INDEX

CSP310 : Artificial Intelligence & Machine Learning Lab

PART-A

1. Write a Program to Implement Breadth First Search using Python.

```
graph = {
      '1' : ['2','10'],
      '2' : ['3','8'],
      '3' : ['4'],
      '4' : ['5','6','7'],
      '5' : [],
      '6' : [],
      '7' : [],
      '8' : ['9'],
      '9' : [],
      '10' : []
        }
visited = []
queue = []
def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)
  while queue:
    m = queue.pop(0)
    print (m, end = " ")
    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
print("Following is the Breadth-First Search")
```

```
bfs(visited, graph, '1')
```

Output:

Following is the Breadth-First Search

1 2 10 3 8 4 9 5 6 7

2. Write a Program to Implement Best First Search using Python.

```
from queue import PriorityQueue
import matplotlib.pyplot as plt
import networkx as nx


# for implementing BFS | returns path having lowest cost
def best_first_search(source, target, n):
    visited = [0] * n
    visited[source] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ") # the path having lowest cost
        if u == target:
            break


        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()
# for adding edges to graph
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
```

```
v = int(input("Enter the number of nodes: "))
graph = [[] for i in range(v)] # undirected Graph
e = int(input("Enter the number of edges: "))
print("Enter the edges along with their weights:")
for i in range(e):
    x, y, z = list(map(int, input().split()))
    addedge(x, y, z)

source = int(input("Enter the Source Node: "))
target = int(input("Enter the Target/Destination Node: "))
print("\nPath: ", end = "")
best_first_search(source, target, v)
```

Output:

```
Enter the number of nodes : 4
Enter the number of edges: 5
Enter the edges along with  their weights:
0 1 1
0 2 1
0 3 2
2 3 2
1 3 3
Enter the source node:2
Enter the Target/Destination  node:1
Path : 2 0 1
```

3. Write a Program to Implement Tic-Tac-Toe application using Python.

```
import numpy as np
import random
from time import sleep


def create_board():
    return(np.array([[0, 0, 0],
                     [0, 0, 0],
                     [0, 0, 0]]))


def possibilities(board):
    l = []


    for i in range(len(board)):
        for j in range(len(board)):


            if board[i][j] == 0:
                l.append((i, j))
    return(l)


def random_place(board, player):
    selection = possibilities(board)
    current_loc = random.choice(selection)
    board[current_loc] = player
    return(board)
```

```
def row_win(board, player):
    for x in range(len(board)):
        win = True

        for y in range(len(board)):
            if board[x, y] != player:
                win = False
                continue

        if win == True:
            return(win)
    return(win)

def col_win(board, player):
    for x in range(len(board)):
        win = True

        for y in range(len(board)):
            if board[y][x] != player:
                win = False
                continue

        if win == True:
            return(win)
    return(win)
```

```python
def diag_win(board, player):
    win = True
    y = 0
    for x in range(len(board)):
        if board[x, x] != player:
            win = False
    if win:
        return win
    win = True
    if win:
        for x in range(len(board)):
            y = len(board) - 1 - x
            if board[x, y] != player:
                win = False
    return win


def evaluate(board):
    winner = 0

    for player in [1, 2]:
        if (row_win(board, player) or
            col_win(board,player) or
            diag_win(board,player)):
            winner = player
    if np.all(board != 0) and winner == 0:
        winner = -1
```

```
    return winner
def play_game():
    board, winner, counter = create_board(), 0, 1
    print(board)
    sleep(2)

    while winner == 0:
        for player in [1, 2]:
            board = random_place(board, player)
            print("Board after " + str(counter) + " move")
            print(board)
            sleep(2)
            counter += 1
            winner = evaluate(board)
            if winner != 0:
                break
    return(winner)
print("Winner is: " + str(play_game()))
```

Output:

```
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 0 0]
 [0 0 0]
```

```
 [0 0 1]]
Board after 2 move
[[0 0 2]
 [0 0 0]
 [0 0 1]]
Board after 3 move
[[0 1 2]
 [0 0 0]
 [0 0 1]]
Board after 4 move
[[0 1 2]
 [0 0 2]
 [0 0 1]]
Board after 5 move
[[0 1 2]
 [0 0 2]
 [0 1 1]]
Board after 6 move
[[0 1 2]
 [2 0 2]
 [0 1 1]]
Board after 7 move
[[0 1 2]
 [2 0 2]
 [1 1 1]]
Winner is: 1
```

4. Write a Program to Implement Depth First Search using Python.

```
# Using a Python dictionary to act as an adjacency list
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['6'],
  '6':  [],
  '2' : ['1'],
    '1':[],
  '4' : ['8'],
  '8' : []
}
visited = set() # Set to keep track of visited nodes of graph.
def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

Output:

Following is the Depth-First Search

5

3

2

1

4

8

7

6

5. Write a Program to Implement Water-Jug Problem using Python.

```
from collections import defaultdict
jug1, jug2, aim = 4, 3, 2
visited = defaultdict(lambda: False)
def waterJugSolver(amt1, amt2):
  if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
    print(amt1, amt2)
    return True
  if visited[(amt1, amt2)] == False:
    print(amt1, amt2)
    visited[(amt1, amt2)] = True
    return (waterJugSolver(0, amt2) or
        waterJugSolver(amt1, 0) or
        waterJugSolver(jug1, amt2) or
        waterJugSolver(amt1, jug2) or
        waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
        amt2 - min(amt2, (jug1-amt1))) or
        waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
        amt2 + min(amt1, (jug2-amt2))))
  else:
    return False

print("Steps: ")
waterJugSolver(0, 0)
```

Output:

Steps:

0 0

4 0

4 3

0 3

3 0

3 3

4 2

0 2

True

6. Write a Program to Implement Tower of Hanoi using Python.

```
def  TowerOfHanoi(n , source, destination, auxiliary):
if n==1:
print ("Move disk 1 from source",source,"to destination",destination)
return
TowerOfHanoi(n-1, source, auxiliary, destination)
print ("Move disk",n,"from source",source,"to destination",destination)
TowerOfHanoi(n-1, auxiliary, destination, source)

n = 3
TowerOfHanoi(n,'A','B','C')
```

Output:

```
Move disk 1 from source A to destination B
Move disk 2 from source A to destination C
Move disk 1 from source B to destination C
Move disk 3 from source A to destination B
Move disk 1 from source C to destination A
Move disk 2 from source C to destination B
Move disk 1 from source A to destination B
```

7. Write a Program to Implement N-Queens Problem using Python.

```
global N
N = 4
def printSolution(board):
  for i in range(N):
    for j in range(N):
      print (board[i][j], end = " ")
    print()

def isSafe(board, row, col):
  for i in range(col):
    if board[row][i] == 1:
      return False

  for i, j in zip(range(row, -1, -1),
          range(col, -1, -1)):
    if board[i][j] == 1:
      return False

  for i, j in zip(range(row, N, 1),
          range(col, -1, -1)):
    if board[i][j] == 1:
      return False
  return True
```

```
def solveNQUtil(board, col):
  if col >= N:
    return True
  for i in range(N):
    if isSafe(board, i, col):
      board[i][col] = 1
      if solveNQUtil(board, col + 1) == True:
        return True
      board[i][col] = 0
  return False


def solveNQ():
  board = [ [0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0] ]
  if solveNQUtil(board, 0) == False:
    print ("Solution does not exist")
    return False
  printSolution(board)
  return True
solveNQ()
```

Output:

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

True

8. Write a Program to Implement A* algorithm using Python.

```
class  Node():
   def  __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position


        self.g = 0
        self.h = 0
        self.f = 0
    def  __eq__(self, other):
        return self.position == other.position
def  astar(maze, start, end):
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0
    open_list = []
    closed_list = []
    open_list.append(start_node)
    while len(open_list) > 0:
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index
```

```
    open_list.pop(current_index)
    closed_list.append(current_node)
    if current_node == end_node:
        path = []
        current = current_node
        while current is not None:
            path.append(current.position)
            current = current.parent
        return path[::-1] # Return reversed path
    children = []
    for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares
        node_position = (current_node.position[0] + new_position[0], current_node.position[1] + new_position[1])
              if node_position[0] > (len(maze) - 1) or node_position[0] < 0 or node_position[1] > (len(maze[len(maze)-1]) -1) or node_position[1] < 0:
            continue
        if maze[node_position[0]][node_position[1]] != 0:
            continue
        new_node = Node(current_node, node_position)
        children.append(new_node)
    for child in children:
                for closed_child in closed_list:
            if child == closed_child:
                continue
        child.g = current_node.g + 1
        child.h = ((child.position[0] - end_node.position[0]) ** 2) + ((child.position[1] - end_node.position[1]) ** 2)
```

```
            child.f = child.g + child.h
            for open_node in open_list:
                if child == open_node and child.g > open_node.g:
                    continue
           open_list.append(child)
def  main():
  maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 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, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 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]]
    start = (0, 0)
    end = (7, 6)
    path = astar(maze, start, end)
    print(path)
if __name__ == '__main__':
    main()
```

Output:

[(0, 0), (1, 1), (2, 2),(3, 3), (4, 3), (5, 4), (6, 5), (7, 6)]

9. Write a Program to Implement AO* algorithm using Python.

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}

    def applyAOStar(self):
        self.aoStar(self.start, False)

    def getNeighbors(self, v):
        return self.graph.get(v,'')

    def getStatus(self,v):
        return self.status.get(v,0)

    def setStatus(self,v, val):
        self.status[v]=val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0)

    def setHeuristicNodeValue(self, n, value):
```

```
        self.H[n]=value
    def printSolution(self):
        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE:",self.start)
        print("------------------------------------------------------------")
        print(self.solutionGraph)
        print("------------------------------------------------------------")

    def computeMinimumCostChildNodes(self, v):
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v):
            cost=0
            nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)
            if flag==True:
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList
                flag=False
            else:
                if minimumCost>cost:
                    minimumCost=cost
```

```
            costToChildNodeListDict[minimumCost]=nodeList
    return minimumCost, costToChildNodeListDict[minimumCost]


def aoStar(self, v, backTracking):
    print("HEURISTIC VALUES :", self.H)
    print("SOLUTION GRAPH :", self.solutionGraph)
    print("PROCESSING NODE :", v)
    print("-----------------------------------------------------------------------------------------")
    if self.getStatus(v) >= 0:
        minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
        print(minimumCost, childNodeList)
        self.setHeuristicNodeValue(v, minimumCost)
        self.setStatus(v,len(childNodeList))
        solved=True
        for childNode in childNodeList:
            self.parent[childNode]=v
            if self.getStatus(childNode)!=-1:
                solved=solved & False
        if solved==True:
            self.setStatus(v,-1)
            self.solutionGraph[v]=childNodeList
        if v!=self.start:
            self.aoStar(self.parent[v], True)
        if backTracking==False:
            for childNode in childNodeList:
                self.setStatus(childNode,0)
```

```
            self.aoStar(childNode, False)


print ("Graph - 1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
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()
```

Output:

```
Graph - 1
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------------------------------------
```

6 ['G']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

10 ['B', 'C']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : G

----

8 ['I']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : B

----

8 ['H']

HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

12 ['B', 'C']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : I

----

0 []

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': []}

PROCESSING NODE : G

----

1 ['I']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I']}

PROCESSING NODE : B

----

2 ['G']

HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

----

6 ['B', 'C']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : C

----

2 ['J']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

----

6 ['B', 'C']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : J

----

0 []

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}

PROCESSING NODE : C

----

1 ['J']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

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

PROCESSING NODE : A

----

5 ['B', 'C']

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

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

10. Write a Program to Implement Travelling Salesman problem using Python.

```
from sys import maxsize
from itertools import permutations
V = 4


def travellingSalesmanProblem(graph, s):
    # store all vertex apart from source vertex
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)


# store minimum weight Hamiltonian Cycle
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:


        # store current Path weight(cost)
        current_pathweight = 0


        # compute current path weight
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
```

```
        # Update minimum
        min_path = min(min_path, current_pathweight)
    return min_path

# Driver Code
if __name__ == "__main__":

# matrix representation of graph
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
            [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

Output:

80

11. Write a Program to Implement 8-Puzzle Problem using Python.

```
import copy

from heapq import heappush, heappop

n = 3

row = [ 1, 0, -1, 0 ]

col = [ 0, -1, 0, 1 ]


class priorityQueue:

    def __init__(self):

        self.heap = []

    def push(self, k):

        heappush(self.heap, k)


    def pop(self):

        return heappop(self.heap)

    def empty(self):

        if not self.heap:

            return True

        else:

            return False


class node:

    def __init__(self, parent, mat, empty_tile_pos,

                cost, level):

        self.parent = parent
```

```
            self.mat = mat
self.empty_tile_pos = empty_tile_pos
self.cost = cost
self.level = level
def __lt__(self, nxt):
            return self.cost < nxt.cost
def calculateCost(mat, final) -> int:
      count = 0
      for i in range(n):
            for j in range(n):
                  if ((mat[i][j]) and
                        (mat[i][j] != final[i][j])):
                        count += 1
                        return count
def newNode(mat, empty_tile_pos, new_empty_tile_pos,
                  level, parent, final) -> node:
      new_mat = copy.deepcopy(mat)
      x1 = empty_tile_pos[0]
      y1 = empty_tile_pos[1]
      x2 = new_empty_tile_pos[0]
      y2 = new_empty_tile_pos[1]
      new_mat[x1][y1], new_mat[x2][y2] = new_mat[x2][y2], new_mat[x1][y1]

      cost = calculateCost(new_mat, final)
      new_node = node(parent, new_mat, new_empty_tile_pos,cost, level)
```

```
        return new_node
def printMatrix(mat):
        for i in range(n):
                for j in range(n):
                        print("%d " % (mat[i][j]), end = " ")

                print()

def isSafe(x, y):
        return x >= 0 and x < n and y >= 0 and y < n

def printPath(root):
        if root == None:
                return

        printPath(root.parent)
        printMatrix(root.mat)
        print()
def solve(initial, empty_tile_pos, final):
                pq = priorityQueue()
        cost = calculateCost(initial, final)
        root = node(None, initial,empty_tile_pos, cost, 0)
        pq.push(root)
        while not pq.empty():
                minimum = pq.pop()
```

```
        if minimum.cost == 0:
            printPath(minimum)
            return
        for i in range(n):
            new_tile_pos = [
                minimum.empty_tile_pos[0] + row[i],
                minimum.empty_tile_pos[1] + col[i], ]

            if isSafe(new_tile_pos[0], new_tile_pos[1]):
                child=newNode(minimum.mat,minimum.empty_tile_pos,new_tile_pos,
                    minimum.level + 1minimum, final,)
                pq.push(child)
initial = [ [ 1, 2, 3 ],[ 5, 6, 0 ],[ 7, 8, 4 ] ]
final = [ [ 1, 2, 3 ],[ 5, 8, 6 ],[ 0, 7, 4 ] ]
empty_tile_pos = [ 1, 2 ]
solve(initial, empty_tile_pos, final)
```

utput:

```
1 2 3
5 6 0
7 8 4

1 2 3
5 0 6
7 8 4

1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4
```

PART-B

1. Write a program to implement the FIND-S Algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

```
import csv

hypo=['%','%','%','%','%','%']

with open('Training_examples.csv') as csv_file:

readcsv = csv.reader(csv_file, delimiter=',')

data=[]

print("\nThe given training examples are:")

for row in readcsv:

print(row)

if row[len(row)-1] =='Yes':

data.append(row)

print("\nThe positive examples are:")

for x in data:

print(x)

TotalExamples=len(data)

i=0

j=0

k=0

print("\nThe steps of the Find-s algorithm are\n",hypo)

list =[]

p=0

d=len(data[p])-1

for j in range(d):
```

```
list.append(data[i][j])
hypo=list
for i in range(1,TotalExamples):
for k in range(d):
if hypo[k]!=data[i][k]:
hypo[k]='?'
else:
hypo[k]
print(hypo)
print("\nThe maximally specific Find-s hypothesis for the given training examples is");
list=[]
for i in range(d):
list.append(hypo[i])
print(list)
```

Output:

The given training examples are:

['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']

The positive examples are:

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The steps of the Find-s algorithm are

['%', '%', '%', '%', '%', '%']

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

['Sunny', 'Warm', '?', 'Strong', '?', '?']

The maximally specific Find-s hypothesis for the given training examples is

['Sunny', 'Warm', '?', 'Strong', '?', '?']

2. Write a program to implement the Candidate-Elimination algorithm, For a given set of training data examples stored in a .CSV file.

```
import csv
with open("Training_examples.csv") as f:
  csv_file=csv.reader(f)
    data=list(csv_file)
    s=data[1][:-1]
    g=[['?' for i in range(len(s))] for j in range(len(s))]
    for i in data:
        if i[-1]=="Yes":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    s[j]='?'
                    g[j][j]='?'
        elif i[-1]=="No":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    g[j][j]=s[j]
                else:
                    g[j][j]="?"
        print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)
        print(s)
        print(g)
    gh=[]
```

```
for i in g:
    for j in i:
        if j!='?':
            gh.append(i)
            break
print("\nFinal specific hypothesis:\n",s)
print("\nFinal general hypothesis:\n",gh)
```

Output:

Steps of Candidate Elimination Algorithm 1

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Steps of Candidate Elimination Algorithm 4

['Sunny', 'Warm', '?', 'Strong', '?', '?']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final specific hypothesis:

['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final general hypothesis:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

3. Write a program to demonstrate the working of the ID3 algorithm.

```
import ast

import csv

import math

import os

def load_csv_to_header_data(filename):

    path = os.path.normpath(os.getcwd() + filename)

    print(path)

    fs = csv.reader(open(path))

    all_row = []

    for r in fs:

        all_row.append(r)

    headers = all_row[0]

    idx_to_name, name_to_idx = get_header_name_to_idx_maps(headers)

    data = { 'header': headers,'rows': all_row[1:],'name_to_idx': name_to_idx,'idx_to_name':
idx_to_name}

    return data

def get_header_name_to_idx_maps(headers):

    name_to_idx = {}

    idx_to_name = {}

    for i in range(0, len(headers)):

        name_to_idx[headers[i]] = i

        idx_to_name[i] = headers[i]

    return idx_to_name, name_to_idx
```

```
def project_columns(data, columns_to_project):
    data_h = list(data['header'])
    data_r = list(data['rows'])
    all_cols = list(range(0,len(data_h)))
    columns_to_project_ix = [data['name_to_idx'][name] for name in columns_to_project]
    columns_to_remove = [cidx for cidx in all_cols if cidx not in columns_to_project_ix]
    for delc in sorted(columns_to_remove, reverse=True):
        del data_h[delc]
        for r in data_r:
            del r[delc]
    idx_to_name, name_to_idx = get_header_name_to_idx_maps(data_h)
    return {'header': data_h, 'rows': data_r,'name_to_idx': name_to_idx,'idx_to_name':
idx_to_name}
def get_uniq_values(data):
    idx_to_name = data['idx_to_name']
    idxs = idx_to_name.keys()
    val_map = {}
    for idx in iter(idxs):
        val_map[idx_to_name[idx]] = set()
    for data_row in data['rows']:
        for idx in idx_to_name.keys():
            att_name = idx_to_name[idx]
            val = data_row[idx]
            if val not in val_map.values():
```

```
            val_map[att_name].add(val)
    return val_map
def get_class_labels(data,target_attribute):
    rows = data['rows']
    col_idx = data['name_to_idx'][target_attribute]
    labels = {}
    for r in rows:
        val = r[col_idx]
        if val in labels:
            labels[val] = labels[val] + 1
        else:
            labels[val] = 1
    return labels
def entropy(n, labels):
    ent = 0
    for label in labels.keys():
        p_x = labels[label] / n
        ent += - p_x * math.log(p_x, 2)
    return ent
def partition_data(data, group_att):
    partitions = {}
    data_rows = data['rows']
    partition_att_idx = data['name_to_idx'][group_att]
```

```
    for row in data_rows:
        row_val = row[partition_att_idx]
        if row_val not in partitions.keys():
            partitions[row_val] = {'name_to_idx': data['name_to_idx'],'idx_to_name':
data['idx_to_name'],'rows': list()}
        partitions[row_val]['rows'].append(row)
    return partitions
def avg_entropy_w_partitions(data, splitting_att, target_attribute):     # find uniq values of
splitting att
    data_rows = data['rows']
    n = len(data_rows)
    partitions = partition_data(data, splitting_att)
    avg_ent = 0
    for partition_key in partitions.keys():
        partitioned_data = partitions[partition_key]
        partition_n = len(partitioned_data['rows'])
        partition_labels = get_class_labels(partitioned_data, target_attribute)
        partition_entropy = entropy(partition_n, partition_labels)
        avg_ent += partition_n / n * partition_entropy
    return avg_ent, partitions
def most_common_label(labels):
    mcl = max(labels, key=lambda k: labels[k])
    return mcl
def id3(data, uniqs, remaining_atts, target_attribute):
    labels = get_class_labels(data, target_attribute)
```

```
node = {}
if len(labels.values()) == 1:
    node['label'] = next(iter(labels.keys()))
    return node
if len(remaining_atts) == 0:
    node['label'] = most_common_label(labels)
    return node
n = len(data['rows'])
ent = entropy(n, labels)
max_info_gain = None
max_info_gain_att = None
max_info_gain_partitions = None
for remaining_att in remaining_atts:
    avg_ent, partitions = avg_entropy_w_partitions(data, remaining_att, target_attribute)
    info_gain = ent - avg_ent
    if max_info_gain is None or info_gain > max_info_gain:
        max_info_gain = info_gain
        max_info_gain_att = remaining_att
        max_info_gain_partitions = partitions
if max_info_gain is None:
    node['label'] = most_common_label(labels)
    return node
node['attribute'] = max_info_gain_att
```

```
    node['nodes'] = {}
    remaining_atts_for_subtrees = set(remaining_atts)
    remaining_atts_for_subtrees.discard(max_info_gain_att)
    uniq_att_values = uniqs[max_info_gain_att]

    for att_value in uniq_att_values:
        if att_value not in max_info_gain_partitions.keys():
            node['nodes'][att_value] = {'label': most_common_label(labels)}
            continue
        partition = max_info_gain_partitions[att_value]
        node['nodes'][att_value] = id3(partition, uniqs, remaining_atts_for_subtrees,
target_attribute)
    return node
def load_config(config_file):
    with open(config_file, 'r') as myfile:
        data = myfile.read().replace('\n', '')
        print(data)
    return ast.literal_eval(data)


def pretty_print_tree(root):
    stack = []
    rules = set()
    def traverse(node, stack, rules):
        if 'label' in node:
```

```
            stack.append(' THEN ' + node['label'])
            rules.add(''.join(stack))
            stack.pop()
        elif 'attribute' in node:
            ifnd = 'IF ' if not stack else ' AND '
            stack.append(ifnd + node['attribute'] + ' EQUALS ')
            for subnode_key in node['nodes']:
                stack.append(subnode_key)
                traverse(node['nodes'][subnode_key], stack, rules)
                stack.pop()
            stack.pop()
    traverse(root, stack, rules)
    print(os.linesep.join(rules))
def main():
    argv ='tennis.cfg'
    print("Command line args are {}: ".format(argv))
    config = load_config(argv)
    print(config)
    data = load_csv_to_header_data(config['data_file'])
    data = project_columns(data, config['data_project_columns'])
    target_attribute = config['target_attribute']
    remaining_attributes = set(data['header'])
    remaining_attributes.remove(target_attribute)
    print(remaining_attributes)
```

```
    uniqs = get_uniq_values(data)
    root = id3(data, uniqs, remaining_attributes, target_attribute)
    pretty_print_tree(root)
if __name__ == "__main__": main()
```

Output:

Command line args are tennis.cfg:

{   'data_file' : '//tennis.csv',   'data_mappers' : [],   'data_project_columns' : ['Outlook', 'Temperature', 'Humidity', 'Windy', 'PlayTennis'],   'target_attribute' : 'PlayTennis'}

{'data_file': '//tennis.csv', 'data_mappers': [], 'data_project_columns': ['Outlook', 'Temperature', 'Humidity', 'Windy', 'PlayTennis'], 'target_attribute': 'PlayTennis'}

C:\Users\ADMIN\machine learning\tennis.csv

{'Outlook', 'Humidity', 'Temperature', 'Windy'}

IF Outlook EQUALS Sunny AND Humidity EQUALS Normal THEN Yes

IF Outlook EQUALS Overcast THEN Yes

IF Outlook EQUALS Rainy AND Windy EQUALS False THEN Yes

IF Outlook EQUALS Sunny AND Humidity EQUALS High THEN

No

4. Write a program to Build an Artificial Neural Network by implementing the Back-propagation algorithm and test the same using appropriate data sets.

```
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=7000 #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)
    hiddengrad = derivatives_sigmoid(hlayer_act)  #how much hidden layer wts contributed to
error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr # dotproduct of nextlayererror and currentlayerop
# bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *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:
[[0.8946056 ]
[0.88075857]
[0.89439617]]
```

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

```
# Importing the libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd


# Importing the dataset

dataset = pd.read_csv('User_Data.csv')

x = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values


# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)


# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)

x_test = sc.transform(x_test)
```

#Displaying the dataset

Dataset:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

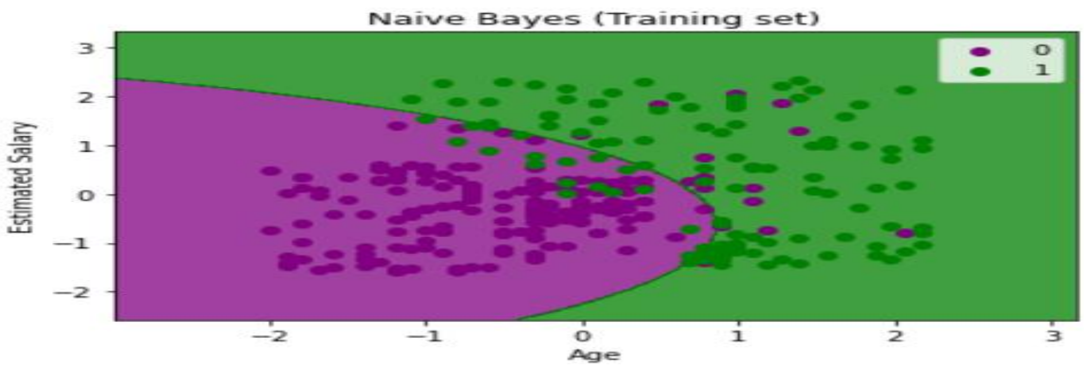
```
# Fitting I Bayes to the Training set

from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(x_train, y_train)


GaussianNB()

# Predicting the Test set results

y_pred = classifier.predict(x_test)


# Making the Confusion Matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print(“Confusion Matrix : \n”,cm)
```

Output:

```
Confusion Matrix :
 [[65  3]
 [ 7 25]]
```

```
# Visualising the Training set results

from matplotlib.colors import ListedColormap

x_set, y_set = x_train, y_train

X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() – 1, stop = x_set[:, 0].max() + 1, step
= 0.01),
```

```
nm.arange(start = x_set[:, 1].min() – 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
        alpha = 0.75, cmap = ListedColormap((‘purple’, ‘green’)))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for I, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
          c = ListedColormap((‘purple’, ‘green’))(i), label = j)
mtp.title(‘I Bayes (Training set)’)
mtp.xlabel(‘Age’)
mtp.ylabel(‘Estimated Salary’)
mtp.legend()
mtp.show()
```




```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
```

X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() – 1, stop = x_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x_set[:, 1].min() – 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap((‘red’, ‘green’)))

mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for I, j in enumerate(nm.unique(y_set)):

mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

c = ListedColormap((‘red’, ‘green’))(i), label = j)

mtp.title(‘I Bayes (test set)’)

mtp.xlabel(‘Age’)

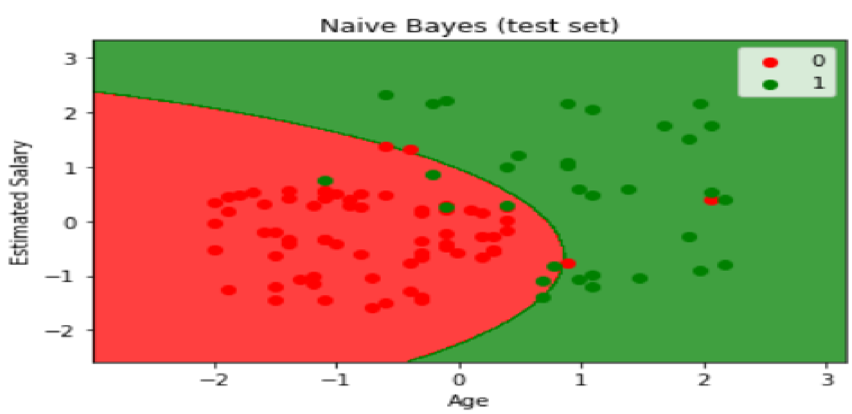
mtp.ylabel(‘Estimated Salary’)

mtp.legend()

mtp.show()

from sklearn.metrics import accuracy_score

print (“Accuracy : “, accuracy_score(y_test, y_pred))




```
Accuracy :  0.9
```

6.Write a program to implement Logistic regression classifier to find accuracy for training and test fruit data set.

```
#Data Pre-procesing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('User_Data.csv')
```

Data_set:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
#Data Pre-procesing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('User_Data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
From sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)

#Predicting the test set result
y_pred= classifier.predict(x_test)
print(y_pred)
```

Output:

```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0
 0 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1]
```

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print("Confusion Matrix : \n",cm)
```

Output:

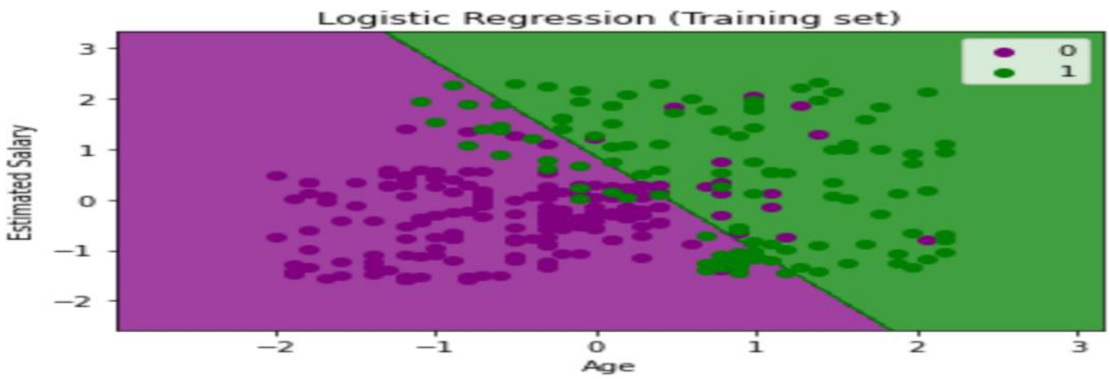
Confusion Matrix :

[[65 3]

[8 24]]

```
#Visualizing the training set result

from matplotlib.colors import ListedColormap

x_set, y_set = x_train, y_train

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() – 1, stop = x_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x_set[:, 1].min() – 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap((‘purple’,’green’ )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for I, j in enumerate(nm.unique(y_set)):

    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

        c = ListedColormap((‘purple’, ‘green’))(i), label = j)

mtp.title(‘Logistic Regression (Training set)’)

mtp.xlabel(‘Age’)

mtp.ylabel(‘Estimated Salary’)

mtp.legend()

mtp.show()
```

Output:



#Visulaizing the test set result

from matplotlib.colors import ListedColormap

x_set, y_set = x_test, y_test

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() – 1, stop = x_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x_set[:, 1].min() – 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap((‘purple’,’green’)))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for I, j in enumerate(nm.unique(y_set)):

    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

        c = ListedColormap((‘purple’, ‘green’))(i), label = j)

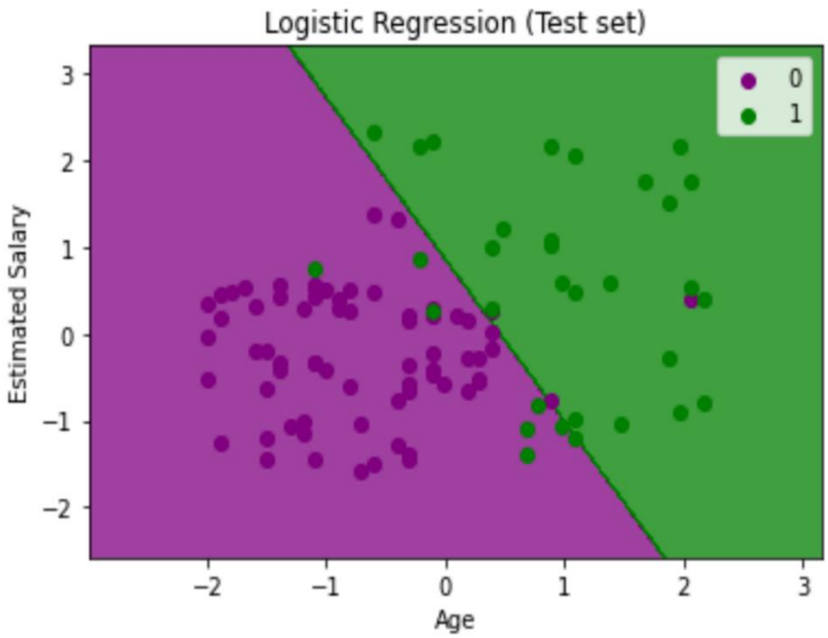
mtp.title(‘Logistic Regression (Test set)’)

mtp.xlabel(‘Age’)

mtp.ylabel(‘Estimated Salary’)

```
mtp.legend()

mtp.show()

from sklearn.metrics import accuracy_score

print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

Output:



Accuracy : 0.89

7.Write a program to implement K- Nearest Neighbors' classifier to find accuracy for training and test fruit data set.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('User_Data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
From sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

#Fitting K-NN classifier to the training set

from sklearn.neighbors import KneighborsClassifier

classifier= KneighborsClassifier(n_neighbors=5, metric='minkowski',p=2)classifier.fit(x_train, y_train)

Output:

KneighborsClassifier()

#Predicting the test set result

y_pred= classifier.predict(x_test)

#Creating the Confusion matrix

from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_test, y_pred)

print("Confusion Matrix : \n",cm)

Output:

Confusion Matrix :

[[64 4]

[3 29]]

#Visulaizing the   rrange   set result

from matplotlib.colors import ListedColormap

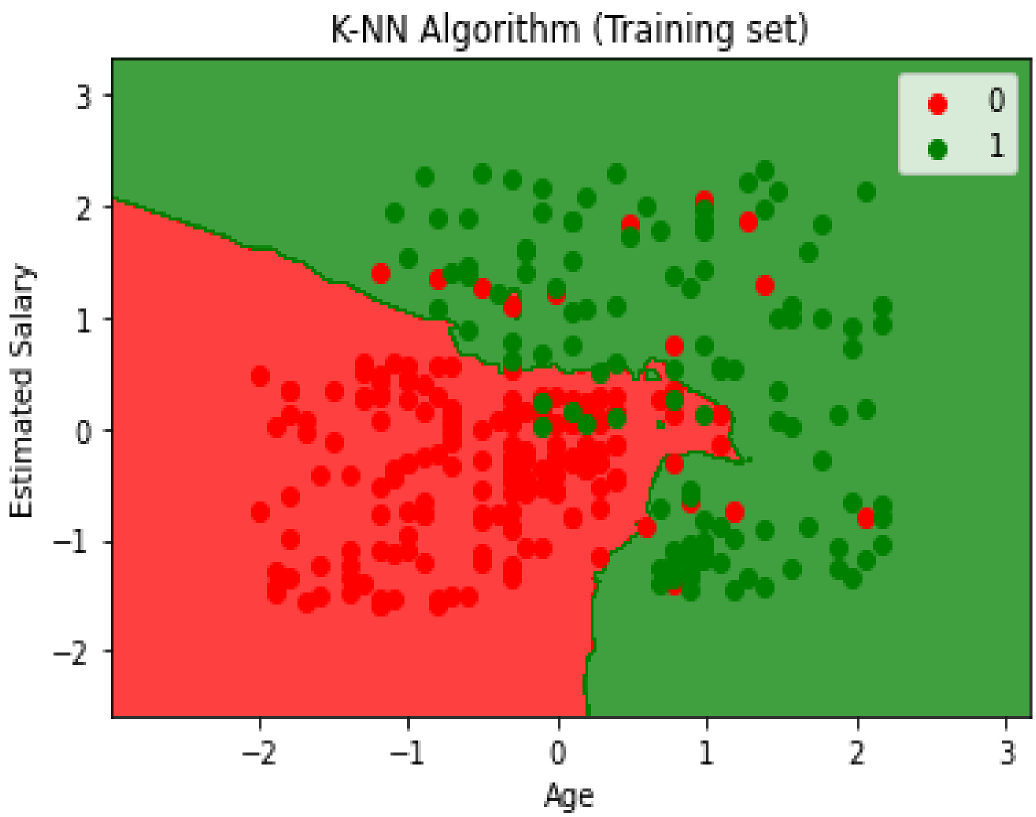
x_set, y_set = x_train, y_train

```
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() – 1, stop = x_set[:, 0].max() + 1, step =0.01),nm.arange(start = x_set[:, 1].min() – 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap((‘red’,’green’ )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for I, j in enumerate(nm.unique(y_set)):

    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

        c = ListedColormap((‘red’, ‘green’))(i), label = j)

mtp.title(‘K-NN Algorithm (Training set)’)

mtp.xlabel(‘Age’)

mtp.ylabel(‘Estimated Salary’)

mtp.legend()

mtp.show()
```

Output:

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.



#Visualizing the test set result

from matplotlib.colors import ListedColormap

x_set, y_set = x_test, y_test

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() – 1, stop = x_set[:, 0].max() + 1, step =0.01), nm.arange(start = x_set[:, 1].min() – 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape), al pha = 0.75, cmap = ListedColormap((‘red’,’green’)))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for I, j in enumerate(nm.unique(y_set)):

    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

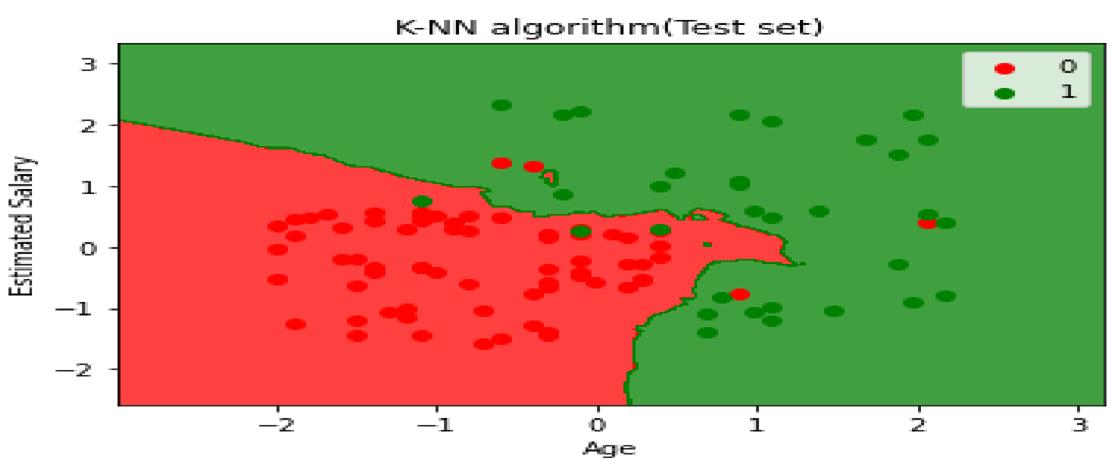
        c = ListedColormap((‘red’, ‘green’))(i), label = j)

```
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

Output:

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.



8.Write a program to implement SVM classifier to find accuracy for training and testing fruit data set.

```
Import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.svm import SVC


fruits = pd.read_table('fruit_data_with_colors.txt')

feature_names = ['mass', 'width', 'height', 'color_score']

X = fruits[feature_names]

Y = fruits['fruit_label']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y)

X_train = MinMaxScaler().fit_transform(X_train)

X_test = MinMaxScaler().fit_transform(X_test)

svm = SVC().fit(X_train, Y_train)

print('Accuracy of SVM classifier on training set : {:.2f}'.format(svm.score(X_train, Y_train)))

print('Accuracy of SVM classifier on testing set : {:.2f}'.format(svm.score(X_test, Y_test)))
```

Output:

Accuracy of SVM classifier on training set : 0.98

Accuracy of SVM classifier on testing set : 0.93

9. Write a program to implement Decision Tree classifier to find accuracy for training and test fruit data set.

```
Import pandas as pd

fruits = pd.read_table('fruit_data_with_colors.txt')

feature_names = ['mass', 'width', 'height', 'color_score']

X = fruits[feature_names]

y = fruits['fruit_label']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Decision Tree classifier

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier().fit(X_train, y_train)

print('Accuracy of Decision Tree classifier on training set: {:.2f}'

     .format(clf.score(X_train, y_train)))

print('Accuracy of Decision Tree classifier on test set: {:.2f}'

     .format(clf.score(X_test, y_test)))
```

Output:

Accuracy of Decision Tree classifier on training set: 1.00

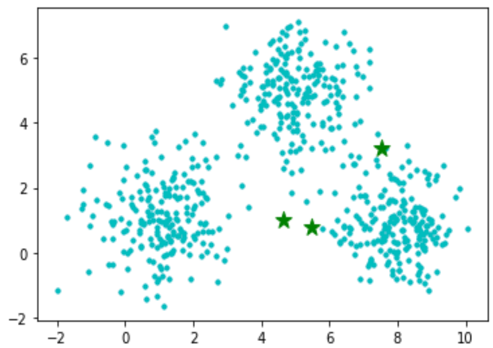
Accuracy of Decision Tree classifier on test set: 0.73

10 . Write a program to implement K-means clustering using random samples

```
from copy import deepcopy

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from matplotlib import pyplot as plt

# Set three centers, the model should predict similar results

center_1 = np.array([1,1])

center_2 = np.array([5,5])

center_3 = np.array([8,1])

# Generate random data and center it to the three centers

data_1 = np.random.randn(200, 2) + center_1

data_2 = np.random.randn(200,2) + center_2

data_3 = np.random.randn(200,2) + center_3

data = np.concatenate((data_1, data_2, data_3), axis = 0)

plt.scatter(data[:,0], data[:,1], s=7)

# Number of clusters

k = 3

# Number of training data

n = data.shape[0]

# Number of features in the data

c = data.shape[1]

# Generate random centers, here we use sigma and mean to ensure it represent the whole data

mean = np.mean(data, axis = 0)

std = np.std(data, axis = 0)
```

```
centers = np.random.randn(k,c)*std + mean

# Plot the data and the centers generated as random

plt.scatter(data[:,0], data[:,1], s=9,color='c')

plt.scatter(centers[:,0], centers[:,1], marker='*', c='g', s=150)

plt.show()
```

Output:



11.Write a program to implement gradient boosting problem in python

```
from sklearn import datasets

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.pipeline import make_pipeline

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.decomposition import PCA

from sklearn.metrics import mean_squared_error


bhp = datasets.load_boston()


X_train, X_test, y_train, y_test = train_test_split(bhp.data, bhp.target, random_state=42, test_size=0.1)


sc = StandardScaler()

X_train_std = sc.fit_transform(X_train)

X_test_std = sc.transform(X_test)


gbr_params = {‘n_estimators’: 1000,

        ‘max_depth’: 3,

        ‘min_samples_split’: 5,

        ‘learning_rate’: 0.01,

        ‘loss’: ‘ls’}
```

```
gbr = GradientBoostingRegressor(**gbr_params)
gbr.fit(X_train_std, y_train)
print("Model Accuracy: %.3f" % gbr.score(X_test_std, y_test))
mse = mean_squared_error(y_test, gbr.predict(X_test_std))
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

output:

Model Accuracy: 0.918

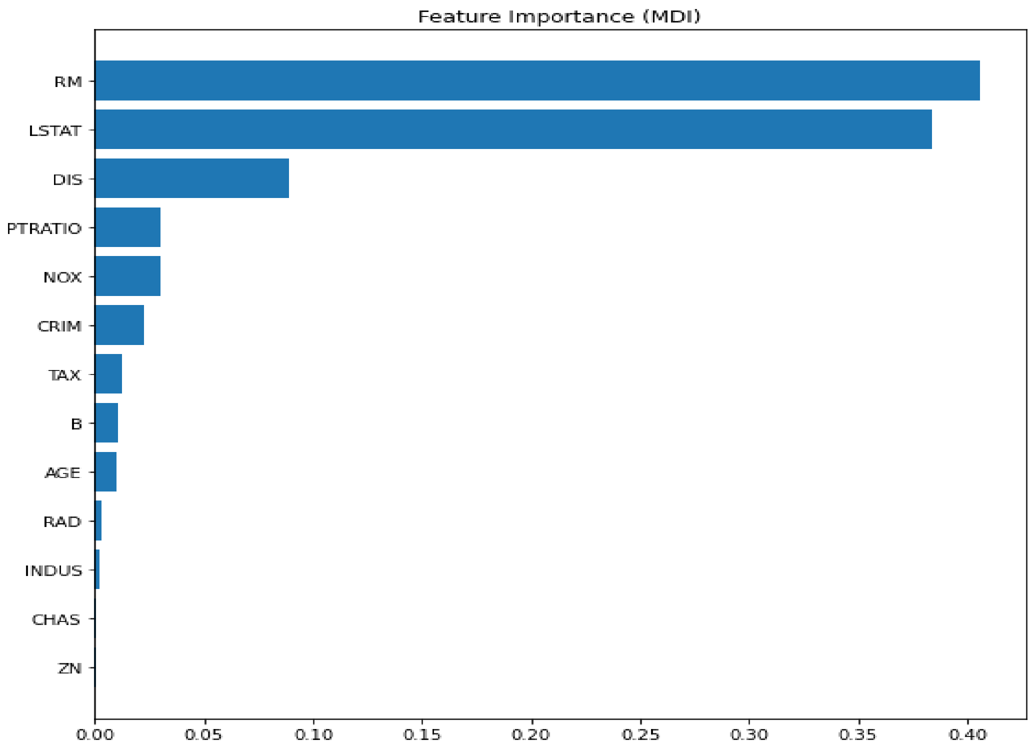
The mean squared error (MSE) on test set: 5.1449

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.inspection import permutation_importance

feature_importance = gbr.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
fig = plt.figure(figsize=(8, 8))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, np.array(bhp.feature_names)[sorted_idx])
plt.title('Feature Importance (MDI)')
result = permutation_importance(gbr, X_test_std, y_test, n_repeats=10,
                                random_state=42, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
```

fig.tight_layout()

plt.show()

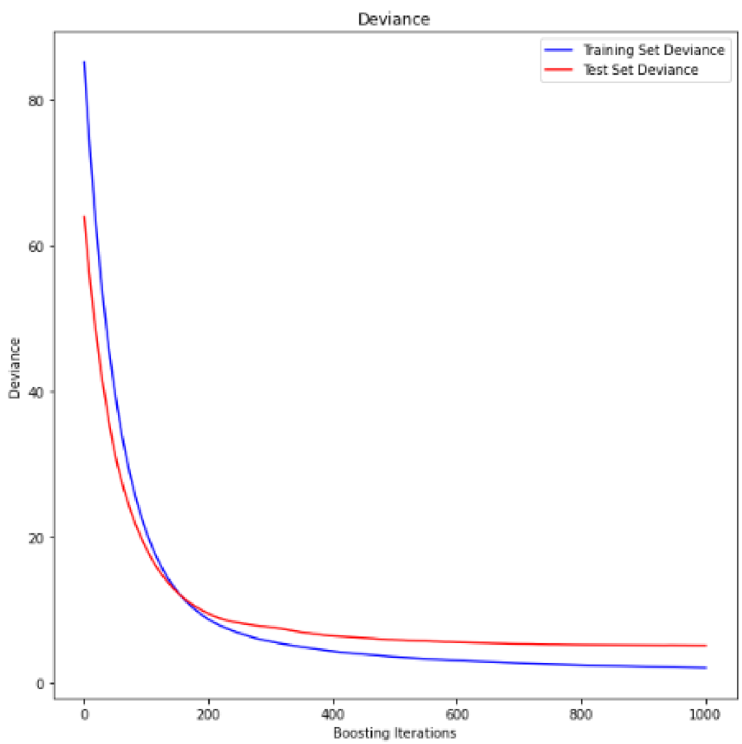
Output:




```
test_score = np.zeros((gbr_params['n_estimators'],), dtype=np.float64)
for I, y_pred in enumerate(gbr.staged_predict(X_test_std)):
    test_score[i] = gbr.loss_(y_test, y_pred)
fig = plt.figure(figsize=(8, 8))
plt.subplot(1, 1, 1)
plt.title('Deviance')
plt.plot(np.arange(gbr_params['n_estimators']) + 1, gbr.train_score_, 'b-',
        label='Training Set Deviance')
```

```
plt.plot(np.arange(gbr_params['n_estimators']) + 1, test_score, 'r-',
    label='Test Set Deviance')
plt.legend(loc='upper right')
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
fig.tight_layout()
plt.show()
```

Output:



* * *