AI & ML VTU 7TH SEM LAB PROGRAMS

CS & IS

ALRIGHT MENTEE

PROGRAM 1

Implement A* Search Algorithm

A* Search Algorithm is a Path Finding Algorithm. It is similar to Breadth First Search(BFS). It will search shortest path using heuristic value assigned to node and actual cost from Source_node to Dest_node

Real-life Examples

- Maps
- Games

Formula for A* Algorithm

h(n) = heuristic_value
g(n) = actual_cost
f(n) = actual_cost + heursitic_value

f(n) = g(n) + h(n)

PROGRAM

```
def aStarAlgo(start_node, stop_node):


    open_set = set(start_node) # {A}, len{open_set}=1
    closed_set = set()
    g = {} # store the distance from starting node
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node # parents['A']='A"

    while len(open_set) > 0 :
        n = None

        for v in open_set: # v='B'/'F'
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v # n='A'

        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)      # m=B weight=6 {'F','B','A'} len{open_set}=2
                    parents[m] = n       # parents={'A':A,'B':A} len{parent}=2
                    g[m] = g[n] + weight # g={'A':0,'B':6, 'F':3} len{g}=2


            #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)# {'F','B'} len=2
        closed_set.add(n) #{A} len=1

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

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_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
    }

    return H_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']
['A', 'F', 'G', 'I', 'J']
```

PROGRAM 2

Implement AO* Algorithm

AO* Search Algorithm is a Path Finding Algorithm and it is similar to A* star, other than AND is used between two nodes along with OR. After getting shortest path it will return back to root node and it will update it's heuristic value. It is similar to Depth First Search(DFS). It will search shortest path using heuristic value assigned to node and actual cost from Source_node to Dest_node

What is difference between A * and AO * algorithm?
An A* algorithm represents an OR graph algorithm that is used to find a single solution (either this or that). An AO* algorithm represents an AND-OR graph algorithm that is used to find more than one solution by ANDing more than one branch.

Real-life Examples

- Maps
- Games

Formula for AO* Algorithm

h(n) = heuristic_value

g(n) = actual_cost
f(n) = actual_cost + heursitic_value

f(n) = g(n) + h(n)

PROGRAM

```
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={}

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

    def getNeighbors(self, v): # gets the Neighbors of a given node
        return self.graph.get(v,'')

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

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

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

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


    def printSolution(self):
        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE:",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
          if minimumCost>cost:
            minimumCost=cost
            costToChildNodeListDict[minimumCost]=nodeList # set the Minimum Cost child node/s


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


    def aoStar(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("-----------------------------------------------------------------------------------------")

      if self.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

        for childNode in childNodeList:
          self.parent[childNode]=v
          if self.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

        if backTracking==False: # check the current call is not for backtracking
          for childNode 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)]] # Each sublist 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 AO* Algorithm search
```

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 STARTNODE: 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 STARTNODE: A
------------------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
------------------------------------------------------------
```

PROGRAM 3

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.

PROGRAM

```
import csv

with open("trainingexamples.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)

    specific = data[1][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"

        print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")
        print(specific)
        print(general)

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
            if j != '?':
```

```
            gh.append(i)
            break
    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```

OUTPUT

Step 1 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 4 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Step 5 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', '?', '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific hypothesis:
['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General hypothesis:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

DATA SET

SKY	**AIRTEMP**	**HUMIDITY**	**WIND**	**WATER**	**FORCAST**	**ENJOYSPORT**
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

PROGRAM 4

ID3 Algorithm

PROGRAM

```
import pandas as pd
import math

# function to calculate the entropy of entire dataset
# -----------------------------------------------------------------------
def base_entropy(dataset):
   p = 0
   n = 0
   target = dataset.iloc[:, -1]
   targets = list(set(target))
   for i in target:
      if i == targets[0]:
         p = p + 1
      else:
         n = n + 1
   if p == 0 or n == 0:
      return 0
   elif p == n:
      return 1
   else:
      entropy = 0 - (
         ((p / (p + n)) * (math.log2(p / (p + n))) + (n / (p + n)) * (math.log2(n/ (p + n)))))
      return entropy

# -----------------------------------------------------------------------

# function to calculate the entropy of attributes
# -----------------------------------------------------------------------
def entropy(dataset, feature, attribute):
   p = 0
   n = 0
   target = dataset.iloc[:, -1]
   targets = list(set(target))
   for i, j in zip(feature, target):
      if i == attribute and j == targets[0]:
         p = p + 1
      elif i == attribute and j == targets[1]:
         n = n + 1
      if p == 0 or n == 0:
         return 0
      elif p == n:
         return 1
      else:
         entropy = 0 - (
            ((p / (p + n)) * (math.log2(p / (p + n))) + (n / (p + n)) * (math.log2(n/ (p + n)))))
         return entropy


# -----------------------------------------------------------------------


# a utility function for checking purity and impurity of a child
# -----------------------------------------------------------------------
```

```
def counter(target, attribute, i):
    p = 0
    n = 0
    targets = list(set(target))
    for j, k in zip(target, attribute):
        if j == targets[0] and k == i:
            p = p + 1
        elif j == targets[1] and k == i:
            n = n + 1
    return p, n

# ---------------------------------------------------------------------
# function that calculates the information gain
# ---------------------------------------------------------------------
def Information_Gain(dataset, feature):
    Distinct = list(set(feature))
    Info_Gain = 0
    for i in Distinct:
        Info_Gain = Info_Gain + feature.count(i) / len(feature) * entropy(dataset,feature, i)
        Info_Gain = base_entropy(dataset) - Info_Gain
    return Info_Gain


# ---------------------------------------------------------------------


# function that generates the childs of selected Attribute
# ---------------------------------------------------------------------
def generate_childs(dataset, attribute_index):
    distinct = list(dataset.iloc[:, attribute_index])
    childs = dict()
    for i in distinct:
        childs[i] = counter(dataset.iloc[:, -1], dataset.iloc[:, attribute_index], i)
    return childs

# ---------------------------------------------------------------------

# function that modifies the dataset according to the impure childs
# ---------------------------------------------------------------------
def modify_data_set(dataset,index, feature, impurity):
    size = len(dataset)
    subdata = dataset[dataset[feature] == impurity]
    del (subdata[subdata.columns[index]])
    return subdata


# ---------------------------------------------------------------------


# function that return attribute with the greatest Information Gain
# ---------------------------------------------------------------------
def greatest_information_gain(dataset):
    max = -1
    attribute_index = 0
    size = len(dataset.columns) - 1
    for i in range(0, size):
        feature = list(dataset.iloc[:, i])
```

```
        i_g = Information_Gain(dataset, feature)
        if max < i_g:
            max = i_g
            attribute_index = i
    return attribute_index


# ----------------------------------------------------------------------


# function to construct the decision tree
# ----------------------------------------------------------------------
def construct_tree(dataset, tree):
    target = dataset.iloc[:, -1]
    impure_childs = []
    attribute_index = greatest_information_gain(dataset)
    childs = generate_childs(dataset, attribute_index)
    tree[dataset.columns[attribute_index]] = childs
    targets = list(set(dataset.iloc[:, -1]))
    for k, v in childs.items():
        if v[0] == 0:
            tree[k] = targets[1]
        elif v[1] == 0:
            tree[k] = targets[0]
        elif v[0] != 0 or v[1] != 0:
            impure_childs.append(k)
    for i in impure_childs:
        sub = modify_data_set(dataset,attribute_index,
        dataset.columns[attribute_index], i)
        tree = construct_tree(sub, tree)
    return tree


# ---------------------------------------------------------------------------
# main function
# ----------------------------------------------------------------------
def main():
    df = pd.read_csv("playtennis.csv")
    tree = dict()
    result = construct_tree(df, tree)
    for key, value in result.items():
        print(key, " => ", value)


# ----------------------------------------------------------------------

if __name__ == "__main__":
    main()
```

OUTPUT

```
outlook  =>  {'sunny': (3, 2), 'overcast': (0, 4), 'rainy': (2, 3)}
overcast  =>  yes
temp  =>  {'mild': (1, 2), 'cool': (1, 1)}
hot  =>  no
```

```
cool  =>  yes
humidity  =>  {'normal': (1, 1)}
high  =>  no
normal  =>  yes
windy  =>  {'Weak': (0, 1), 'Strong': (1, 0)}
Weak  =>  yes
Strong  =>  no
```

DATA SET

OUTLOOK	TEMPERATURE	HUMIDITY	WIND	PLAY TENNIS
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

PROGRAM 5

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

BACKPROPAGATION (training_example, ƞ, nin, nout, nhidden)

PROGRAM

```python
import numpy 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
def derivatives_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
for i 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

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
```

```
 [1.        0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89571283]
 [0.88239245]
 [0.89153673]]
```

PROGRAM 6

Naive Bayesian Classifier

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 few test data sets.

PROGRAM

```
# import necessary libraries
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# Load Data from CSV
data = pd.read_csv('tennisdata.csv')
print("The first 5 Values of data is :\n", data.head())
```

```
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
```

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

```
The First 5 values of the train data is
    Outlook Temperature Humidity   Windy
0     Sunny         Hot     High    Weak
1     Sunny         Hot     High  Strong
2  Overcast         Hot     High    Weak
3      Rain        Mild     High    Weak
4      Rain        Cool   Normal    Weak
```

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

The First 5 values of train output is
0   No
1   No
2   Yes
3   Yes
4   Yes
Name: PlayTennis, dtype: object

```
# convert them in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the Train output is\n", X.head())
```

Now the Train output 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

In [18]:

```
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

Now the Train output is
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.20)

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

from sklearn.metrics import accuracy_score
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

Accuracy is: 0.3333333333333333

PROGRAM 7

EM k-Means algorithm.

PROGRAM

```
#           Kmeans
from sklearn import datasets
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
print(iris)
X_train,X_test,y_train,y_test = train_test_split(iris.data,iris.target)
model =KMeans(n_clusters=3)
model.fit(X_train,y_train)
model.score
print('K-Mean: ',metrics.accuracy_score(y_test,model.predict(X_test)))

#-------Expectation and Maximization----------
from sklearn.mixture import GaussianMixture
model2 = GaussianMixture(n_components=3)
model2.fit(X_train,y_train)
model2.score
print('EM Algorithm:',metrics.accuracy_score(y_test,model2.predict(X_test)))
```

OUTPUT

```
{'data': array([[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, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[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': array([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]), 'frame': None, 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive attributes and the class\n    :Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n                - Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ====================\n                    Min  Max   Mean    SD   Class Correlation\n    ============== ==== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature.  Fisher\'s paper is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant.  One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the data.\n   - Many, many more ...', 'feature_names': ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'filename': 'C:\\Users\\HP\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv'}
K-Mean:  0.8421052631578947
EM Algorithm: 0.9210526315789473
```

PROGRAM 8

Write a program to implement K-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

PROGRAM

In [1]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
iris=datasets.load_iris()
print("Iris Data set loaded...")
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
#random_state=0
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
print("Results of Classification using K-nn with K=1 ")
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));
```

DATASET

```
Iris Data set loaded...
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample: [5.  3.6 1.4 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [4.5 2.3 1.3 0.3]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [5.1 3.5 1.4 0.3]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.1 2.6 5.6 1.4]  Actual-label: 2  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [4.4 2.9 1.4 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [5.2 3.5 1.5 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.2 3.4 5.4 2.3]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333
 Sample: [4.8 3.4 1.9 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.9 3.1 5.4 2.1]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333
 Sample: [5.6 3.  4.1 1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [4.7 3.2 1.6 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.3 2.3 4.4 1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [5.1 3.4 1.5 0.2]  Actual-label: 0  Predicted-label: 0
```

```
Classification Accuracy : 0.9333333333333333
 Sample: [6.  2.9 4.5 1.5]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [5.4 3.9 1.3 0.4]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
```

PROGRAM 9

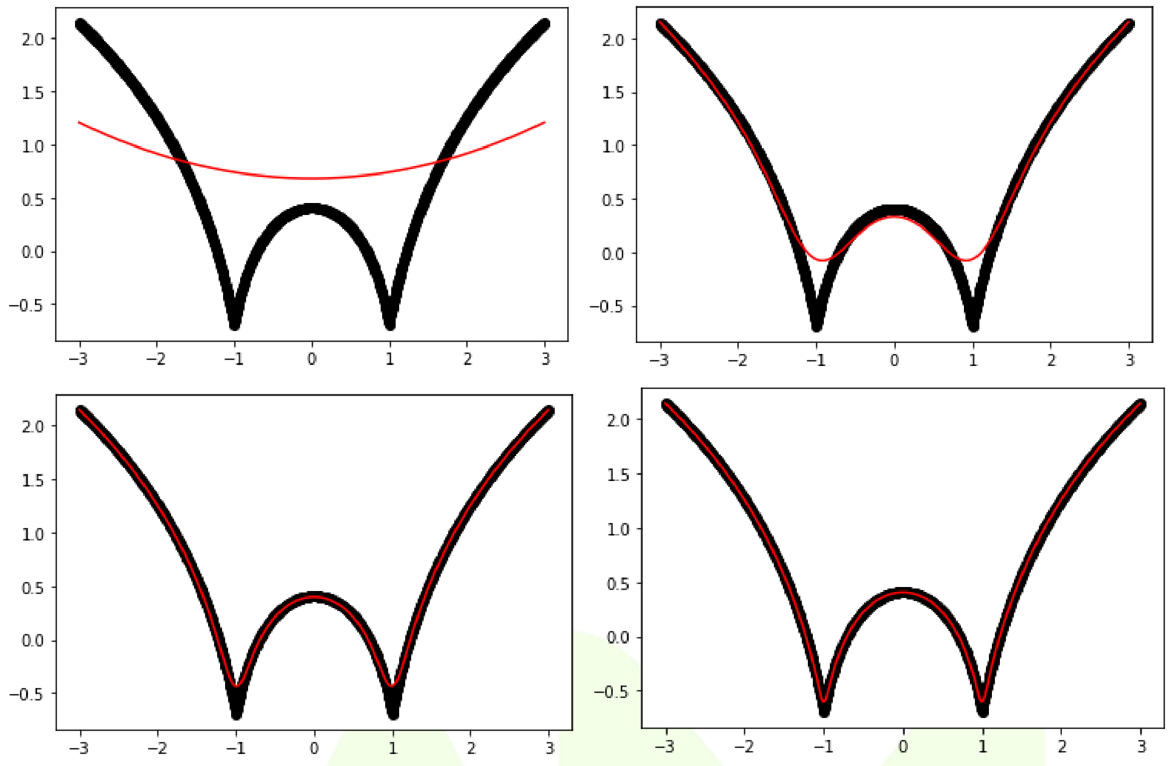
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

PROGRAM

In [2]:

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

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)
```

OUTPUT


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