


||JAI SRI GURUDEV||

S J C INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LABORATORY MANUAL [18CSL76]

(VII SEM CSE-CBCS REVISED 2018 SCHEME)

S.J.C.INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CHICKBALLAPUR -562101

During the Year: 2023

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY
(Effective from the academic year 2018 -2019)

SEMESTER – VII **Course Code 18CSL76**
CIE Marks 40 **SEE Marks** 60
Number of Contact Hours/Week 0:0:2 **Total Number of Lab Contact Hours** 36
Exam Hours 03 **Credits – 2**

Course Learning Objectives: This course (18CSL76) will enable students to:
• Implement and evaluate AI and ML algorithms in and Python programming language.
Descriptions (if any):
Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:
1. Implement A* Search algorithm.
2. Implement AO* Search algorithm.
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.
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.
5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the Same using appropriate data sets.
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.
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.
8. Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print Both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
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

Laboratory Course Outcomes: The student should be able to:
• Implement and demonstrate AI and ML algorithms.
• Evaluate different algorithms.

Conduct of Practical Examination:
- Experiment distribution for laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
- For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Coursed to change in accordance with university regulations)*
- For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
- For laboratories having PART A and PART B

i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks

ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

PROGRAM.NO.1

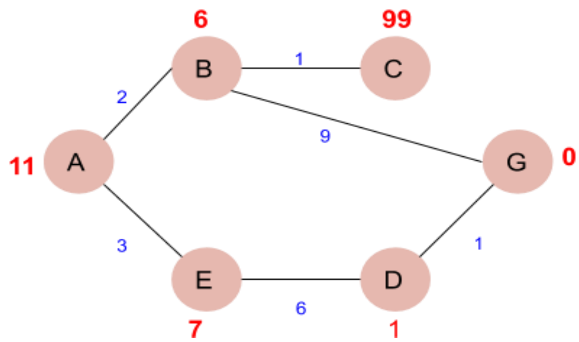
Implement A* Search algorithm.

```
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

        #ditance 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': 99,
            'D': 1,
            'E': 7,
            'G': 0,

        }

        return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],

}
aStarAlgo('A', 'G')
```

OUTPUT:-



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

PROGRAM.No.2(AO* Algorithm)

Implement AO* Search algorithm.

AO* Algorithm

AO* Algorithm basically based on problem decompositon (Breakdown problem into small pieces)
When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, **AND-OR graphs** or **AND - OR trees** are used for representing the solution.
The decomposition of the problem or problem reduction generates AND arcs.

AND-OR Graph

The figure shows an AND-OR graph

1. To pass any exam, we have two options, either cheating or hard work.
2. In this graph we are given two choices, first do cheating **or (The red line)** work hard and **(The arc)** pass.
3. When we have more than one choice and we have to pick one, we apply **OR condition** to choose one.(That's what we did here).
   - Basically the **ARC** here denote **AND condition**.
   - Here we have replicated the arc between the work hard and the pass because by doing the hard work possibility of passing an exam is more than cheating.

A* Vs AO*

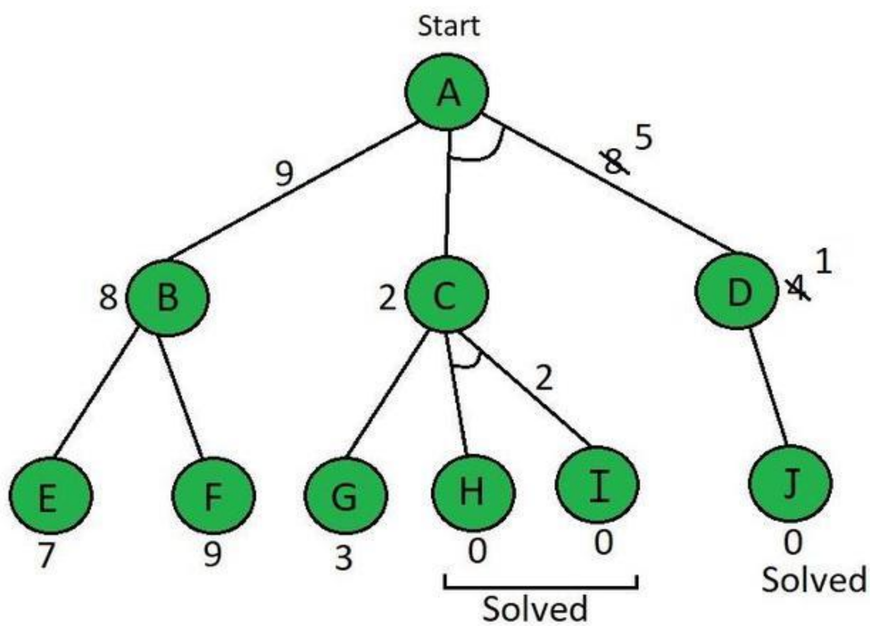
1. Both are part of informed search technique and use heuristic values to solve the problem.
2. The solution is guaranteed in both algorithm.
3. A* **always** gives an **optimal solution** (shortest path with low cost) But It is not guaranteed to that **AO*** always provide **an optimal solutions**.
4. **Reason:** Because AO* does not explore all the solution path once it got solution.

```
def Cost(H, condition, weight = 1):
    cost = {}
    if 'AND' in condition:
        AND_nodes = condition['AND']
        Path_A = ' AND '.join(AND_nodes)
        PathA = sum(H[node]+weight for node in AND_nodes)
        cost[Path_A] = PathA

    if 'OR' in condition:
        OR_nodes = condition['OR']
        Path_B =' OR '.join(OR_nodes)
        PathB = min(H[node]+weight for node in OR_nodes)
        cost[Path_B] = PathB
    return cost

# Update the cost
def update_cost(H, Conditions, weight=1):
    Main_nodes = list(Conditions.keys())
    Main_nodes.reverse()
    least_cost= {}
    for key in Main_nodes:
        condition = Conditions[key]
        print(key,':', Conditions[key],'>>>', Cost(H, condition, weight))
        c = Cost(H, condition, weight)
        H[key] = min(c.values())
        least_cost[key] = Cost(H, condition, weight)
    return least_cost

# Print the shortest path
def shortest_path(Start,Updated_cost, H):
    Path = Start
    if Start in Updated_cost.keys():
        Min_cost = min(Updated_cost[Start].values())
        key = list(Updated_cost[Start].keys())
        values = list(Updated_cost[Start].values())
        Index = values.index(Min_cost)

        # FIND MINIMIMUM PATH KEY
        Next = key[Index].split()
        # ADD TO PATH FOR OR PATH
        if len(Next) == 1:

            Start =Next[0]
            Path += '<--' +shortest_path(Start, Updated_cost, H)
        # ADD TO PATH FOR AND PATH
        else:
            Path +='<--('+key[Index]+') '

            Start = Next[0]
            Path += '[' +shortest_path(Start, Updated_cost, H) + ' + '

            Start = Next[-1]
            Path +=  shortest_path(Start, Updated_cost, H) + ']'

    return Path
```

```
H = {'A': -1, 'B': 5, 'C': 2, 'D': 4, 'E': 7, 'F': 9, 'G': 3, 'H': 0, 'I':0, 'J':0}

Conditions = {
 'A': {'OR': ['B'], 'AND': ['C', 'D']},
 'B': {'OR': ['E', 'F']},
 'C': {'OR': ['G'], 'AND': ['H', 'I']},
 'D': {'OR': ['J']}
}
# weight
weight = 1
# Updated cost
print('Updated Cost :')
Updated_cost = update_cost(H, Conditions, weight=1)
print('*'*75)
print('Shortest Path :\n',shortest_path('A', Updated_cost,H))
```

output:

```
Updated Cost :
D : {'OR': ['J']} >>> {'J': 1}
C : {'OR': ['G'], 'AND': ['H', 'I']} >>> {'H AND I': 2, 'G': 4}
B : {'OR': ['E', 'F']} >>> {'E OR F': 8}
A : {'OR': ['B'], 'AND': ['C', 'D']} >>> {'C AND D': 5, 'B': 9}
***************************************************************************
Shortest Path :
 A<--(C AND D) [C<--(H AND I) [H + I] + D<--J]
```



Program.No.3(Candidate Elimination Algorithm)

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
data = pd.read_csv('data.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")
```

Data Set: (Note:-Use data.csv file)

sky,airtemp,humidity,wind,water,forecast,Enjoysports
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

```
import numpy as np
import math
from data_loader import read_data


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

    # def __str__(self):
    # return self.attribute


def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])

    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1

        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict


def entropy(S):
    items = np.unique(S)
    if items.size == 1:
        return 0

    counts = np.zeros((items.shape[0], 1))
    sums = 0

    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums
```

```
def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))
    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0] / (total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy / iv


def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    gains = np.zeros((data.shape[1] - 1, 1))

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)

    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)

    items, dict = subtables(data, split, delete=True)

    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node


def empty(size):
    s = ""
    for x in range(size):
        s += "   "
    return s


def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return

    print(empty(level), node.attribute)

    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)
```

```
metadata, traindata = read_data("id3.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

data_loader [another supporting file]

```
import csv


def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)

        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)
```

Data Set (id3.csv):

Outlook	**Temperature**	**Humidity**	**Windy**	**PlayTennis**
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**
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**
Rain	**Mild**	**High**	**Strong**	**No**

OUTPUT

```
Outlook
   Overcast
b'Yes'
   Rainy
      Windy
b'FALSE'
b'Yes'
b'TRUE'
b'No'
   Sunny
      Humidity
b'High'
b'No'
b'Normal'
b'Yes'
```

5.Build an Artificial Neural Network by implementing the Back propagation 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)#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
  # 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:
 [[0.89429777]
 [0.87537484]
 [0.90042802]]
```

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.

```
print("\nNaive Bayes Classifier for concept learning problem")
import csv
import random
import math
import operator


def safe_div(x, y):
    if y == 0:
        return 0
    return x / y


def loadCsv(filename):
    lines = csv.reader(open(filename))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset


def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    copy = list(dataset)
    i = 0
    while len(trainSet) < trainSize:
        # index = random.randrange(len(copy))

        trainSet.append(copy.pop(i))
    return [trainSet, copy]


def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated


def mean(numbers):
    return safe_div(sum(numbers), float(len(numbers)))


def stdev(numbers):
    avg = mean(numbers)
    variance = safe_div(sum([pow(x - avg, 2) for x in numbers]), float(len(numbers) - 1))
    return math.sqrt(variance)


def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries


def summarizeByClass(dataset):
```

```
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries


def calculateProbability(x, mean, stdev):
    exponent = math.exp(-safe_div(math.pow(x - mean, 2), (2 * math.pow(stdev, 2))))
    final = safe_div(1, (math.sqrt(2 * math.pi) * stdev)) * exponent
    return final


def calculateClassProbabilities(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean, stdev)
    return probabilities


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


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


def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    accuracy = safe_div(correct, float(len(testSet))) * 100.0
    return accuracy


def main():
    filename = 'Naive.csv'
    splitRatio = 0.75
    dataset = loadCsv(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('Split {0} rows into'.format(len(dataset)))
    print('Number of Training data: ' + (repr(len(trainingSet))))
    print('Number of Test Data: ' + (repr(len(testSet))))
    print("\nThe values assumed for the concept learning attributes are\n")
    print(
        "OUTLOOK=> Sunny=1 Overcast=2 Rain=3\nTEMPERATURE=> Hot=1 Mild=2 Cool=3\nHUMIDITY=> 
High=1 Normal=2\nWIND=> Weak=1 Strong=2")
    print("TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5")
```

```
    print("\nThe Training set are:")
    for x in trainingSet:
        print(x)
    print("\nThe Test data set are:")
    for x in testSet:
        print(x)
    print("\n")
    # prepare model
    summaries = summarizeByClass(trainingSet)
    # test model
    predictions = getPredictions(summaries, testSet)
    actual = []
    for i in range(len(testSet)):
        vector = testSet[i]
        actual.append(vector[-1])
    # Since there are five attribute values, each attribute constitutes to 20% accuracy. So if
all attributes match with predictions then 100% accuracy
    print('Actual values: {0}%'.format(actual))
    print('Predictions: {0}%'.format(predictions))
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: {0}%'.format(accuracy))


main()
```

INPUT : dataset(Naive.csv)

1,1,1,1,5
1,1,1,2,5
2,1,1,2,10
3,2,1,1,10
3,3,2,1,10
3,3,2,2,5
2,3,2,2,10
1,2,1,1,5
1,3,2,1,10
3,2,2,2,10
1,2,2,2,10
2,2,1,2,10
2,1,2,1,10
3,2,1,2,5
1,2,1,2,10
1,2,1,2,5

OUTPUT

Naive Bayes Classifier for concept learning problem
Split 16 rows into
Number of Training data: 12

Number of Test Data: 4

The values assumed for the concept learning attributes are

OUTLOOK=> Sunny=1 Overcast=2 Rain=3
TEMPERATURE=> Hot=1 Mild=2 Cool=3
HUMIDITY=> High=1 Normal=2
WIND=> Weak=1 Strong=2
TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5

The Training set are:
[1.0, 1.0, 1.0, 1.0, 5.0]
[1.0, 1.0, 1.0, 2.0, 5.0]
[2.0, 1.0, 1.0, 2.0, 10.0]
[3.0, 2.0, 1.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 2.0, 5.0]
[2.0, 3.0, 2.0, 2.0, 10.0]
[1.0, 2.0, 1.0, 1.0, 5.0]
[1.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 2.0, 2.0, 10.0]
[1.0, 2.0, 2.0, 2.0, 10.0]
[2.0, 2.0, 1.0, 2.0, 10.0]

The Test data set are:
[2.0, 1.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 2.0, 5.0]
[1.0, 2.0, 1.0, 2.0, 10.0]
[1.0, 2.0, 1.0, 2.0, 5.0]

Actual values: [10.0, 5.0, 10.0, 5.0]%
Predictions: [5.0, 10.0, 5.0, 5.0]%
Accuracy: 25.0%

Process finished with exit code 0

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

```
from sklearn.cluster import KMeans

#from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv("kmeansdata.csv")
df1=pd.DataFrame(data)
print(df1)
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values

X=np.matrix(list(zip(f1,f2)))
plt.plot()
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('Speeding_Feature')
plt.xlabel('Distance_Feature')
plt.scatter(f1,f2)
plt.show()


# create new plot and data
plt.plot()
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# KMeans algorithm
#K = 3
kmeans_model = KMeans(n_clusters=3).fit(X)

plt.plot()
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')
    plt.xlim([0, 100])
    plt.ylim([0, 50])

plt.show()
```

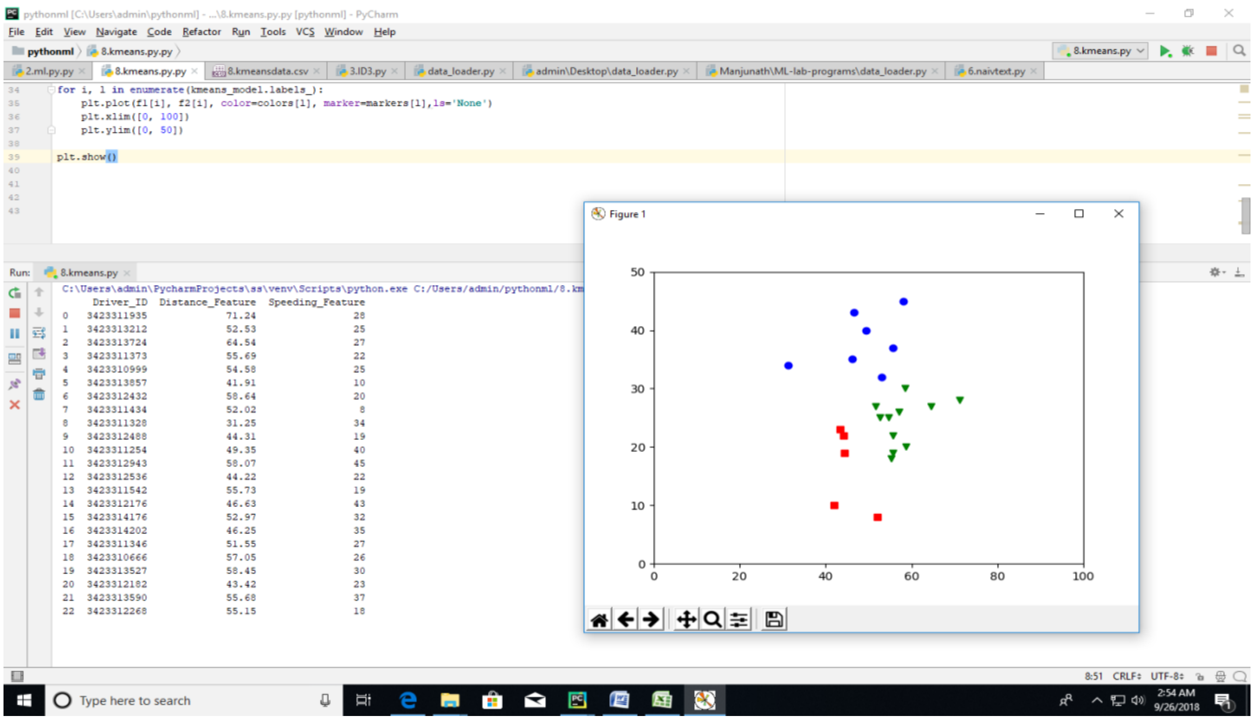
INPUT data set (`kmeansdata.csv`)

Driver_ID,Distance_Feature,Speeding_Feature
3423311935,71.24,28
3423313212,52.53,25
3423313724,64.54,27
3423311373,55.69,22
3423310999,54.58,25
3423313857,41.91,10
3423312432,58.64,20
3423311434,52.02,8
3423311328,31.25,34

3423312488,44.31,19
3423311254,49.35,40
3423312943,58.07,45
3423312536,44.22,22
3423311542,55.73,19
3423312176,46.63,43
3423314176,52.97,32
3423314202,46.25,35
3423311346,51.55,27
3423310666,57.05,26
3423313527,58.45,30
3423312182,43.42,23
3423313590,55.68,37
3423312268,55.15,18

OUTPUT

	Driver_ID	Distance_Feature	Speeding_Feature
0	3423311935	71.24	28
1	3423313212	52.53	25
2	3423313724	64.54	27
3	3423311373	55.69	22
4	3423310999	54.58	25
5	3423313857	41.91	10
6	3423312432	58.64	20
7	3423311434	52.02	8
8	3423311328	31.25	34
9	3423312488	44.31	19
10	3423311254	49.35	40
11	3423312943	58.07	45
12	3423312536	44.22	22
13	3423311542	55.73	19
14	3423312176	46.63	43
15	3423314176	52.97	32
16	3423314202	46.25	35
17	3423311346	51.55	27
18	3423310666	57.05	26
19	3423313527	58.45	30
20	3423312182	43.42	23
21	3423313590	55.68	37
22	3423312268	55.15	18



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.
KNN ALGORITHM

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target
print(iris_data)
print(iris_labels)
x_train, x_test, y_train, y_test=train_test_split(iris_data,iris_labels,test_size=0.30)

classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
print('confusion matrix is as follows')
print(confusion_matrix(y_test,y_pred))
print('Accuracy metrics')
print(classification_report(y_test,y_pred))
```

INPUT

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,
5.4,3.7,1.5,0.2,Iris-setosa,
4.8,3.4,1.6,0.2,Iris-setosa,
4.8,3,1.4,0.1,Iris-setosa,
4.3,3,1.1,0.1,Iris-setosa,
5.8,4,1.2,0.2,Iris-setosa,
5.7,4.4,1.5,0.4,Iris-setosa,
5.4,3.9,1.3,0.4,Iris-setosa,
5.1,3.5,1.4,0.3,Iris-setosa,
5.7,3.8,1.7,0.3,Iris-setosa,
5.1,3.8,1.5,0.3,Iris-setosa,
5.4,3.4,1.7,0.2,Iris-setosa,
5.1,3.7,1.5,0.4,Iris-setosa,
4.6,3.6,1,0.2,Iris-setosa,

5.1,**3.3**,**1.7**,**0.5**,**Iris-setosa**,
4.8,**3.4**,**1.9**,**0.2**,**Iris-setosa**,
5,**3**,**1.6**,**0.2**,**Iris-setosa**,
5,**3.4**,**1.6**,**0.4**,**Iris-setosa**,
5.2,**3.5**,**1.5**,**0.2**,**Iris-setosa**,
5.2,**3.4**,**1.4**,**0.2**,**Iris-setosa**,
4.7,**3.2**,**1.6**,**0.2**,**Iris-setosa**,
4.8,**3.1**,**1.6**,**0.2**,**Iris-setosa**,
5.4,**3.4**,**1.5**,**0.4**,**Iris-setosa**,
5.2,**4.1**,**1.5**,**0.1**,**Iris-setosa**,
5.5,**4.2**,**1.4**,**0.2**,**Iris-setosa**,
4.9,**3.1**,**1.5**,**0.1**,**Iris-setosa**,
5,**3.2**,**1.2**,**0.2**,**Iris-setosa**,
5.5,**3.5**,**1.3**,**0.2**,**Iris-setosa**,
4.9,**3.1**,**1.5**,**0.1**,**Iris-setosa**,
4.4,**3**,**1.3**,**0.2**,**Iris-setosa**,
5.1,**3.4**,**1.5**,**0.2**,**Iris-setosa**,
5,**3.5**,**1.3**,**0.3**,**Iris-setosa**,
4.5,**2.3**,**1.3**,**0.3**,**Iris-setosa**,
4.4,**3.2**,**1.3**,**0.2**,**Iris-setosa**,
5,**3.5**,**1.6**,**0.6**,**Iris-setosa**,
5.1,**3.8**,**1.9**,**0.4**,**Iris-setosa**,
4.8,**3**,**1.4**,**0.3**,**Iris-setosa**,
5.1,**3.8**,**1.6**,**0.2**,**Iris-setosa**,
4.6,**3.2**,**1.4**,**0.2**,**Iris-setosa**,
5.3,**3.7**,**1.5**,**0.2**,**Iris-setosa**,
5,**3.3**,**1.4**,**0.2**,**Iris-setosa**,
7,**3.2**,**4.7**,**1.4**,**Iris-versicolor**,
6.4,**3.2**,**4.5**,**1.5**,**Iris-versicolor**,
6.9,**3.1**,**4.9**,**1.5**,**Iris-versicolor**,
5.5,**2.3**,**4**,**1.3**,**Iris-versicolor**,
6.5,**2.8**,**4.6**,**1.5**,**Iris-versicolor**,
5.7,**2.8**,**4.5**,**1.3**,**Iris-versicolor**,
6.3,**3.3**,**4.7**,**1.6**,**Iris-versicolor**,
4.9,**2.4**,**3.3**,**1**,**Iris-versicolor**,
6.6,**2.9**,**4.6**,**1.3**,**Iris-versicolor**,
5.2,**2.7**,**3.9**,**1.4**,**Iris-versicolor**,
5,**2**,**3.5**,**1**,**Iris-versicolor**,
5.9,**3**,**4.2**,**1.5**,**Iris-versicolor**,
6,**2.2**,**4**,**1**,**Iris-versicolor**,
6.1,**2.9**,**4.7**,**1.4**,**Iris-versicolor**,
5.6,**2.9**,**3.6**,**1.3**,**Iris-versicolor**,
6.7,**3.1**,**4.4**,**1.4**,**Iris-versicolor**,
5.6,**3**,**4.5**,**1.5**,**Iris-versicolor**,
5.8,**2.7**,**4.1**,**1**,**Iris-versicolor**,
6.2,**2.2**,**4.5**,**1.5**,**Iris-versicolor**,
5.6,**2.5**,**3.9**,**1.1**,**Iris-versicolor**,

5.9,**3.2**,**4.8**,**1.8**,**Iris-versicolor**,
6.1,**2.8**,**4**,**1.3**,**Iris-versicolor**,
6.3,**2.5**,**4.9**,**1.5**,**Iris-versicolor**,
6.1,**2.8**,**4.7**,**1.2**,**Iris-versicolor**,
6.4,**2.9**,**4.3**,**1.3**,**Iris-versicolor**,
6.6,**3**,**4.4**,**1.4**,**Iris-versicolor**,
6.8,**2.8**,**4.8**,**1.4**,**Iris-versicolor**,
6.7,**3**,**5**,**1.7**,**Iris-versicolor**,
6,**2.9**,**4.5**,**1.5**,**Iris-versicolor**,
5.7,**2.6**,**3.5**,**1**,**Iris-versicolor**,
5.5,**2.4**,**3.8**,**1.1**,**Iris-versicolor**,
5.5,**2.4**,**3.7**,**1**,**Iris-versicolor**,
5.8,**2.7**,**3.9**,**1.2**,**Iris-versicolor**,
6,**2.7**,**5.1**,**1.6**,**Iris-versicolor**,
5.4,**3**,**4.5**,**1.5**,**Iris-versicolor**,
6,**3.4**,**4.5**,**1.6**,**Iris-versicolor**,
6.7,**3.1**,**4.7**,**1.5**,**Iris-versicolor**,
6.3,**2.3**,**4.4**,**1.3**,**Iris-versicolor**,
5.6,**3**,**4.1**,**1.3**,**Iris-versicolor**,
5.5,**2.5**,**4**,**1.3**,**Iris-versicolor**,
5.5,**2.6**,**4.4**,**1.2**,**Iris-versicolor**,
6.1,**3**,**4.6**,**1.4**,**Iris-versicolor**,
5.8,**2.6**,**4**,**1.2**,**Iris-versicolor**,
5,**2.3**,**3.3**,**1**,**Iris-versicolor**,
5.6,**2.7**,**4.2**,**1.3**,**Iris-versicolor**,
5.7,**3**,**4.2**,**1.2**,**Iris-versicolor**,
5.7,**2.9**,**4.2**,**1.3**,**Iris-versicolor**,
6.2,**2.9**,**4.3**,**1.3**,**Iris-versicolor**,
5.1,**2.5**,**3**,**1.1**,**Iris-versicolor**,
5.7,**2.8**,**4.1**,**1.3**,**Iris-versicolor**,
6.3,**3.3**,**6**,**2.5**,**Iris-virginica**,
5.8,**2.7**,**5.1**,**1.9**,**Iris-virginica**,
7.1,**3**,**5.9**,**2.1**,**Iris-virginica**,
6.3,**2.9**,**5.6**,**1.8**,**Iris-virginica**,
6.5,**3**,**5.8**,**2.2**,**Iris-virginica**,
7.6,**3**,**6.6**,**2.1**,**Iris-virginica**,
4.9,**2.5**,**4.5**,**1.7**,**Iris-virginica**,
7.3,**2.9**,**6.3**,**1.8**,**Iris-virginica**,
6.7,**2.5**,**5.8**,**1.8**,**Iris-virginica**,
7.2,**3.6**,**6.1**,**2.5**,**Iris-virginica**,
6.5,**3.2**,**5.1**,**2**,**Iris-virginica**,
6.4,**2.7**,**5.3**,**1.9**,**Iris-virginica**,
6.8,**3**,**5.5**,**2.1**,**Iris-virginica**,
5.7,**2.5**,**5**,**2**,**Iris-virginica**,
5.8,**2.8**,**5.1**,**2.4**,**Iris-virginica**,
6.4,**3.2**,**5.3**,**2.3**,**Iris-virginica**,
6.5,**3**,**5.5**,**1.8**,**Iris-virginica**,

7.7,**3.8**,**6.7**,**2.2**,**Iris-virginica**,
7.7,**2.6**,**6.9**,**2.3**,**Iris-virginica**,
6,**2.2**,**5**,**1.5**,**Iris-virginica**,
6.9,**3.2**,**5.7**,**2.3**,**Iris-virginica**,
5.6,**2.8**,**4.9**,**2**,**Iris-virginica**,
7.7,**2.8**,**6.7**,**2**,**Iris-virginica**,
6.3,**2.7**,**4.9**,**1.8**,**Iris-virginica**,
6.7,**3.3**,**5.7**,**2.1**,**Iris-virginica**,
7.2,**3.2**,**6**,**1.8**,**Iris-virginica**,
6.2,**2.8**,**4.8**,**1.8**,**Iris-virginica**,
6.1,**3**,**4.9**,**1.8**,**Iris-virginica**,
6.4,**2.8**,**5.6**,**2.1**,**Iris-virginica**,
7.2,**3**,**5.8**,**1.6**,**Iris-virginica**,
7.4,**2.8**,**6.1**,**1.9**,**Iris-virginica**,
7.9,**3.8**,**6.4**,**2**,**Iris-virginica**,
6.4,**2.8**,**5.6**,**2.2**,**Iris-virginica**,
6.3,**2.8**,**5.1**,**1.5**,**Iris-virginica**,
6.1,**2.6**,**5.6**,**1.4**,**Iris-virginica**,
7.7,**3**,**6.1**,**2.3**,**Iris-virginica**,
6.3,**3.4**,**5.6**,**2.4**,**Iris-virginica**,
6.4,**3.1**,**5.5**,**1.8**,**Iris-virginica**,
6,**3**,**4.8**,**1.8**,**Iris-virginica**,
6.9,**3.1**,**5.4**,**2.1**,**Iris-virginica**,
6.7,**3.1**,**5.6**,**2.4**,**Iris-virginica**,
6.9,**3.1**,**5.1**,**2.3**,**Iris-virginica**,
5.8,**2.7**,**5.1**,**1.9**,**Iris-virginica**,
6.8,**3.2**,**5.9**,**2.3**,**Iris-virginica**,
6.7,**3.3**,**5.7**,**2.5**,**Iris-virginica**,
6.7,**3**,**5.2**,**2.3**,**Iris-virginica**,
6.3,**2.5**,**5**,**1.9**,**Iris-virginica**,
6.5,**3**,**5.2**,**2**,**Iris-virginica**,
6.2,**3.4**,**5.4**,**2.3**,**Iris-virginica**,
5.9,**3**,**5.1**,**1.8**,**Iris-virginica**,

OUTPUT

```
 [[5.1 3.5 1.4 0.2]
[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
```

```
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
[4.8 3.  1.4 0.1]
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.1 1.5 0.1]
[4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
```

```
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
 [6.  2.2 4.  1. ]
 [6.1 2.9 4.7 1.4]
 [5.6 2.9 3.6 1.3]
 [6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
 [5.8 2.7 4.1 1. ]
 [6.2 2.2 4.5 1.5]
 [5.6 2.5 3.9 1.1]
 [5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
 [6.8 2.8 4.8 1.4]
[6.7 3.5.  1.7]
 [6.  2.9 4.5 1.5]
 [5.7 2.6 3.5 1. ]
 [5.5 2.4 3.8 1.1]
 [5.5 2.4 3.7 1. ]
 [5.8 2.7 3.9 1.2]
 [6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
 [6.  3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]
 [6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
 [5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
 [5.  2.3 3.3 1. ]
 [5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
 [5.7 2.9 4.2 1.3]
 [6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
 [5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
```

```
 [6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
[6.7 3.  5.2 2.3]
[6.3 2.5 5.  1.9]
[6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
[5.9 3.  5.1 1.8]]
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
confusion matrix is as follows
[[21  0  0]
[ 0 11  2]
[ 0  0 11]]
Accuracy metrics
precision    recall  f1-score   support

          0       1.00      1.00      1.00        21
          1       1.00      0.85      0.92        13
          2       0.85      1.00      0.92        11

avg / total       0.96      0.96      0.96        45
```

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.

LOCALLY WEIGHTED REGRESSION

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


def lowess(x, y, f=2./3., iter=3):
    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(iter):
        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

if __name__ == '__main__':
    import math
    n = 100
    x = np.linspace(0, 2 * math.pi, n)
    print("==========================values of x=====================")
    print(x)
    y = np.sin(x) + 0.3*np.random.randn(n)
    print("================================Values of y===================")
    print(y)
    f = 0.25
    yest = lowess(x, y, f=f, iter=3)

    import pylab as pl
    pl.clf()
    pl.plot(x, y, label='y noisy')
    pl.plot(x, yest, label='y pred')
    pl.legend()
    pl.show()
```

OUTPUT

```
==========================values of x=====================
[0.         0.06346652 0.12693304 0.19039955 0.25386607 0.31733259
 0.38079911 0.44426563 0.50773215 0.57119866 0.63466518 0.6981317
 0.76159822 0.82506474 0.88853126 0.95199777 1.01546429 1.07893081
 1.14239733 1.20586385 1.26933037 1.33279688 1.3962634  1.45972992
```

```
 1.52319644 1.58666296 1.65012947 1.71359599 1.77706251 1.84052903
 1.90399555 1.96746207 2.03092858 2.0943951  2.15786162 2.22132814
 2.28479466 2.34826118 2.41172769 2.47519421 2.53866073 2.60212725
 2.66559377 2.72906028 2.7925268  2.85599332 2.91945984 2.98292636
 3.04639288 3.10985939 3.17332591 3.23679243 3.30025895 3.36372547
 3.42719199 3.4906585  3.55412502 3.61759154 3.68105806 3.74452458
3.8079911  3.87145761 3.93492413 3.99839065 4.06185717 4.12532369
4.1887902  4.25225672 4.31572324 4.37918976 4.44265628 4.5061228
 4.56958931 4.63305583 4.69652235 4.75998887 4.82345539 4.88692191
 4.95038842 5.01385494 5.07732146 5.14078798 5.2042545  5.26772102
 5.33118753 5.39465405 5.45812057 5.52158709 5.58505361 5.64852012
 5.71198664 5.77545316 5.83891968 5.9023862  5.96585272 6.02931923
 6.09278575 6.15625227 6.21971879 6.28318531]
=================================Values of y===================
[ 0.32536008 -0.0080573   0.11946369  0.41612046  0.45098579  0.43815367
0.10801193  0.68389606  0.86074625  0.04549917  0.68505644  0.60342634
1.17247156  0.88083937  0.71119685  0.95001511  0.54481781  0.7051224
1.25351458  0.8712536   0.92022204  0.7352142   0.88698095  0.91535147
0.83840992  0.7904273   1.75713902  0.9658919   0.39042121  0.66715723
0.82248617  1.16770788  1.62890879  0.55892447  1.66198264  0.02503305
0.79764264  0.55443527  1.21535481  1.09842121  0.94842294  0.73174791
  0.684332    0.28964437  0.71744902  0.37907153  0.2530457   0.15897645
0.07088533  0.54206641  0.12110612  0.08384214  0.12731212 -0.53552899
  0.11736083 -0.56747834 -0.21437779 -0.53090037 -0.02105477 -0.7363005
 -0.43987103 -0.67372833 -0.38014677 -0.17410718 -0.67528673 -0.80375547
 -0.62601973 -0.74283758 -0.75248483 -0.67113581 -1.20706585 -0.64311434
 -1.59478696 -1.23125828 -0.8670961  -0.64860678 -0.9419199  -0.42584513
 -0.78040914 -1.10565932 -0.990609   -0.89934155 -0.60020463 -0.38534216
 -1.28563144 -0.71983964 -0.43870468 -1.03712938 -0.28325743 -0.63386377
 -0.49045503 -0.45722592 -0.0669703  -0.47006542 -0.44179404 -0.66259661
 -0.21934077 -0.51959973 -0.11584542  0.19354907]
```