# ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)

## PROGRAM 01

**1. Implement A* Search Algorithm**

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}              #store distance from starting node
    parents = {}        # parents contains an adjacency map of all nodes
    #distance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        #node with lowest f() is found
        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):
                #nodes 'm' not in first and last set are added to first
                #n is set its parent
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                #for each node m,compare its distance from start i.e g(m) to the
```

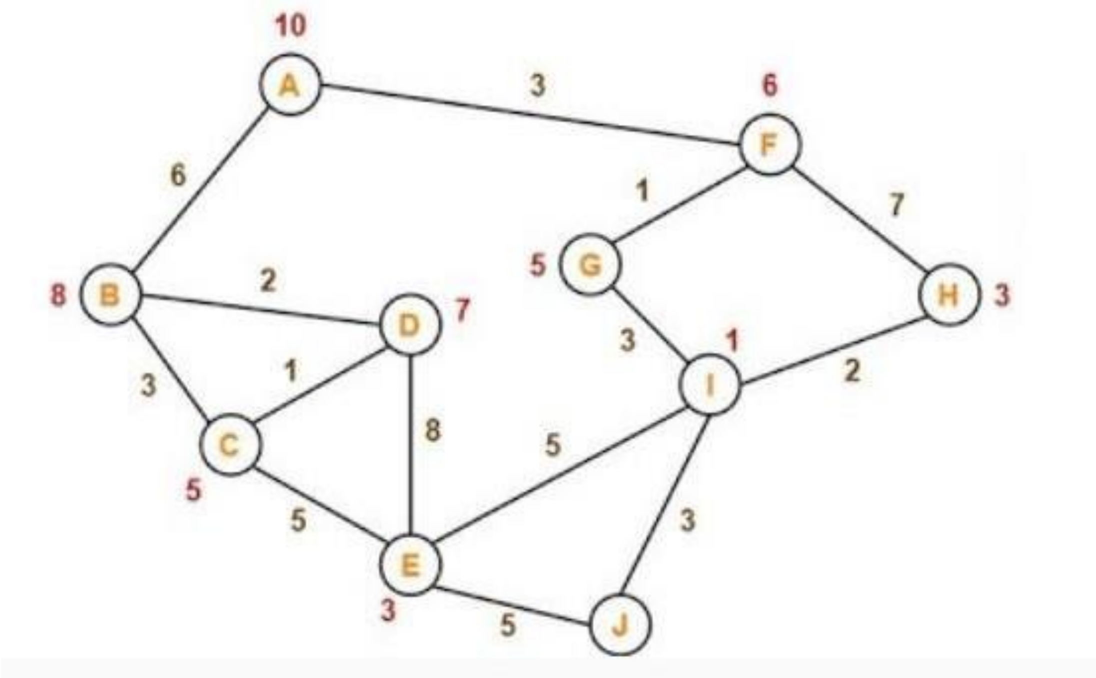
```
            #from start through n node
            else:
                if g[m] > g[n] + weight:
                    #update g(m)
                    g[m] = g[n] + weight
                    #change parent of m to n
                    parents[m] = n
                    #if m in closed set,remove and add to open
                    if m in closed_set:
                        closed_set.remove(m)
                        open_set.add(m)
    if n == None:
        print('Path does not exist!')
        return None
    # if the current node is the stop_node
    # then we begin reconstructin the path from it to the start_node
    if n == stop_node:
        path = []
        while parents[n] != n:
            path.append(n)
            n = parents[n]
        path.append(start_node)
        path.reverse()
        print('Path found: {}'.format(path))
        return path
    # remove n from the open_list, and add it to closed_list
    # because all of his neighbors were inspected
    open_set.remove(n)
    closed_set.add(n)
print('Path does not exist!')
```

```
    return None
#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]
#Describe your graph here
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('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')
```

**Output:**

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

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



# ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)

## PROGRAM 02

### 2. Implement AO* Search 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
```

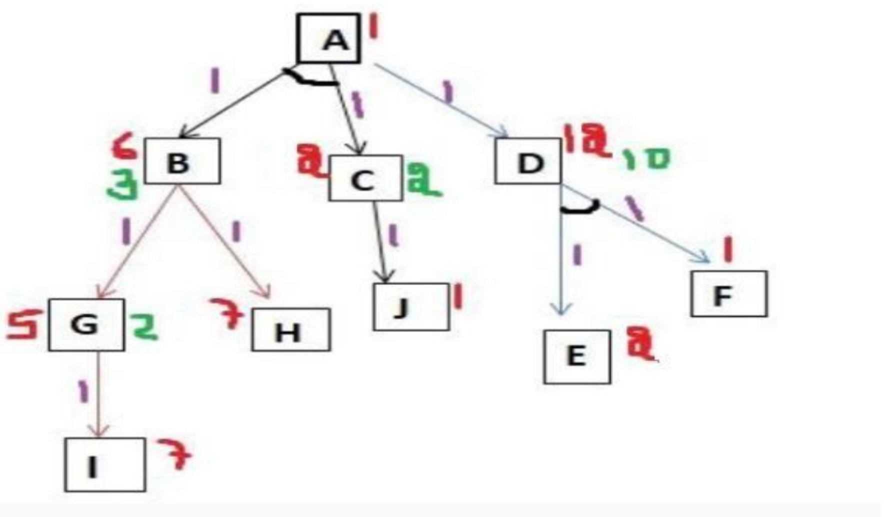
```
    def aoStar(self, v, backTracking): # AO* algorithm for a start node and backTracking
status flag
        print("HEURISTIC VALUES :", self.H)
        print("SOLUTION GRAPH :", self.solutionGraph)
        print("PROCESSING NODE :", v)
        print("-----------------------------------------------------------------------------------------")
        if self.getStatus(v) >= 0: # if status node v >= 0, compute Minimum Cost nodes of v
            minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
            print(minimumCost, childNodeList)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))
            solved=True # check the Minimum Cost nodes of v are solved
            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False
            if solved==True: # if the Minimum Cost nodes of v are solved, set the current node
status as solved(-1)
                self.setStatus(v,-1)
                self.solutionGraph[v]=childNodeList # update the solution graph with the solved
nodes which may be a part of solution
            if v!=self.start: # check the current node is the start node for backtracking the current
node value
                self.aoStar(self.parent[v], True) # backtracking the current node value with
backtracking status set to true
            if backTracking==False: # check the current call is not for backtracking
                for childNode in childNodeList: # for each Minimum Cost child node
                    self.setStatus(childNode,0) # set the status of child node to 0(needs exploration)
                    self.aoStar(childNode, False) # Minimum Cost child node is further explored
with backtracking status as false
#for simplicity we ll consider heuristic distances given
```

```python
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
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------
7 ['D']
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G
': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : D
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------
3 ['E', 'F']
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 2, 'D': 3, 'E': 2, 'F': 1, 'G'
: 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------
4 ['D']
HEURISTIC VALUES : {'A': 4, 'B': 6, 'C': 2, 'D': 3, 'E': 2, 'F': 1, 'G'
: 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : E
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------
0 []
HEURISTIC VALUES : {'A': 4, 'B': 6, 'C': 2, 'D': 3, 'E': 0, 'F': 1, 'G'
: 5, 'H': 7, 'I': 7, 'J': 1}
```

```
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : D
-----------------------------------------------------------------------------------------------------------------------------

1 ['E', 'F']
HEURISTIC VALUES : {'A': 4, 'B': 6, 'C': 2, 'D': 1, 'E': 0, 'F': 1, 'G'
: 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------------------------------------------

2 ['D']
HEURISTIC VALUES : {'A': 2, 'B': 6, 'C': 2, 'D': 1, 'E': 0, 'F': 1, 'G'
: 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : F
-----------------------------------------------------------------------------------------------------------------------------

0 []
HEURISTIC VALUES : {'A': 2, 'B': 6, 'C': 2, 'D': 1, 'E': 0, 'F': 0, 'G'
: 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {'E': [], 'F': []}
PROCESSING NODE : D
-----------------------------------------------------------------------------------------------------------------------------
1 ['E', 'F']
HEURISTIC VALUES : {'A': 2, 'B': 6, 'C': 2, 'D': 1, 'E': 0, 'F': 0, 'G'
: 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------------------------------------------
2 ['D']
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
------------------------------------------------------------------------------------------
```



In [22]:

```
import numpy as np
import pandas as pd
# Loading Data from a CSV File
data = pd.DataFrame(data = pd.read_csv("Training1.csv"))

# Separating concept features from Target
concepts  = np.array(data.iloc[:,0:-1])

target = np.array(data.iloc[:,-1])

def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h
# The learning iterations
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "No":
            for x in range(len(specific_h)):
                print(f"specific={specific_h[x]}")
    # For negative hypothesis change values only in G
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                    #print(f"general{x}={general_h[x][x]}")
                else:
                    general_h[x][x] = '?'
# find indices where we have empty rows, meaning those that are unchanged
    indices = [i for i,val in enumerate(general_h) if val == ['?', '?', '?', '?', '?
    for i in indices:
        # remove those rows from general_h
        general_h.remove(['?', '?', '?', '?', '?', '?'])
# Return final values
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
s_final
g_final
```

```
specific=Sunny
specific=Warm
specific=?
specific=Strong
specific=Warm
specific=Same
```

Out[22]:

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

In [ ]:

# ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)

## PROGRAM 04

**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.**

```
# import libraries

import numpy as np

import pandas as pd

#load dataset

from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()

data.data

data.feature_names

data.target

data.target_names

#create dataframe

df = pd.DataFrame(np.c_[data.data, data.target],
columns=[list(data.feature_names)+['target']])

df.head()

df.tail()

row1=df.iloc[3]

row1

df.shape

#Split Data

X = df.iloc[:, 0:-1]

y = df.iloc[:, -1]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)

print('Shape of X_train = ', X_train.shape)

print('Shape of y_train = ', y_train.shape)

print('Shape of X_test = ', X_test.shape)
```

```
print('Shape of y_test = ', y_test.shape)
#Train Decision Tree Classification Model
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini')
classifier.fit(X_train, y_train)
classifier.score(X_test, y_test)
classifier_entropy = DecisionTreeClassifier(criterion='entropy')
classifier_entropy.fit(X_train, y_train)
classifier_entropy.score(X_test, y_test) #Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_sc = sc.transform(X_train)
X_test_sc = sc.transform(X_test)
classifier_sc = DecisionTreeClassifier(criterion='gini')
classifier_sc.fit(X_train_sc, y_train)
classifier_sc.score(X_test_sc, y_test)
#Predict Cancer
patient1 = [17.99,
0.38,
122.8,
1001.0,
0.1184,
0.2776,
0.3001,
0.1471,
0.2419,
0.07871,
1.095,
0.9053,
```

```
8.589,
153.4,
0.006399,
0.04904,
0.05373,
0.01587,
0.03003,
0.006193,
25.38,
17.33,
184.6,
2019.0,
0.1622,
0.6656,
0.7119,
0.2654,
0.4601,
0.1189]
patient1 = np.array([patient1])
patient1
classifier.predict(patient1)
data.target_names
pred = classifier.predict(patient1)
if pred[0] == 0:
    print('Patient has Cancer (malignant tumor)')
else:
    print('Patient has no Cancer (malignant benign)')
```

**Output:**

```
Shape of X_train =  (455, 30)
Shape of y_train =  (455,)
Shape of X_test =  (114, 30)
Shape of y_test =  (114,)
Patient has Cancer (malignant tumor)
```

## ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)

### PROGRAM 04

**4. 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=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)#how much hidden layer wts contributed to
error
    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) *lr # dotproduct of nextlayererror and currentlayerop
    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:
 [[0.87134038]
 [0.85505899]
 [0.86918134]]
---------------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:
 [[0.87152832]
 [0.85524601]
 [0.86936978]]
---------------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:
 [[0.8717144 ]
 [0.85543121]
 [0.86955637]]
---------------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:
 [[0.87189864]
 [0.8556146 ]
 [0.86974111]]
---------------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:
 [[0.87208107]
 [0.85579622]
 [0.86992403]]
---------------Epoch- 5 Ends--------------

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.87208107]
 [0.85579622]
 [0.86992403]]
```

## Training Examples:

| Example | Sleep | Study | Expected % in Exams |
|---|---|---|---|
| **1** | 2 | 9 | 92 |
| **2** | 1 | 5 | 86 |
| **3** | 3 | 6 | 89 |

## Normalize the input

| Example | Sleep | Study | Expected % in Exams |
|---|---|---|---|
| **1** | 2/3 = 0.66666667 | 9/9 = 1 | 0.92 |
| **2** | 1/3 = 0.33333333 | 5/9 = 0.55555556 | 0.86 |
| **3** | 3/3 = 1 | 6/9 = 0.66666667 | 0.89 |

## ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)

### PROGRAM 06

**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 libarities

import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.naive_bayes import GaussianNB


# load data from CSV

data = pd.read_csv('tennisdata.csv')

print("THe first 5 values of data is :\n",data.head())


# obtain Train data and Train output

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

print("\nThe First 5 values of train data is\n",X.head())


y = data.iloc[:,-1]

print("\nThe first 5 values of Train output is\n",y.head())


# Convert then 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 data 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))
```

**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 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 data 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
```

**Dataset:**

**3**

| 1 | Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|---|
| 2 | Sunny | Hot | High | False | No |
| 3 | Sunny | Hot | High | True | No |
| 4 | Overcast | Hot | High | False | Yes |
| 5 | Rainy | Mild | High | False | Yes |
| 6 | Rainy | Cool | Normal | False | Yes |
| 7 | Rainy | Cool | Normal | True | No |
| 8 | Overcast | Cool | Normal | True | Yes |
| 9 | Sunny | Mild | High | False | No |
| 10 | Sunny | Cool | Normal | False | Yes |
| 11 | Rainy | Mild | Normal | False | Yes |
| 12 | Sunny | Mild | Normal | True | Yes |
| 13 | Overcast | Mild | High | True | Yes |
| 14 | Overcast | Hot | Normal | False | Yes |
| 15 | Rainy | Mild | High | True | No |

## ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)

### PROGRAM 08

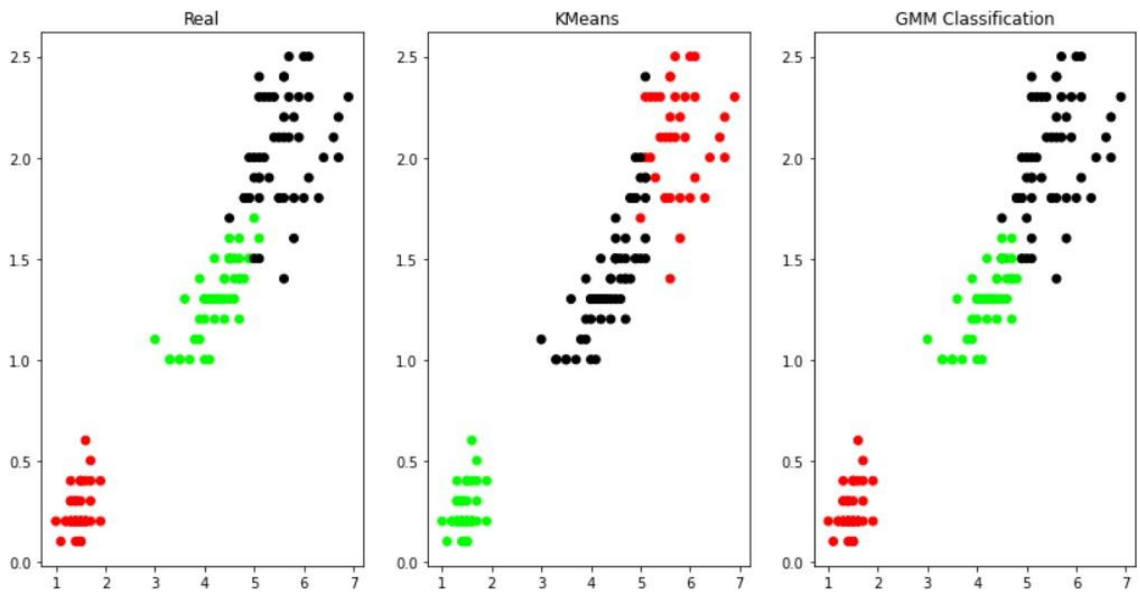
**8. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same dataset for clustering using k-means algorithm. Compute the results of these two algorithms and comment on the quality of clustering. You can add Java/ Python ML library classes/API in the program.**

```
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("8-dataset.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))
```

**Output:**

```
The accuracy score of K-Mean: 0.09333333333333334
The Confusion matrixof K-Mean:
 [[ 0 50  0]
 [ 2  0 48]
 [36  0 14]]
The accuracy score of EM: 0.9666666666666667
The Confusion matrix of EM:
  [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```



Real
KMeans
GMM Classification

## ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (18CSL76)

### PROGRAM 08

**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.**

```
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']

# Read dataset to pandas dataframe

dataset = pd.read_csv("9-dataset.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))
```

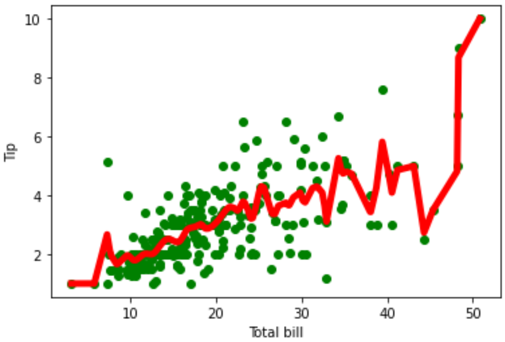
**Output:**

```
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-virginica            Iris-virginica            Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
```

```
Iris-setosa              Iris-setosa              Correct
Iris-virginica           Iris-virginica           Correct
Iris-versicolor          Iris-versicolor          Correct
Iris-setosa              Iris-setosa              Correct
Iris-virginica           Iris-virginica           Correct
Iris-setosa              Iris-setosa              Correct
Iris-versicolor          Iris-versicolor          Correct
Iris-setosa              Iris-setosa              Correct
Iris-versicolor          Iris-versicolor          Correct
Iris-virginica           Iris-virginica           Correct
-----------------------------------------------------------------------------------------------
--

Confusion Matrix:
 [[6 0 0]
 [0 5 0]
 [0 0 4]]
-----------------------------------------------------------------------------------------------
--

Classification Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         6
Iris-versicolor       1.00      1.00      1.00         5
 Iris-virginica       1.00      1.00      1.00         4

       accuracy                           1.00        15
      macro avg       1.00      1.00      1.00        15
   weighted avg       1.00      1.00      1.00        15

-----------------------------------------------------------------------------------------------
--
Accuracy of the classifer is 1.00
-----------------------------------------------------------------------------------------------
--
```

In [7]:

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

# load data points
data = pd.read_csv('tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
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))

#set k here
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();
```




In [ ]:

```
```