AIML LAB MANUAL

LAB 1 : """Implement A* search 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] is None:
            pass
        else:
            for (m, weight) in get_neighbours(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 is 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_neighbours(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']

'''
```

LAB 2 : """Recursive implementation of AO* algorithm"""

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

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

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

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

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

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

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

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

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

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

```
        cost = 0
        nodeList = []
        for c, weight in nodeInfoTupleList:
            cost = cost + self.getHeuristicNodeValue(c) + weight
            nodeList.append(c)

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

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

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

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

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

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

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

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

```
        if not backTracking:  # check the current call is not for backtracking
            for childNode in childNodeList:  # for each Minimum Cost child node
                self.setStatus(childNode, 0)  # set the status of child node to 0(needs exploration)
                self.aoStar(childNode,
                            False)  # Minimum Cost child node is further explored with backtracking status as false


h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}
G1 = Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()

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

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

<u># OUTPUT:</u>

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

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

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

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

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

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

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

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

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : D

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : F

HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': []}
PROCESSING NODE : D

HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE : A

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

{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}

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

```
import csv

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

    specific = data[0][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

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

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

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

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
            if j != '?':
                gh.append(i)
                break
```

```
    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```

trainingexamples.csv

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

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

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

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

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

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

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

LAB 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 knowledge to classify a new sample """

```
import pandas as pd
from pprint import pprint
from sklearn.feature_selection import mutual_info_classif
from collections import Counter

def id3(df, target_attribute, attribute_names, default_class=None):
    cnt=Counter(x for x in df[target_attribute])
    if len(cnt)==1:
        return next(iter(cnt))

    elif df.empty or (not attribute_names):
         return default_class

    else:
        gainz = mutual_info_classif(df[attribute_names],df[target_attribute],discrete_features=True)
        index_of_max=gainz.tolist().index(max(gainz))
        best_attr=attribute_names[index_of_max]
        tree={best_attr:{}}
        remaining_attribute_names=[i for i in attribute_names if i!=best_attr]

        for attr_val, data_subset in df.groupby(best_attr):
            subtree=id3(data_subset, target_attribute, remaining_attribute_names,default_class)
            tree[best_attr][attr_val]=subtree

        return tree

df=pd.read_csv("Deskpot/p-tennis.csv")

attribute_names=df.columns.tolist()
print("List of attribut name")

attribute_names.remove("PlayTennis")

for colname in df.select_dtypes("object"):
    df[colname], _ = df[colname].factorize()

print(df)
```

```
tree= id3(df,"PlayTennis", attribute_names)
print("The tree structure")
pprint(tree)
```

p-tennis.csv

Outlook	Temperat	Humidity	Windy	PlayTennis		
Sunny	Hot	High	FALSE	No		
Sunny	Hot	High	TRUE	No		
Overcast	Hot	High	FALSE	Yes		
Rainy	Mild	High	FALSE	Yes		
Rainy	Cool	Normal	FALSE	Yes		
Rainy	Cool	Normal	TRUE	No		
Overcast	Cool	Normal	TRUE	Yes		
Sunny	Mild	High	FALSE	No		
Sunny	Cool	Normal	FALSE	Yes		

OUTPUT:

List of attribut name

	Outlook	Temperature	Humidity	Windy	PlayTennis
0	0	0	0	False	0
1	0	0	0	True	0
2	1	0	0	False	1
3	2	1	0	False	1
4	2	2	1	False	1
5	2	2	1	True	0
6	1	2	1	True	1
7	0	1	0	False	0
8	0	2	1	False	1
9	2	1	1	False	1
10	0	1	1	True	1
11	1	1	0	True	1
12	1	0	1	False	1
13	2	1	0	True	0

The tree structure

```
{'Outlook': {0: {'Humidity': {0: 0, 1: 1}},
        1: 1,
        2: {'Windy': {False: 1, True: 0}}}}
```

Lab 5: Build an Artificial Neural Network by implementing the Backpropagation 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)
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) * lr
    # 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:
 [[0.89613503]
 [0.87647952]
 [0.89699621]]
'''
```

Lab 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. """

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

# Load Data from CSV
data = pd.read_csv('Desktop/p-tennis.csv')
print("The first 5 Values of data is :\n", data.head())

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

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

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

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

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

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

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

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

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

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

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

'Desktop/p-tennis.csv'

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak
5	No	Rain	Cool	Normal	Strong
6	Yes	Overcast	Cool	Normal	Strong
7	No	Sunny	Mild	High	Weak
8	Yes	Sunny	Cool	Normal	Weak
9	Yes	Rain	Mild	Normal	Weak
10	Yes	Sunny	Mild	Normal	Strong
11	Yes	Overcast	Mild	High	Strong
12	Yes	Overcast	Hot	Normal	Weak
13	No	Rain	Mild	High	Strong

OUTPUT

```
The first 5 Values of data is :
    Outlook Temperature Humidity  Windy PlayTennis
0    Sunny       Hot     High  False       No
1    Sunny       Hot     High   True       No
2  Overcast       Hot     High  False       Yes
3    Rainy       Mild     High  False      Yes
4    Rainy       Cool   Normal  False       Yes

The First 5 values of the train data is
    Outlook Temperature Humidity  Windy
0    Sunny       Hot     High  False
1    Sunny       Hot     High   True
2  Overcast       Hot     High  False
3    Rainy       Mild     High  False
4    Rainy       Cool   Normal  False

The First 5 values of train output is
 0     No
1     No
2    Yes
```

3   Yes
4   Yes
Name: PlayTennis, dtype: object

Now the Train output is

	Outlook	Temperature	Humidity	Windy
0	2	1	0	0
1	2	1	0	1
2	0	1	0	0
3	1	2	0	0
4	1	0	1	0

Now the Train output is
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 0.6666666666666666

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

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

# import some data to play with
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']

# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to

# # Visualise the clustering results
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features
plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Plot the Models Classifications
plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# General EM for GMM
```

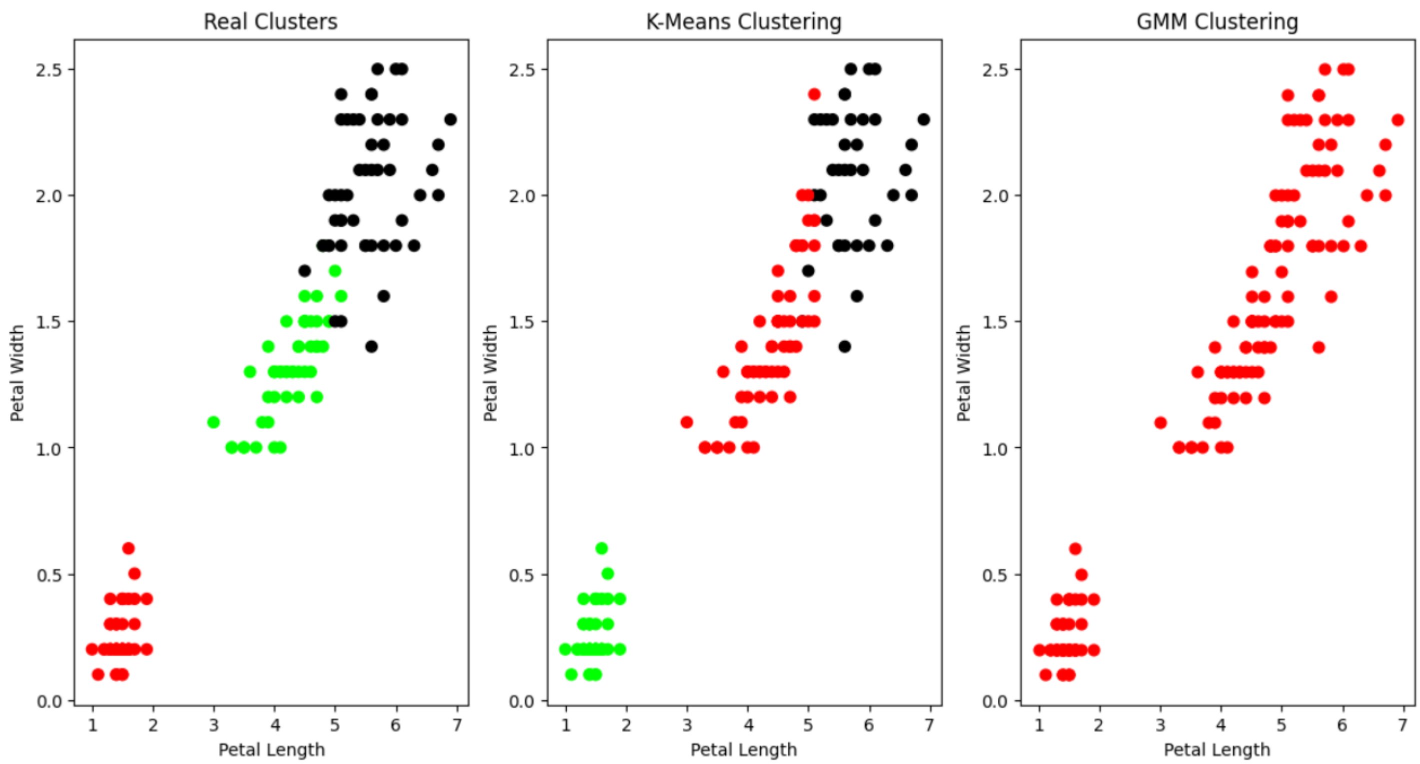
```
from sklearn import preprocessing

# transform your data such that its distribution will have a # mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=40)
gmm.fit(xs)
plt.subplot(1, 3, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[0], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')
```

OUTPUT

```
Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.
```



Lab 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.

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

   print("Classification Accuracy :" , classifier.score(x_test,y_test));
```

OUTPUT

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

```
Classification Accuracy : 0.9333333333333333
 Sample: [4.8 3.4 1.9 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.9 3.1 5.4 2.1]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333
 Sample: [5.6 3.  4.1 1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [4.7 3.2 1.6 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.3 2.3 4.4 1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [5.1 3.4 1.5 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.  2.9 4.5 1.5]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [5.4 3.9 1.3 0.4]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
```

LAB 9 : """Implement the non-parametric Locally Weighted Regression algorithm in order to fit data point's .Select appropriate data set for your experiment and draw graphs """

```
from math import ceil
import numpy as np
from scipy import linalg


def lowess(x, y, f, iterations):
    n = len(x)
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:, i]
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
            A = np.array([[np.sum(weights), np.sum(weights * x)], [np.sum(weights * x), np.sum(weights * x * x)]])
            beta = linalg.solve(A, b)
            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta ** 2) ** 2

    return yest


import math

n = 100
x = np.linspace(0, 2 * math.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f = 0.25
iterations = 3
yest = lowess(x, y, f, iterations)
```

```
import matplotlib.pyplot as plt

plt.plot(x, y, "r.")
plt.show()
plt.plot(x, yest, "b-")
plt.show()
```

<u>OUTPUT</u>