1. Implement A* Search algorithm

ALGORITHM

Step 1: Initialize the open list

Step 2: Initialize the closed list, put the starting node on the open list (you can leave its f at zero)

Step 3. while the open list is not empty

    a) find the node with the least f on the open list, call it "q"
    b) pop q off the open list
    c) generate q's 8 successors and set their parents to q
    d) for each successor
        i) if successor is the goal, stop search
          successor.g = q.g + distance between
                              successor and q
          successor.h = distance from goal to successor
          successor.f = successor.g + successor.h
        ii) if a node with the same position as
           successor is in the OPEN list which has a lower f than successor, skip this
           successor
        iii) if a node with the same position as
           successor is in the CLOSED list which has a lower f than successor, skip this
           successor otherwise, add the node to the open list
         end (for loop)
      e) push q on the closed list
    end (while loop)

PROGRAM

```
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': 11,
        'B': 6,
        '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': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}
aStarAlgo('A', 'J')
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
```

```
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0,
    }
    return H_dist[n]
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('A', 2), ('C', 1), ('G', 9)],
    'C': [('B', 1)],
    'D': [('E', 6), ('G', 1)],
    'E': [('A', 3), ('D', 6)],
    'G': [('B', 9), ('D', 1)]
}
aStarAlgo('A', 'G')
```

OUTPUT

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

Path found: ['A', 'E', 'D', 'G']

2. Implement AO* Search algorithm.

ALGORITHM

Step-1: Create an initial graph with a single node (start node).

Step-2: Transverse the graph following the current path, accumulating node that has not yet been expanded or solved.

Step-3: Select any of these nodes and explore it. If it has no successors then call this value- FUTILITY else calculate f'(n) for each of the successors.

Step-4: If f'(n)=0, then mark the node as SOLVED.

Step-5: Change the value of f'(n) for the newly created node to reflect its successors by backpropagation.

Step-6: Whenever possible use the most promising routes, If a node is marked as SOLVED then mark the parent node as SOLVED.

Step-7: If the starting node is SOLVED or value is greater than FUTILITY then stop else repeat from Step-2.

PROGRAM

```
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']}
```

3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm in python to output a description of the set of all hypotheses consistent with the training examples.

ALGORITHM

For each training example d, do:

If d is positive example

Remove from G any hypothesis h inconsistent with d

For each hypothesis s in S not consistent with d:

Remove s from S

Add to S all minimal generalizations of s consistent with d and having a generalization in G

Remove from S any hypothesis with a more specific h in S

If d is negative example

Remove from S any hypothesis h inconsistent with d

For each hypothesis g in G not consistent with d:

Remove g from G

Add to G all minimal specializations of g consistent with d and having a specialization in S

Remove from G any hypothesis having a more general hypothesis in G

PROGRAM

```
import numpy as np

import pandas as pd

data = pd.read_csv(path+'/enjoysport.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")
```

DATASET

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

OUTPUT

```
Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' '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 Bundary 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 Bundary after 2 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']

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

Instance 3 is ['rainy' 'cold' 'high' 'strong' 'warm' 'change']

Instance is Negative

Specific Bundary 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 Bundary 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', '?', '?', '?', '?']]

4. Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

ALGORITHM

ID3(Examples, Target_attribute, Attributes)

Create a Root node for the tree

If all Examples are positive, Return the single-node tree Root, with label = +

If all Examples are negative, Return the single-node tree Root, with label = -

If Attributes is empty, Return the single-node tree Root,

with label = most common value of Target_attribute in Examples

Otherwise Begin

A ← the attribute from Attributes that best* classifies Examples

The decision attribute for Root ← A

For each possible value, vi, of A,

Add a new tree branch below Root, corresponding to the test A = vi

Let Examples vi, be the subset of Examples that have value vi for A

If Examples vi , is empty

Then below this new branch add a leaf node with

label = most common value of Target_attribute in Examples

Else

below this new branch add the subtree

ID3(Examples vi, Targe_tattribute, Attributes – {A}))

End

Return Root

PROGRAM

```
import pandas as pd

import math

import numpy as np

data = pd.read_csv("3-dataset.csv")

features = [feat for feat in data]

features.remove("answer")

class Node:

    def __init__(self):

        self.children = []

        self.value = ""

        self.isLeaf = False

        self.pred = ""

def entropy(examples):

    pos = 0.0

    neg = 0.0

    for _, row in examples.iterrows():

        if row["answer"] == "yes":

            pos += 1

        else:

            neg += 1

    if pos == 0.0 or neg == 0.0:

        return 0.0

    else:

        p = pos / (pos + neg)
```

```
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain
def ID3(examples, attrs):
    root = Node()
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    uniq = np.unique(examples[max_feat])
    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata) == 0.0:
            newNode = Node()
```

```
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
root = ID3(data, features)
printTree(root)
```

DATASET

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***	***PlayTennis***
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

OUTPUT

```
Outlook
    Overcast ->  ['yes']
    Rainy
        Wind
            Strong ->  ['No']
            Weak ->  ['yes']
    Sunny
        Humidity
            High ->  ['No']
            Normal ->  ['yes']
```

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

ALGORITHM

BACKPROPAGATION (*training_example*, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form ($\vec{x}$, $\vec{t}$), where ($\vec{x}$) is the vector of network input values, ($\vec{t}$) and is the vector of target network output values.
η is the learning rate (e.g., .05). n_i, is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.
The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do

    - For each ($\vec{x}$, $\vec{t}$), in training examples, Do

      Propagate the input forward through the network:
      1. Input the instance $\vec{x}$, to the network and compute the output o_u of every unit u in the network.

      Propagate the errors backward through the network:

    2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

    3. For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

    4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

    Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

PROGRAM

```
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=5 #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)
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    print ("-----------Epoch-", i+1, "Starts----------")
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends----------\n")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

OUTPUT

```
-----------Epoch- 1 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.71669304]
 [0.70551416]
 [0.72402119]]
-----------Epoch- 1 Ends----------
-----------Epoch- 2 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
```

```
Predicted Output:
 [[0.72021397]
 [0.70884396]
 [0.72759384]]
-----------Epoch- 2 Ends----------


-----------Epoch- 3 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.72361682]
 [0.71206417]
 [0.7310446 ]]
-----------Epoch- 3 Ends----------


-----------Epoch- 4 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
```

```
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.72690698]
 [0.71517975]
 [0.73437912]]
-----------Epoch- 4 Ends----------


-----------Epoch- 5 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.73008956]
 [0.71819539]
 [0.73760274]]
-----------Epoch- 5 Ends----------
```

Input:

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

Actual Output:

```
[[0.92]
 [0.86]
 [0.89]]
```

Predicted Output:

```
 [[0.73008956]
 [0.71819539]
 [0.73760274]]
```

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

PROGRAM

```
import numpy as np

import random

import csv

import pdb

def read_data(filename):

    with open(filename,'r')as csvfile:

        datareader =csv.reader(csvfile)

        metadata=next(datareader)

        traindata=[]

        for row in datareader:

            traindata.append(row)

    return(metadata,traindata)

def splitdataset(dataset,splitratio):

    trainsize=int(len(dataset)*splitratio)

    trainset=[]

    testset=list(dataset)

    i=0

    while len(trainset)<trainsize:

        trainset.append(testset.pop(i))

    return[trainset,testset]

def classifydata(data,test):

    total_size=data.shape[0]
```

```
print("\n")
print("training data size=",total_size)
print("test data size=",test.shape[0])
countyes=0
countno=0
probyes=0
probno=0
print("\n")
print("target count probability")
for x in range(data.shape[0]):
    if data[x,data.shape[1]-1]=='yes':
        countyes=countyes+1
    if data[x,data.shape[1]-1]=='No':
        countno=countno+1
probyes=countyes/total_size
probno=countno/total_size
print("yes","\t",countyes,"\t",probyes)
print("no","\t",countno,"\t",probno)
prob0=np.zeros((test.shape[1]-1))
prob1=np.zeros((test.shape[1]-1))
accuracy=0
print("\n")
print("instance prediction target")
for t in range(test.shape[0]):
    for k in range(test.shape[1]-1):
```

```
            count1=count0=0
            for j in range(data.shape[0]):
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='No':
                    count0=count0+1
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='yes':
                    count1=count1+1
            prob0[k]=count0/countno
            prob1[k]=count1/countyes
        probNo=probno
        probYes=probyes
        for i in range(test.shape[1]-1):
            probNo=probNo*prob0[i]
            probYes=probYes*prob1[i]
        if probNo>probYes:
                predict='no'
        else:
                predict='yes'


        print(t+1,"\t",predict,"\t",test[t,test.shape[1]-1])
        if predict==test[t,test.shape[1]-1]:
                accuracy+=1
    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")
    return
metadata,traindata=read_data("data3.csv")
```

```
print("attribute names of the traning dta are:", metadata)
splitratio=0.6
trainingset,testset=splitdataset(traindata,splitratio)
training=np.array(trainingset)
print("\n tarining data set are")
for x in trainingset:
    print(x)
testing=np.array(testset)
print("\n the test data set are:")
for  x in testing:
      print(x)
classifydata(training, testing)
```

DATASET

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***	***PlayTennis***
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes

D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

OUTPUT

attribute names of the traning dta are: ['Outlook', 'Temperature', 'Humidity', 'Wind', 'Target']

training data set are

['Sunny', 'Hot', 'High', 'Weak', 'no']

['Sunny', 'Hot', 'High', 'Strong', 'no']

['Overcast', 'Hot', 'High', 'Weak', 'yes']

['Rainy', 'Mild', 'High', 'Weak', 'yes']

['Rainy', 'Cool', 'Normal', 'Weak', 'yes']

['Rainy', 'Cool', 'Normal', 'Strong', 'no']

['Overcast', 'Cool', 'Normal', 'Strong', 'yes']

['Sunny', 'Mild', 'High', 'Weak', 'no']

the test data set are:

['Sunny' 'Cool' 'Normal' 'Weak' 'yes']

['Rainy' 'Mild' 'Normal' 'Weak' 'yes']

['Sunny' 'Mild' 'Normal' 'Strong' 'yes']

['Overcast' 'Mild' 'High' 'Strong' 'yes']

['Overcast' 'Hot' 'Normal' 'Weak' 'yes']

['Rainy' 'Mild' 'High' 'Strong' 'no']

training data size= 8

test data size= 6

target count probability

yes	4	0.5
no	4	0.5

instance prediction target

1	yes	yes
2	yes	yes
3	yes	yes
4	yes	yes
5	yes	yes
6	yes	no

accuracy 83.33333333333334 %

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

ALGORITHM

EM Algorithm:

Step 1: Given a set of incomplete data, consider a set of starting parameters.

Step 2: Expectation step (E – step): Using the observed available data of the dataset, estimate (guess) the values of the missing data.

Step 3: Maximization step (M – step): Complete data generated after the expectation (E) step is used in order to update the parameters.

Step 4: Repeat step 2 and step 3 until convergence.

K- Means Clustering:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which mean reassign each data point to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

PROGRAM

```
from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("iris.csv", names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))

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

# REAL PLOT

plt.subplot(1,3,1)

plt.title('Real')

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT

model=KMeans(n_clusters=3, random_state=0).fit(X)

plt.subplot(1,3,2)

plt.title('KMeans')

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))

print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))
```

```
# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

DATASET (iris.csv total 150 rows in this the attribute names should be ignored while using it as .csv file)

sepal.length	**sepal.width**	**petal.length**	**petal.width**	**variety**
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
…	…	…	…	…
…	…	…	…	…
5.9	3	5.1	1.8	Iris-virginica

OUTPUT

The accuracy score of K-Mean: 0.24

The Confusion matrix of K-Mean:

[[0 50 0]

[48 0 2]

[14 0 36]]

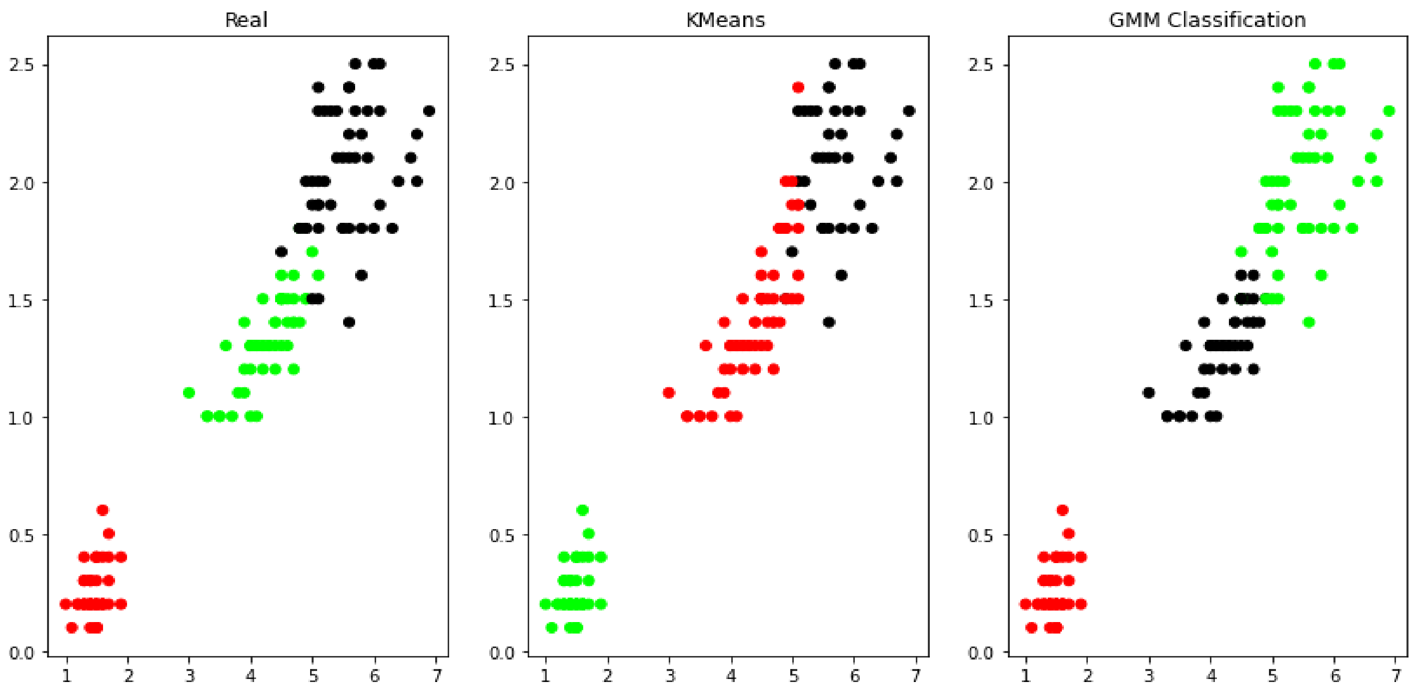
The accuracy score of EM: 0.36666666666666664

The Confusion matrix of EM:

[[50 0 0]

[0 5 45]

[0 50 0]]



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.

ALGORITHM

K-Nearest Neighbor

Training algorithm:

- For each training example (x, f (x)), add the example to the list training examples

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

- Where, $f(x_i)$ function to calculate the mean value of the k nearest training examples.

PROGRAM

```
import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

dataset = pd.read_csv("iris.csv", names=names)

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, -1]

print(X.head())
```

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)
classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)
ypred = classifier.predict(Xtest)
i = 0
print ("\n-----------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-----------------------------------------------------------------------")
for label in ytest:
    print ('%-25s %-25s' % (label, ypred[i]), end="")
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("-----------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-----------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-----------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-----------------------------------------------------------------------")
```

DATASET (iris.csv)

sepal.length	sepal.width	petal.length	petal.width	variety
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
…	…	…	…	…
…	…	…	…	…
5.9	3	5.1	1.8	Iris-virginica

OUTPUT

```
sepal-length  sepal-width  petal-length  petal-width
0      5.1      3.5      1.4      0.2
1      4.9      3.0      1.4      0.2
2      4.7      3.2      1.3      0.2
3      4.6      3.1      1.5      0.2
4      5.0      3.6      1.4      0.2
------------------------------------------------------------------------
Original Label        Predicted Label        Correct/Wrong
------------------------------------------------------------------------
Iris-setosa        Iris-setosa        Correct
```

```
Iris-setosa             Iris-setosa             Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-virginica          Iris-virginica          Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-virginica          Iris-virginica          Correct
Iris-virginica          Iris-virginica          Correct
Iris-versicolor         Iris-virginica          Wrong
Iris-versicolor         Iris-versicolor         Correct
Iris-setosa             Iris-setosa             Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-virginica          Iris-virginica          Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-versicolor         Iris-versicolor         Correct
----------------------------------------------------------------------
Confusion Matrix:
 [[3 0 0]
 [0 7 1]
 [0 0 4]]
----------------------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support


Iris-setosa       1.00      1.00      1.00         3
Iris-versicolor   1.00      0.88      0.93         8
```

```
Iris-virginica       0.80      1.00      0.89         4

      accuracy                           0.93        15
     macro avg       0.93      0.96      0.94        15
  weighted avg       0.95      0.93      0.93        15
```

Accuracy of the classifer is 0.93

----

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

ALGORITHM

1. Read the Given data Sample to X and the curve (linear or non linear) to Y

2. Set the value for Smoothening parameter or Free parameter say τ

3. Set the bias /Point of interest set x0 which is a subset of X

4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using:

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

6. Prediction = x0*β

PROGRAM

```
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('10-dataset.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();
```

DATASET (restaurant bill.csv)

total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2

OUTPUT