 Cool=3\nHUMIDITY=> High=1 Normal=2\nWIND=> Weak=1 Strong=2")
    print("TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5")
    print("\nThe Training set are:")
    for x in trainingSet:
    print(x)
    print("\nThe Test data set are:")
    for x in testSet:
    print(x)
    print("\n")

    # prepare model
    summaries = summarizeByClass(trainingSet)

    # test model
    predictions = getPredictions(summaries, testSet)
    actual = []
    fori in range(len(testSet)):
    vector = testSet[i]
    actual.append(vector[-1])
    # Since there are five attribute values, each attribute constitutes to 20% accuracy. So if all attributes
    #match with predictions then 100% accuracy
    print('Actual values: {0}%'.format(actual))
    print('Predictions: {0}%'.format(predictions))
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: {0}%'.format(accuracy))
main()
```

OUTPUT:

```
runfile('C:/Users/CSE/Desktop/meena/AI _ ML Lab
Programs/PGM6/PGM6.py', wdir='C:/Users/CSE/Desktop/meena/AI _ ML Lab
Programs/PGM6')

Naive Bayes Classifier for concept learning problem
Split 14 rows into
Number of Training data: 12
Number of Test Data: 2

The values assumed for the concept learning attributes are
OUTLOOK=> Sunny=1 Overcast=2 Rain=3
TEMPERATURE=> Hot=1 Mild=2 Cool=3
HUMIDITY=> High=1 Normal=2
WIND=> Weak=1 Strong=2
TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5

The Training set are:
[1.0, 1.0, 1.0, 1.0, 5.0]

The Test data set are:
[1.0, 1.0, 1.0, 2.0, 5.0]

The Test data set are:
[2.0, 1.0, 1.0, 1.0, 10.0]

The Test data set are:
[3.0, 2.0, 1.0, 1.0, 10.0]

The Test data set are:
[3.0, 3.0, 2.0, 1.0, 10.0]

The Test data set are:
[3.0, 3.0, 2.0, 2.0, 5.0]

The Test data set are:
```

```
[2.0, 3.0, 2.0, 2.0, 10.0]

The Test data set are:
[1.0, 2.0, 1.0, 1.0, 5.0]

The Test data set are:
[1.0, 3.0, 2.0, 1.0, 10.0]

The Test data set are:
[3.0, 2.0, 2.0, 1.0, 10.0]

The Test data set are:
[1.0, 2.0, 2.0, 2.0, 10.0]

The Test data set are:
[2.0, 2.0, 1.0, 2.0, 10.0]

The Test data set are:
[2.0, 1.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 2.0, 5.0]

Summarize Attributes By Class
{5.0: [(1.5, 1.0), (1.75, 0.9574271077563381), (1.25, 0.5), (1.5,
0.5773502691896257)], 10.0: [(2.125, 0.8345229603962802), (2.25,
0.7071067811865476), (1.625, 0.5175491695067657), (1.375,
0.5175491695067657)]}

Actual values: [5.0]%
Predictions: [5.0, 5.0]%
Accuracy: 50.0%
```

Program7

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the samedata set for clustering using k-Means algorithm. Compare the results of these twoalgorithms and comment on the quality of clustering. You can add Java/Python MLlibrary classes/API in the program.

Introduction to Expectation-Maximization(EM)

The EM algorithm tends to get stuck less than K-means algorithm. The idea is to assign datapoints partially to different clusters instead of assigning to only one cluster. To do this partialassignment, we model each cluster using a probabilistic distribution So a data point associateswith a cluster with certain probability and it belongs to the cluster with the highest probabilityin the final assignment.

Expectation-Maximization (EM)algorithm

Step 1: An initial guess is made for the model’s parameters and a probability Distribution is created. This is sometimes called the “E-Step” for the “Expected” distribution.

Step 2: Newly observed data is fed into the model.

Step 3: The probability distribution from the E-step is drawn to include the new data. This is sometimes called the “M-step.”

Step 4: Steps 2 through 4 are repeated until stability.
Dataset:

EMalgorithmPrograms:
#Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set
#for clustering using k-Means algorithm. Compare the results of these two algorithms and
#comment on the quality of clustering. You can add Java/Python ML library classes/API in

```
importmatplotlib.pyplot as plt
fromsklearn import datasets
fromsklearn.cluster import KMeans
importsklearn.metrics as sm
import pandas as pd
importnumpy as np


l1 = [0,1,2]
def rename(s):
    l2 = []
    fori in s:
        ifi not in l2:
            l2.append(i)

    fori in range(len(s)):
        pos = l2.index(s[i])
        s[i] = l1[pos]

    return s

# import some data to play with
iris = datasets.load_iris()

print("\n IRIS DATA :",iris.data);
print("\n IRIS FEATURES :\n",iris.feature_names)
print("\n IRIS TARGET  :\n",iris.target)
print("\n IRIS TARGET NAMES:\n",iris.target_names)


# Store the inputs as a Pandas Dataframe and set the column names
X = pd.DataFrame(iris.data)

#print(X)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

#print(X.columns) #print("X:",x)
```

```
#print("Y:",y)
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Set the size of the plot
plt.figure(figsize=(14,7))

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

# Plot Sepal
plt.subplot(1,2,1)
plt.scatter(X.Sepal_Length,X.Sepal_Width, c=colormap[y.Targets], s=40)
plt.title('Sepal')

plt.subplot(1,2,2)
plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Petal')
plt.show()

print("Actual Target is:\n", iris.target)

# K Means Cluster
model = KMeans(n_clusters=3)
model.fit(X)

# Set the size of the plot
plt.figure(figsize=(14,7))

# Create a colormap
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')
```

```
# 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.show()

km = rename(model.labels_)
print("\nWhatKMeans thought: \n", km)
print("Accuracy of KMeans is ",sm.accuracy_score(y, km))
print("Confusion Matrix for KMeans is \n",sm.confusion_matrix(y, km))

#The GaussianMixturescikit-learn class can be used to model this problem
#and estimate the parameters of the distributions using the expectation-
maximization algorithm.

fromsklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
print("\n",xs.sample(5))

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

y_cluster_gmm = gmm.predict(xs)

plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_cluster_gmm], s=40)
plt.title('GMM Classification')
plt.show()

em = rename(y_cluster_gmm)
print("\nWhat EM thought: \n", em)
print("Accuracy of EM is ",sm.accuracy_score(y, em))
print("Confusion Matrix for EM is \n", sm.confusion_matrix(y, em))
```

Output

```
n [1]: runfile('C:/Users/CSE/Desktop/meena/AI _ ML Lab Programs/PGM7/PGM7.py', wdir='C:/Users/CSE/Desktop/meena/AI _ ML Lab Programs/PGM7')

 IRIS DATA : [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [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]
```

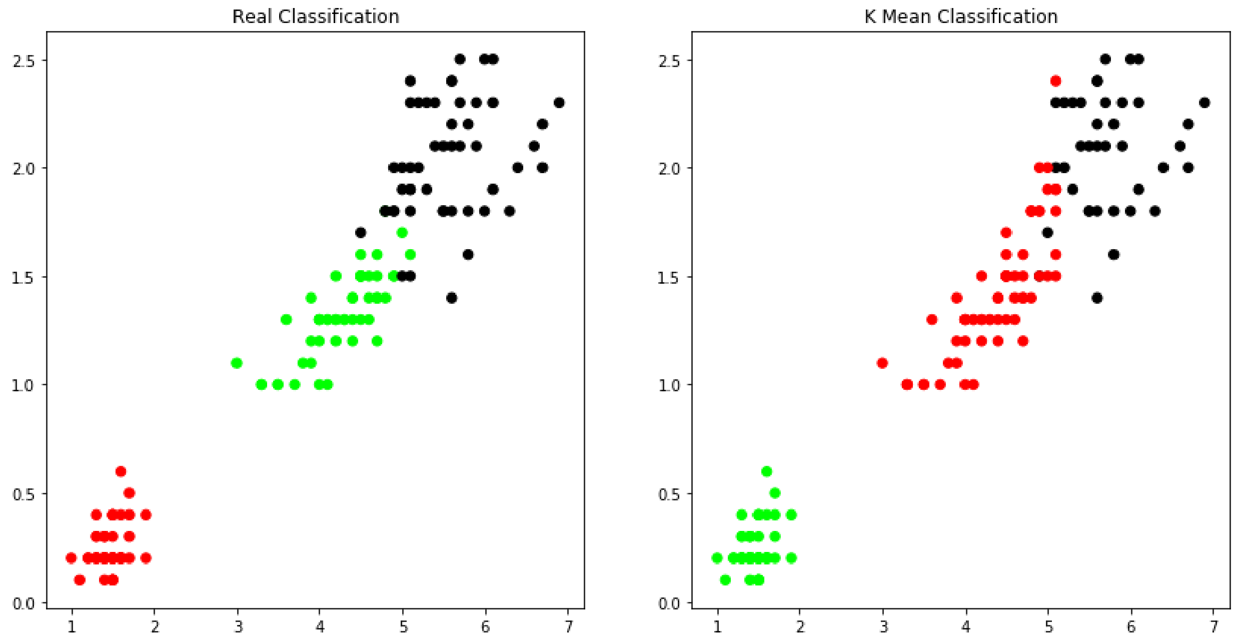
```
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.1 1.5 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
```

```
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
```

```
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
```

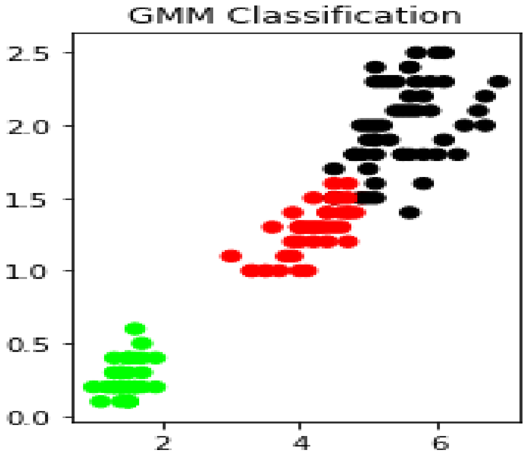
```
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]

 IRIS FEATURES :
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

 IRIS TARGET  :
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

 IRIS TARGET NAMES:
['setosa' 'versicolor' 'virginica']
```

```
Actual Target is:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```



```
What KMeansthought:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
Accuracy of KMeansis  0.8933333333333333
Confusion Matrix for KMeans is
 [[50  0  0]
[ 0 48  2]
[ 0 14 36]]
```

```
Sepal_LengthSepal_WidthPetal_LengthPetal_Width
59      -0.779513    -0.819166     0.080370     0.264699
16      -0.537178     1.957669    -1.398138    -1.050031
6      -1.506521     0.800654    -1.341272    -1.181504
23      -0.900681     0.569251    -1.170675    -0.918558
101     -0.052506    -0.819166     0.762759     0.922064
```



```
What EM thought:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1
 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Accuracy of EM is  0.9666666666666667
Confusion Matrix for EM is
 [[50  0  0]
[ 0 45  5]
[ 0  0 50]]
```

Program8

Writeaprogramtoimplementk-NearestNeighboralgorithmtoclassifytheirisdataset.Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

TASK: The task of this program is to classify the IRIS data set examples by using thek-Nearest Neighbour algorithm. The new instance has to be classified based on its k nearestneighbors.

Dataset: iris.csv

ALGORITHM

Let m be the number of training data samples. Let p be an unknownpoint.

1. Store the training samples in an array of data points arr[]. This means each element of thisarray represents a tuple (x,y).
2. for i=0 tom:
   Calculate Euclidean distance d(arr[i],p).
3. MakesetSofKsmallestdistancesobtained.Eachofthesedistancescorrespondtoanalready classified datapoint.
4. Return the majority label amongS.

KNN Program

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jul  3 17:58:01 2018

@author: Sharmila
"""
#1.Import Data

fromsklearn.datasets import load_iris
iris = load_iris()
print("Feature Names:",iris.feature_names,"IrisData:",iris.data,"Target
Names:",iris.target_names,"Target:",iris.target)

#2. Split the data into Test and Data
fromsklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
iris.data, iris.target, test_size = .25)

#neighbors_settings = range(1, 11)
#for n_neighbors in neighbors_settings:

#3.Build The Model
fromsklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)

#4.Calculate Accuracy of the Test  data with the trained data
print(" Accuracy=",clf.score(X_test, y_test))



#5 Calculate the prediction with the labels of the test data
print("Predicted Data")
print(clf.predict(X_test))

prediction=clf.predict(X_test)

print("Test data :")
print(y_test)

#6 To identify the miss classification
diff=prediction-y_test
print("Result is ")
print(diff)
print('Total no of samples misclassied =', sum(abs(diff)))
```

Output:

```
runfile('C:/Users/CSE/Desktop/meena/18CSL76 Lab Programs/P9 KNN/P9_KNN.py', wdir='C:/Users/CSE/Desktop/meena/18CSL76 Lab Programs/P9 KNN')
```

```
Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)'] Iris Data: [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [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 0.1]
 [5.  3.2 1.2 0.2]
```

```
[5.5 3.5 1.3 0.2]
[4.9 3.1 1.5 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
```

```
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
```

```
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
[6.7 3.  5.2 2.3]
[6.3 2.5 5.  1.9]
```

```
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]] Target Names: ['setosa' 'versicolor' 'virginica'] Target: [0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
 Accuracy= 0.9473684210526315
Predicted Data
[0 2 0 0 2 1 1 0 1 2 2 0 2 0 0 0 1 2 0 1 1 1 0 0 0 2 0 2 2 2 2 2 1 2 0 2 1
 1]
Test data :
[0 2 0 0 2 1 1 0 1 2 1 0 2 0 0 0 1 2 0 1 1 1 0 0 0 2 0 2 1 2 2 2 1 2 0 2 1
 1]
Result is
[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 0]
Total no of samples misclassied = 2
Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)'] Iris Data: [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
```

```
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[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 0.1]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.1 1.5 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
```

```
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
```

```
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
```

```
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]] Target Names: ['setosa' 'versicolor' 'virginica'] Target: [0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
 Accuracy= 0.9736842105263158
Predicted Data
[2 2 0 2 1 2 1 0 2 0 0 1 0 0 2 1 2 0 1 0 0 0 1 2 2 0 0 2 2 1 1 0 1 2 1 2 0
 0]
Test data :
[2 2 0 2 1 2 1 0 1 0 0 1 0 0 2 1 2 0 1 0 0 0 1 2 2 0 0 2 2 1 1 0 1 2 1 2 0
 0]
Result is
[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0]
```

Total no of samples misclassied = 1

Program9

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

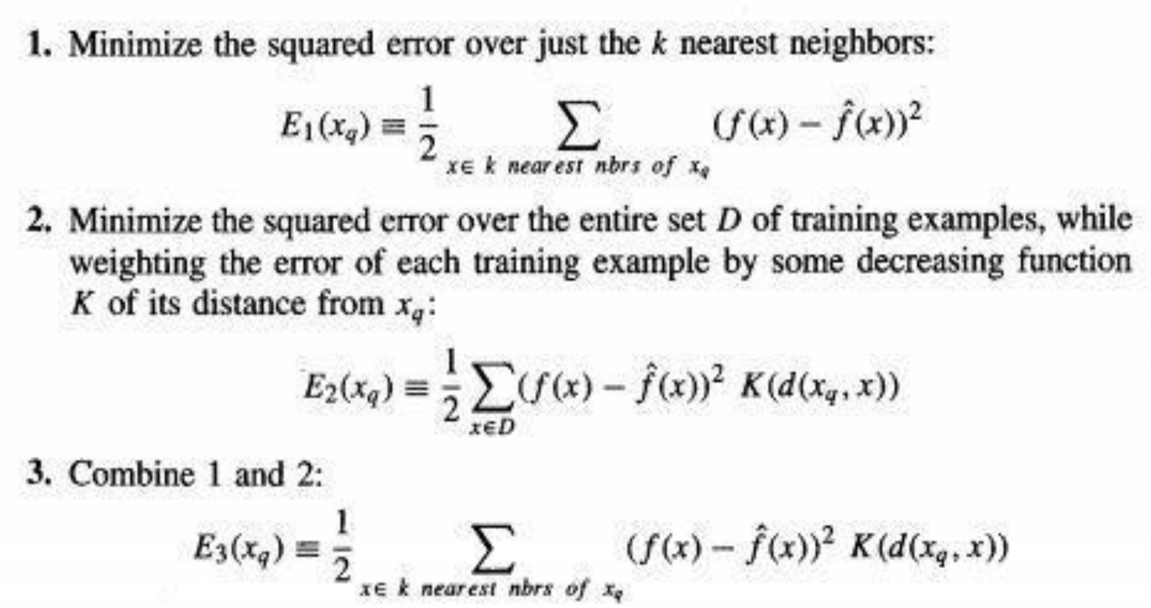
Locally Weighted Regression–

1. Nonparametric regression is a category of regression analysis in whichthe predictordoesnottakeapredeterminedformbutisconstructedaccordingto information derived from the data(trainingexamples).
2. Nonparametric regression requires larger sample sizes than regression basedon parametric models. Because larger the data available ,accuracy will behigh.

Locally Weighted Linear Regression–

- Locally weighted regression is called local because the function isapproximated basedaonlyondatanearthequerypoint,weightedbecausethecontributionof each training example is weighted by its distance from the querypoint.
- Query point is nothing but the point nearer to the target function , which willhelp in finding the actual position of the targetfunction.

Letusconsiderthecaseoflocallyweightedregressioninwhichthetargetfunctionfis approximated near x, using a linear function of theform



1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in \, k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

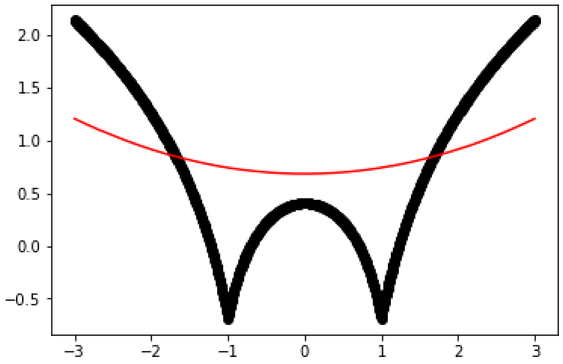
3. Combine 1 and 2:

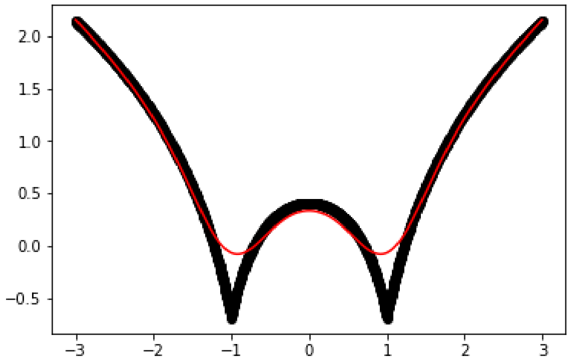
$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \, k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

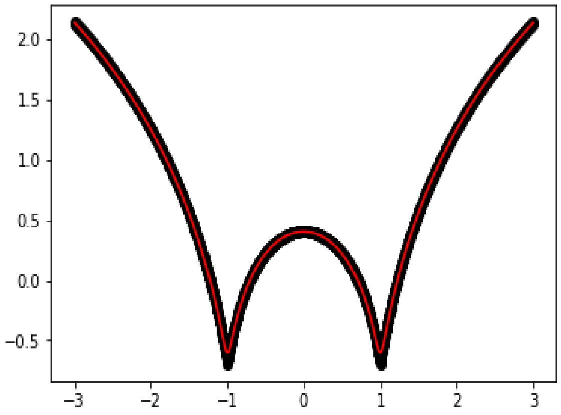
program

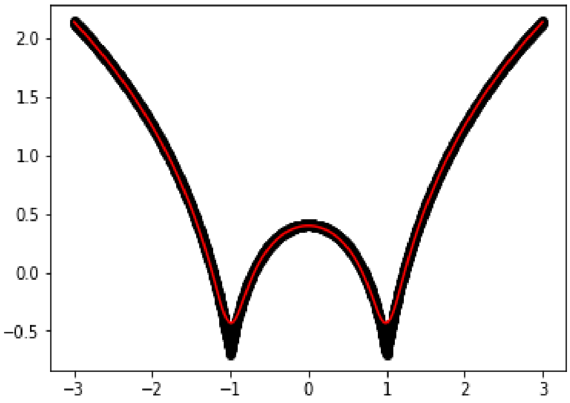
```
importnumpy as np
importmatplotlib.pyplot as plt

deflocal_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)
```

Output


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


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


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


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

\