AIML LAB MANUAL

Program – 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] == 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': 2,
            'B': 6,
            'C': 2,
            'D': 3,
            'S': 4,
            'G': 0,
            }
    return H_dist[n]


Graph_nodes = {
        'A': [('B', 3), ('C', 1)],
        'B': [('D', 3)],
        'C': [('D', 1), ('G', 5)],
        'D': [('G', 3)],
        'S': [('A', 1)],
        'G': []
        }
aStarAlgo('S', 'G')
aStarAlgo('A', 'B')
aStarAlgo('B', 'S')
```

Output:

```
Path found: ['S', 'A', 'C', 'D', 'G']
Path found: ['A', 'B']
Path does not exist!
```

Program – 2

Implement AO* Search algorithm

```
class Graph:

    def __init__(self, graph, heuristicNodeList, startNode):

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

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

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

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

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

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

    def setHeuristicNodeValue(self, n, value):
        self.H[n] = value

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

print("==============================================================")
        print(self.solutionGraph)

print("==============================================================")


    def computeMinimumCostChildNodes(self, v):
```

```
    minimumCost = 0
    costToChildNodeListDict = {}
    costToChildNodeListDict[minimumCost] = []
    flag = True

    for nodeInfoTupleList in self.getNeighbors(v):
        cost = 0
        nodeList = []

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

        if flag == True:
            minimumCost = cost
            costToChildNodeListDict[minimumCost] = nodeList
            flag = False
        else:
            if minimumCost > cost:
                minimumCost = cost
                costToChildNodeListDict[minimumCost] = nodeList

    return minimumCost, costToChildNodeListDict[minimumCost]


def aoStar(self, v, backTracking):

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

    if self.getStatus(v) >= 0:

        minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
        self.setHeuristicNodeValue(v, minimumCost)
        self.setStatus(v, len(childNodeList))
        solved = True

        for childNode in childNodeList:
            self.parent[childNode] = v
            if self.getStatus(childNode) != -1:
                solved = solved & False

    if solved == True:
        self.setStatus(v, -1)
        self.solutionGraph[v] = childNodeList
```

```
        if v != self.start:
            self.aoStar(self.parent[v], True)

        if backTracking == False:
            for childNode in childNodeList:
                self.setStatus(childNode, 0)
                self.aoStar(childNode, False)


h1 = {'A': 38, 'B': 17, 'C': 9, 'D': 27, 'E': 5, 'F': 10, 'G': 3,
      'H': 4,'I': 15, 'J': 10}

graph1 = {
        'A': [[('B', 1), ('C', 1)], [('D', 1)]],
        'B': [[('E', 1)], [('F', 1)]],
        'C': [[('G', 1)], [('H', 1)]],
        'D': [[('I', 1), ('J', 1)]]
        }

G1 = Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()

print("HEURISTIC VALUES :", G1.H)
print("SOLUTION GRAPH :", G1.solutionGraph)

print('status:', G1.status)
```

Output:

```
HEURISTIC VALUES : {'A': 38, 'B': 17, 'C': 9, 'D': 27, 'E': 5, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J':
10}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 28, 'B': 17, 'C': 9, 'D': 27, 'E': 5, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J':
10}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 28, 'B': 6, 'C': 9, 'D': 27, 'E': 5, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J':
10}
SOLUTION GRAPH : {}
PROCESSING NODE : A
```

```
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 17, 'B': 6, 'C': 9, 'D': 27, 'E': 5, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {}
PROCESSING NODE : E
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 17, 'B': 6, 'C': 9, 'D': 27, 'E': 0, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : B
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 17, 'B': 1, 'C': 9, 'D': 27, 'E': 0, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {'E': [], 'B': ['E']}
PROCESSING NODE : A
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 12, 'B': 1, 'C': 9, 'D': 27, 'E': 0, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {'E': [], 'B': ['E']}
PROCESSING NODE : C
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 12, 'B': 1, 'C': 4, 'D': 27, 'E': 0, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {'E': [], 'B': ['E']}
PROCESSING NODE : A
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 1, 'C': 4, 'D': 27, 'E': 0, 'F': 10, 'G': 3, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {'E': [], 'B': ['E']}
PROCESSING NODE : G
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 1, 'C': 4, 'D': 27, 'E': 0, 'F': 10, 'G': 0, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {'E': [], 'B': ['E'], 'G': []}
PROCESSING NODE : C
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 1, 'C': 1, 'D': 27, 'E': 0, 'F': 10, 'G': 0, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {'E': [], 'B': ['E'], 'G': [], 'C': ['G']}
PROCESSING NODE : A
-----------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
============================================================
{'E': [], 'B': ['E'], 'G': [], 'C': ['G'], 'A': ['B', 'C']}
============================================================
HEURISTIC VALUES : {'A': 4, 'B': 1, 'C': 1, 'D': 27, 'E': 0, 'F': 10, 'G': 0, 'H': 4, 'I': 15, 'J': 10}
SOLUTION GRAPH : {'E': [], 'B': ['E'], 'G': [], 'C': ['G'], 'A': ['B', 'C']}
```

status: {'A': -1, 'B': -1, 'E': -1, 'C': -1, 'G': -1}

Program – 3

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

```
import numpy as np
import pandas as pd

# Loading Data from a CSV File
data = pd.DataFrame(data=pd.read_csv('/Users/amithpradhaan/Documents/ARTIFICIAL
INTELLIGENCE AND MACHINE LEARNING/AIML LAB/pg3 dataset.csv'))
print(data)

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

# Isolating target into a separate DataFrame
# copying last column to target array
target = np.array(data.iloc[:,-1])
print(target)

def learn(concepts, target):

    '''
    learn() function implements the learning method of the Candidate elimination algorithm.
    Arguments:
        concepts - a data frame with all the features
        target - a data frame with corresponding output values
    '''

    # Initialise S0 with the first instance from concepts
    # .copy() makes sure a new list is created instead of just pointing to the same memory
location
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and general_h")
    print(specific_h)
    #h=["#" for i in range(0,5)]
    #print(h)
```

```
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    # The learning iterations
     for i, h in enumerate(concepts):

        # Checking if the hypothesis has a positive target
        if target[i] == "Yes":
            for x in range(len(specific_h)):

                # Change values in S & G only if values change
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        # Checking if the hypothesis has a positive target
        if target[i] == "No":
            for x in range(len(specific_h)):
                # For negative hyposthesis change values only  in G
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("\nSteps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)

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

Output:

```
    sky airtemp humidity    wind water forcast enjoysport
0  Sunny    Warm   Normal  Strong  Warm    Same        Yes
1  Sunny    Warm     High  Strong  Warm    Same        Yes
2  Rainy    Cold     High  Strong  Warm  Change         No
3  Sunny    Warm     High  Strong  Cool  Change        Yes
```

[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
['Yes' 'Yes' 'No' 'Yes']

Initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

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

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

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

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

Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

Program – 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 math
import pandas as pd
from pprint import pprint
from collections import Counter
def entropy(probs):
    return sum([-prob*math.log(prob,2) for prob in probs])
def entropy_list(a_list):
    cnt = Counter(x for x in a_list)
    num_instance = len(a_list)*1.0
    probs = [x/num_instance for x in cnt.values()]
    return entropy(probs)
def info_gain(df,split,target,trace=0):
    df_split = df.groupby(split)
    nobs = len(df.index)*1.0
    df_agg_ent = df_split.agg({ target:[entropy_list, lambda x: len(x)/nobs] })
    df_agg_ent.columns = ['Entropy','PropObserved']
    new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent["PropObserved"])
    old_entropy = entropy_list(df[target])
    return old_entropy - new_entropy
def id3(df,target,attribute_name,default_class = None):
    cnt = Counter(x for x in df[target])
    if len(cnt)==1:
        return next(iter(cnt))
    elif df.empty or (not attribute_name):
        return default_class
    else:
        default_class = max(cnt.keys())
        gains = [info_gain(df,attr,target) for attr in attribute_name]
        index_max = gains.index(max(gains))
        best_attr = attribute_name[index_max]
        tree = { best_attr:{ } }
        remaining_attr = [x for x in attribute_name if x!=best_attr]
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,target,remaining_attr,default_class)
            tree[best_attr][attr_val] = subtree
        return tree
```

```
df_tennis = pd.read_csv("/Users/amithpradhaan/Documents/ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING/AIML LAB/pg4 dataset.csv")
print(df_tennis)
attribute_names = list(df_tennis.columns)
attribute_names.remove('PlayTennis') #Remove the class attribute
tree = id3(df_tennis,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
pprint(tree)

df_tennis = pd.read_csv("/Users/amithpradhaan/Documents/ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING/AIML LAB/pg4 dataset(test).csv")
attribute_names = list(df_tennis.columns)
attribute_names.remove('PlayTennis') #Remove the class attribute
tree = id3(df_tennis,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision tree for sample data is :\n")
pprint(tree)
```

Output:

```
    outlook temperature humidity    wind PlayTennis
0     sunny        hot     high    weak         no
1     sunny        hot     high  strong         no
2  overcast        hot     high    weak        yes
3      rain       mild     high    weak        yes
4      rain       cool   normal    weak        yes
5      rain       cool   normal  strong         no
6  overcast       cool   normal  strong        yes
7     sunny       mild     high    weak         no
8     sunny       cool   normal    weak        yes
9      rain       mild   normal    weak        yes
10    sunny       mild   normal  strong        yes
11 overcast       mild     high  strong        Yes
12 overcast        hot   normal    weak        yes
13     rain       mild     high  strong         no


The Resultant Decision Tree is :

{'outlook': {'overcast': {'temperature': {'cool': 'yes',
                                          'hot': 'yes',
                                          'mild': 'Yes'}},
             'rain': {'wind': {'strong': 'no', 'weak': 'yes'}},
             'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}
```

The Resultant Decision tree for sample data is :

{'Outlook': {'overcast': 'yes', 'rain': 'no', 'sunny': 'yes'}}

Program – 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) # two inputs [sleep,study]
y = np.array(([92], [86], [89]), dtype=float) # one output [Expected % in Exams]
X = X/np.amax(X,axis=0)
print(X) # maximum of X array longitudinally
y = y/100
print(y)
#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=5000    #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)) #weight of the link
from input node to hidden node
bh=np.random.uniform(size=(1,hiddenlayer_neurons)) # bias of the link from input node to
hidden node
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons)) #weight of the link
from hidden node to output node
bout=np.random.uniform(size=(1,output_neurons)) #bias of the link from hidden node to
output node


#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

# dotproduct of nextlayererror and currentlayerop
wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hiddenlayer) *lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:

```
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
[[0.92]
 [0.86]
 [0.89]]
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.74179329]
 [0.72033487]
```

[0.73908861]]

Program – 6

Write a program to implement the naive 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 pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv("/Users/amithpradhaan/Documents/ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING/AIML LAB/diabetes.csv")
feature_col_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
predicted_class_names = ['Outcome']

X = df[feature_col_names].values # these are factors for the prediction
y = df[predicted_class_names].values # this is what we want to predict

#splitting the dataset into train and test data

xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)

print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)


# Training Naive Bayes (NB) classifier on training data.

clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[0,137,40,35,168,43.1,2.288,33]])

#printing Confusion matrix, accuracy, Precision and Recall

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
```

```
print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

Output:

```
the total number of Training Data : (514, 1)

 the total number of Test Data : (254, 1)

 Confusion matrix
[[136  22]
 [ 36  60]]

 Accuracy of the classifier is 0.7716535433070866

 The value of Precision 0.7317073170731707

 The value of Recall 0.625
Predicted Value for individual Test Data: [1]
```

Program – 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.

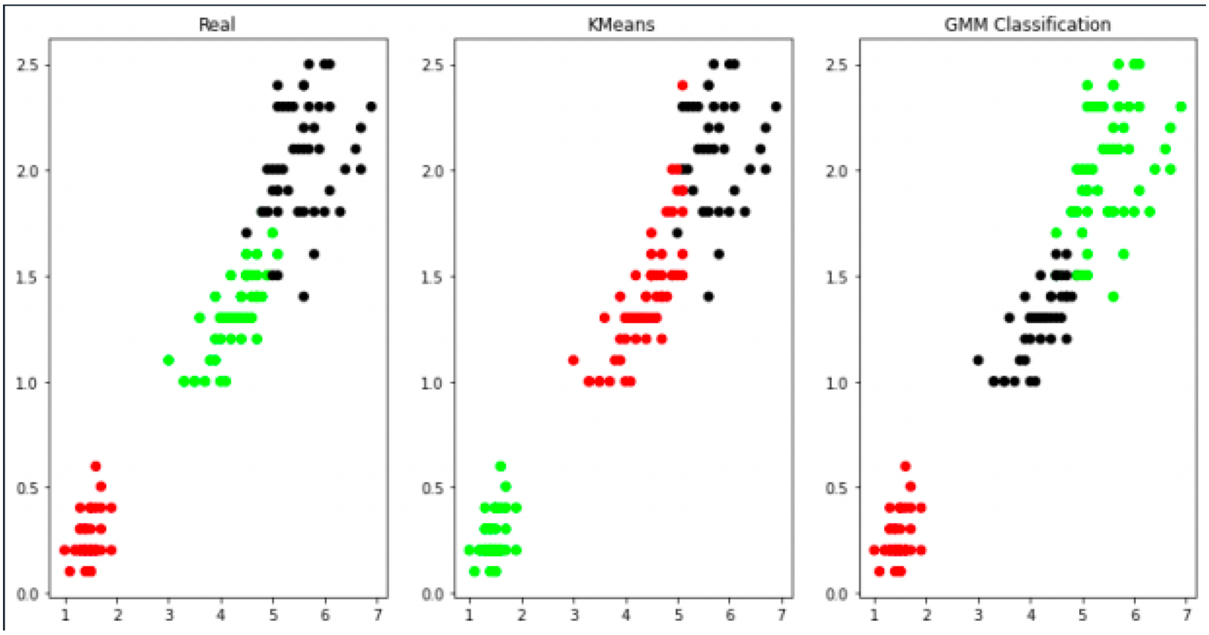
```
import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import sklearn.metrics as metrics

from sklearn.cluster import KMeans
```

```
from sklearn.mixture import GaussianMixture

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

dataset = pd.read_csv("/Users/amithpradhaan/Documents/ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING/AIML LAB/KMEANSdataset.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'])

plt.subplot(1, 3, 1)

plt.title('Real')

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

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 K-Mean: ', metrics.accuracy_score(y, model.labels_))

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

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
```

plt.show()

Output:

```
The accuracy score K-Mean:  0.24
The Confusion matrix 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]]
```



Program – 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 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('/Users/amithpradhaan/Documents/ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING/AIML LAB/KNNdataset.csv')

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

y = dataset.iloc[:, -1]

print('sepal-length', 'sepal-width', 'petal-length', 'petal-width')

print(X.head())

print('Target value')

print(y.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)

print("\n-------------------------------------------------------------------------")

print('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))

print("-------------------------------------------------------------------------")

i = 0
```

```
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("\classification Report:\n", metrics.classification_report(ytest, ypred))
print("------------------------------------------------------------------------")
print('Accuracy of the classifier is %0.2f' % metrics.accuracy_score(ytest, ypred))
print("------------------------------------------------------------------------")
```

Output:

```
Output from spyder call 'get_namespace_view':
sepal-length sepal-width petal-length petal-width
   5.1  3.5  1.4  0.2
0  4.9  3.0  1.4  0.2
1  4.7  3.2  1.3  0.2
2  4.6  3.1  1.5  0.2
3  5.0  3.6  1.4  0.2
4  5.4  3.9  1.7  0.4
Target value
0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
Name: Iris-setosa, dtype: object
```

```
-------------------------------------------------------------------
Original Label          Predicted Label          Correct/Wrong
---------------------------------------------------------------------
Iris-setosa             Iris-setosa             Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-setosa             Iris-setosa             Correct
Iris-virginica          Iris-virginica          Correct
Iris-virginica          Iris-virginica          Correct
Iris-virginica          Iris-virginica          Correct
Iris-virginica          Iris-virginica          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
---------------------------------------------------------------------

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

Classification Report:
                 precision    recall  f1-score   support

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

       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 classifier is 1.00
---------------------------------------------------------------------
```

Program – 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 numpy as np
import numpy as np1
import pandas as pd
import matplotlib.pyplot as plt


def kernel(point, xmat, k):
    m, n = np.shape(xmat)
    weights = np.mat(np1.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('/Users/amithpradhaan/Documents/ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING/AIML LAB/LWRdataset.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(np1.ones(m))
X = np.hstack((one.T, mbill.T))


ypred = localWeightRegression(X, mtip, 0.5)
SortIndex = X[:, 1].argsort(0)
xsort = X[SortIndex][:, 0]

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.scatter(bill, tip, color='green')
ax.plot(xsort[:, 1], ypred[SortIndex], color='red', linewidth=5)

plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

Output: