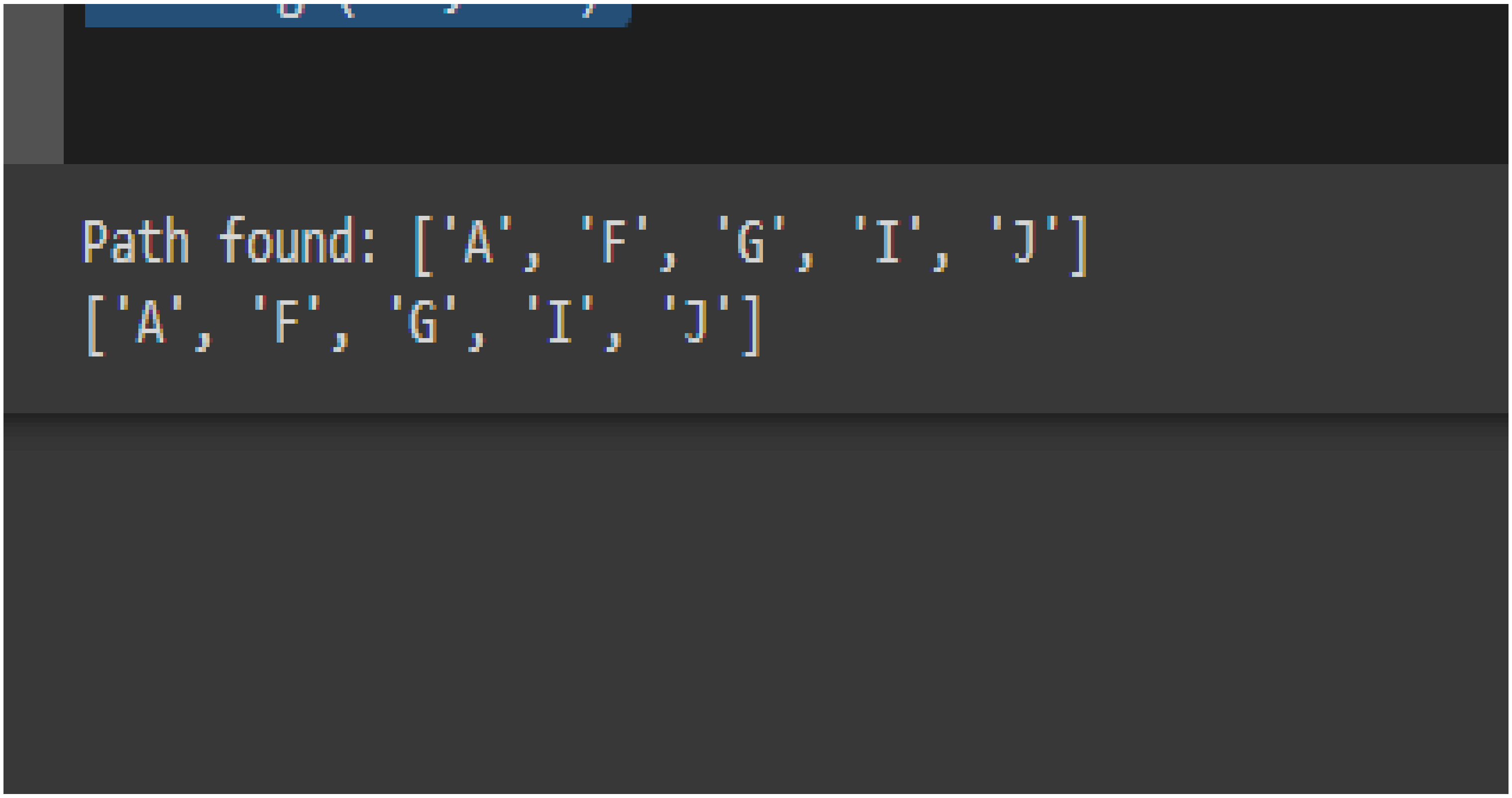
Program 1
AStar-Algorithm

```
def aStarAlgo(start_node, stop_node):

    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node


    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v



        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n

                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)

        if n == None:
            print('Path does not exist!')
            return None
        if n == stop_node:
            path = []

            while parents[n] != n:
                path.append(n)
                n = parents[n]

            path.append(start_node)

            path.reverse()
```

```
            print('Path found: {}'.format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)

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


def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

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]


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
AOStar-Algorithm

```
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)
            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)


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}
graph2 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'D': [[('E', 1), ('F', 1)]]
}
G2 = Graph(graph2, h2, 'A')
G2.applyAOStar()
G2.printSolution()
```

Output : For Graph1

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

```
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5,
'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : G
----------------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8,
'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : B
----------------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8,
'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
----------------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8,
'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : I
----------------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8,
'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': []}
PROCESSING NODE   : G
----------------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1,
'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I']}
PROCESSING NODE   : B
----------------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1,
'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : A
----------------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1,
'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : C
----------------------------------------------------------------------------------------
--------
```

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

Output : For Graph2

```
HEURISTIC VALUES  : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5,
'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
--------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5,
'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : D
--------------------------------------------------------------------------------
--------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5,
'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
--------------------------------------------------------------------------------
--------
```

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

Program 3
Candidate Elimination Algorithm

```
import numpy as np
import pandas as pd

data = pd.read_csv('prgm.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)
```

```
    print("Final Specific_h: ", s_final, sep="\n")
    print("Final General_h: ", g_final, sep="\n")
```

Output :

Instances are:

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

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

['Rain' 'Cold' 'High' 'Strong' 'Warm' 'Change']

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

Target Values are: ['yes' 'yes' 'no' 'yes']

Initialization of specific_h and genearal_h

Specific Boundary: ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

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

Instance 1 is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Instance is Positive

Specific Boundary after 1 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

Instance is Positive

Specific Boundary after 2 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is ['Rain' 'Cold' 'High' 'Strong' 'Warm' 'Change']

Instance is Negative

Specific Boundary after 3 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

Generic Boundary after 3 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Instance 4 is ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']

Instance is Positive

Specific Boundary after 4 Instance is ['Sunny' 'Warm' '?' 'Strong' '?' '?']

Generic Boundary after 4 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:

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

Final General_h:

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

Program 4

ID3 Algorithm(Decision Tree)

```
import math
import csv

def load_csv(filename):
  lines = csv.reader(open(filename, "r"))
  dataset = list(lines)
  headers = dataset.pop(0)
  return dataset, headers

class Node:
  def __init__(self, attribute) :
    self.attribute = attribute
    self.children = []
    self.answer = ""

def subtables(data, col, delete) :
  dic = {}
  coldata = [row[col] for row in data]
  attr=list(set(coldata))
  for k in attr:
    dic[k] = []
  for y in range(len(data)):
    key = data[y][col]
    if delete:
      del data[y][col]
    dic[key].append(data[y])
  return attr, dic

def entropy(S):
  attr=list(set(S))
  if len(attr) == 1:
    return 0
  counts =[0,0]
  for i in range(2):
    counts[i]=sum([1 for x in S if attr[i] == x])/(len(S)*1.0)
  sums=0
  for cnt in counts:
    sums+= -1*cnt*math.log(cnt, 2)
  return sums

def compute_gain(data, col):
   attValues, dic=subtables(data, col, delete=False)
```

```
    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attValues)):
      ratio = len(dic[attValues[x]])/(len(data)*1.0)
      entro = entropy([row[-1] for row in dic[attValues[x]]])
      total_entropy-= ratio*entro
    return total_entropy

def build_tree(data,features) :
  lastcol=[row[-1] for row in data]
  if (len(set(lastcol))) == 1:
    node=Node("")
    node.answer = lastcol[0]
    return node
  n = len(data[0])-1
  gains =[compute_gain(data, col) for col in range(n) ]
  split=gains.index(max(gains))
  node=Node(features[split])

  fea=features[:split]+features[split+1:]
  attr, dic = subtables(data, split, delete=True)
  for x in range(len(attr)):
    child = build_tree(dic[attr[x]], fea)
    node.children.append((attr[x], child))
  return node

def print_tree (node, level) :
  if node.answer!= "":
    print(" "*level, node.answer)
    return
  print(" "*level, node.attribute)
  for value, n in node.children:
    print(" "*(level+1), value)
    print_tree (n, level + 2)


def classify (node,x_test,features):
  if node.answer!="" :
    print(node.answer)
    return
  pos = features.index(node.attribute)
  for value, n in node.children:
    if x_test[pos]==value:
        classify(n,x_test, features)



dataset, features = load_csv("id3.csv")
```

```
node = build_tree(dataset, features)
print("The decision tree for the dataset using ID3 algorithm is ")
print_tree(node, 0)
testdata, features =load_csv("id3_test_1.csv")
for xtest in testdata:
  print("The test instance xtest",xtest)
  print("The predicted labe1 ", end="" )
  classify(node, xtest,features)
```

Output :

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
  rain
   Wind
    weak
     yes
    strong
     no
  overcast
   yes
  sunny
   Humidity
    normal
     yes
    high
     no
The test instance xtest ['rain', 'cool', 'normal', 'strong']
The predicted labe1 no
The test instance xtest ['sunny', 'mild', 'normal', 'strong']
The predicted labe1 yes
```

Program 5
Backpropagation Algorithm

```
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)
y = y/100


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


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


epoch=500000
eta=0.2

input_neurons = 2
hidden_neurons = 3
output_neurons = 1


wh=np.random.uniform(size=(input_neurons,hidden_neurons))
bh=np.random.uniform(size=(1,hidden_neurons))
wout=np.random.uniform(size=(hidden_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))


for i in range(epoch):

    h_ip=np.dot(X,wh)+bh
    h_act = sigmoid(h_ip)
    o_ip=np.dot(h_act,wout)+bout
    output = sigmoid(o_ip)

    EO = y-output
    outgrad = sigmoid_grad(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = sigmoid_grad(h_act)
    d_hiddenlayer = EH * hiddengrad
```

```
    wout += h_act.T.dot(d_output) *eta
    wh += X.T.dot(d_hiddenlayer) *eta



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:
 [[0.91999675]
 [0.86000135]
 [0.89000106]]
```

Program 6
Naive Bayes Classifier

```
import math
import random
import csv


def encode_class(mydata):
  classes = []
  for i in range(len(mydata)):
    if mydata[i][-1] not in classes:
      classes.append(mydata[i][-1])
  for i in range(len(classes)):
    for j in range(len(mydata)):
      if mydata[j][-1] == classes[i]:
        mydata[j][-1] = i
  return mydata


def splitting(mydata, ratio):
  train_num = int(len(mydata) * ratio)
  train = []

  test = list(mydata)
  while len(train) < train_num:
    index = random.randrange(len(test))
    train.append(test.pop(index))
  return train, test


def groupUnderClass(mydata):
  dict = {}
  for i in range(len(mydata)):
    if (mydata[i][-1] not in dict):
      dict[mydata[i][-1]] = []
    dict[mydata[i][-1]].append(mydata[i])
  return dict


def mean(numbers):
  return sum(numbers) / float(len(numbers))

def std_dev(numbers):
  avg = mean(numbers)
```

```
  variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
  return math.sqrt(variance)

def MeanAndStdDev(mydata):
  info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]

  del info[-1]
  return info


def MeanAndStdDevForClass(mydata):
  info = {}
  dict = groupUnderClass(mydata)
  for classValue, instances in dict.items():
    info[classValue] = MeanAndStdDev(instances)
  return info



def calculateGaussianProbability(x, mean, stdev):
  expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
  return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo


def calculateClassProbabilities(info, test):
  probabilities = {}
  for classValue, classSummaries in info.items():
    probabilities[classValue] = 1
    for i in range(len(classSummaries)):
      mean, std_dev = classSummaries[i]
      x = test[i]
      probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
  return probabilities



def predict(info, test):
  probabilities = calculateClassProbabilities(info, test)
  bestLabel, bestProb = None, -1
  for classValue, probability in probabilities.items():
    if bestLabel is None or probability > bestProb:
      bestProb = probability
      bestLabel = classValue
  return bestLabel
```

```
def getPredictions(info, test):
  predictions = []
  for i in range(len(test)):
    result = predict(info, test[i])
    predictions.append(result)
  return predictions


def accuracy_rate(test, predictions):
  correct = 0
  for i in range(len(test)):
    if test[i][-1] == predictions[i]:
      correct += 1
  return (correct / float(len(test))) * 100.0



filename = r'naivedata.csv'



mydata = csv.reader(open(filename, "rt"))
mydata = list(mydata)
mydata = encode_class(mydata)
for i in range(len(mydata)):
  mydata[i] = [float(x) for x in mydata[i]]



ratio = 0.7
train_data, test_data = splitting(mydata, ratio)
print('Total number of examples are: ', len(mydata))
print('Out of these, training examples are: ', len(train_data))
print("Test examples are: ", len(test_data))


info = MeanAndStdDevForClass(train_data)


predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print("Accuracy of your model is: ", accuracy)
```

Output :

```
Total number of examples are:  768
Out of these, training examples are:  537
Test examples are:  231
Accuracy of your model is:  76.62337662337663
```

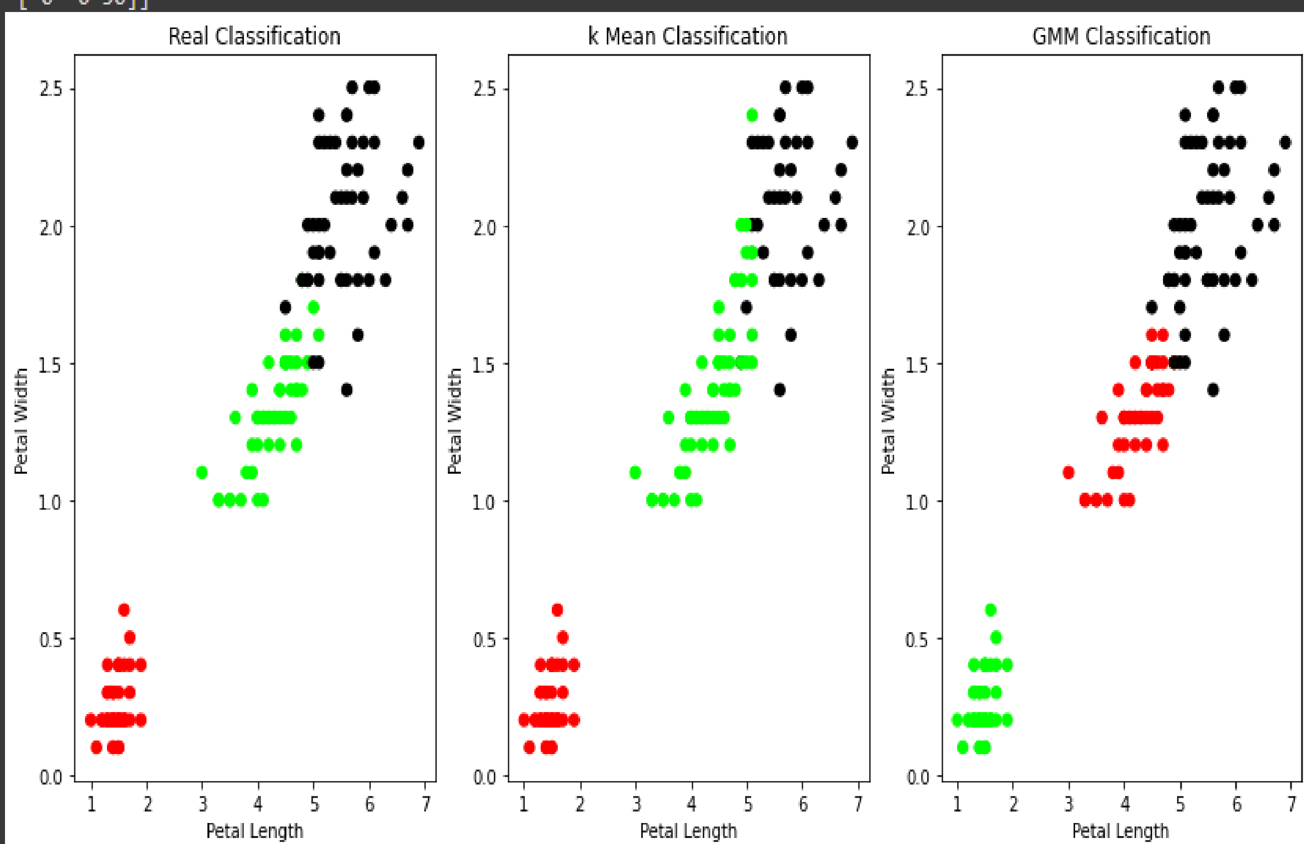
Program 7

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris=datasets.load_iris()
x=pd.DataFrame(iris.data)
x.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(iris.target)
y.columns=['Targets']

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

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

plt.subplot(1,3,1)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.subplot(1,3,2)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[model.labels_],s=40)
plt.title('k Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('the accuracy score of k-mean:',sm.accuracy_score(y,model.labels_))
print('the confusion matrix of k-mean:',sm.confusion_matrix(y,model.labels_))


from sklearn import preprocessing
scaler=preprocessing.StandardScaler()
scaler.fit(x)
xsa=scaler.transform(x)
xs=pd.DataFrame(xsa,columns=x.columns)

from sklearn.mixture import GaussianMixture
gm=GaussianMixture(n_components=3)
gm.fit(xs)

y_gm=gm.predict(xs)
```

```
plt.subplot(1,3,3)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[y_gm],s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('the accuracy score of EM:',sm.accuracy_score(y,y_gm))
print('the confusion matrix of k-mean:',sm.confusion_matrix(y,y_gm))
```

Output :

```
the accuracy score of k-mean: 0.8933333333333333
the confusion matrix of k-mean: [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
the accuracy score of EM: 0.3333333333333333
the confusion matrix of k-mean: [[ 0 50  0]
 [45  0  5]
 [ 0  0 50]]
```



Program 8

```
from sklearn.datasets import load_iris
iris = load_iris()
print ("Feature Names: ", iris.feature_names, "Iris Data:", iris.data, "Target Names:", iris.target_names, "Target: ", iris.target)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split( iris.data, iris.target, test_size= 0.25)
from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier()
clf.fit(x_train, y_train)
print("Predicted Data")
print(clf.predict (x_test))
prediction=clf.predict(x_test)
print("Test data :")
print(y_test)
diff=prediction-y_test
print("Result is ")
print(diff)
print('Total no of samples misclassified=',sum(abs(diff)))
```

Output:

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

```
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[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 Names: ['setosa' 'versicolor' 'virginica'] Target:  [0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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]
Predicted Data
[2 1 1 1 1 2 0 0 2 2 1 2 2 2 2 2 0 1 0 1 0 1 1 2 0 2 0 0 1 1 0 2 0 1 0 0 1
 1]
Test data :
[2 1 1 1 1 2 0 0 2 2 1 2 2 2 2 2 0 1 0 1 0 1 2 2 0 2 0 0 1 1 0 2 0 1 0 0 1
 1]
Result is
[ 0  0  0  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  0  0  0  0  0  0]
Total no of samples misclassified= 1
```

Program 9

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernel(point,xmat,k):
  m,n=np.shape(xmat)
  weights=np.mat(np.eye((m)))
  for j in range(m):
    diff=point-x[j]
    weights[j,j]=np.exp(diff*diff.T/(-2.0*k**2))
  return weights
def localWeight(point,xmat,ymat,k):
  wei=kernel(point,xmat,k)
  w=(x.T*(wei*x)).I*(x.T*(wei*ymat.T))
  return w
def localWeightRegression(xmat,ymat,k):
  m,n=np.shape(xmat)
  ypred=np.zeros(m)
  for i in range(m):
    ypred[i]=xmat[i]*localWeight(xmat[i],xmat,ymat,k)
  return ypred

data=pd.read_csv('tips.csv')
bill=np.array(data.total_bill)
tip=np.array(data.tip)
mbill=np.mat(bill)
mtip=np.mat(tip)
m=np.shape(mbill)[1]
one=np.mat(np.ones(m))
x=np.hstack((one.T,mbill.T))
ypred=localWeightRegression(x,mtip,0.5)
sortIndex=x[:,1].argsort(0)
xsort=x[sortIndex][:,0]
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='green')
ax.plot(xsort[:,1],ypred[sortIndex],color='red',linewidth=5)
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```

Output :