K.N.S INSTITUTE OF TECHNOLOGY

Department Of Computer Science and Engineering

Tirumenahalli, Yelahanka, Bengaluru, Karnataka 560064

Sub-: AI & MI LAB MANUAL

SUBJECT CODE: 18CSL76

PREPARED BY :

Prof. Ateeq Ahmed

Department of Computer Science & Engineering

Bengaluru, Karnataka 560064

Lab Syllabus Programs :-

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 algorithmsand comment on the quality of clustering. You can add JAVA/Python Libarary classes for ML 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.
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.

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': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
```

```
        'J': 0
    }
    return H_dist[n]
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}
aStarAlgo('A', 'J')
```

OUTPUT:-

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

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)
            print(minimumCost, childNodeList)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))
            solved=True
            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False
```

```
        if solved==True:
            self.setStatus(v,-1)
            self.solutionGraph[v]=childNodeList
        if v!=self.start:
            self.aoStar(self.parent[v], True)
        if backTracking==False:
            for childNode in childNodeList:
                self.setStatus(childNode,0)
                self.aoStar(childNode, False)
                print ("Graph - 1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}
G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

OUTPUT:-

```
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-------------------------------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
```

----

6 ['G']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

10 ['B', 'C']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : G

----

8 ['I']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : B

----

8 ['H']

HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

12 ['B', 'C']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : I

----

0 []

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': []}

PROCESSING NODE : G

----

1 ['I']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I']}

PROCESSING NODE : B

----

2 ['G']

HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

----

6 ['B', 'C']

Graph - 1

Graph - 1

Graph - 1

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : C

----

2 ['J']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

----

6 ['B', 'C']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : J

----

0 []

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}

PROCESSING NODE : C

----

1 ['J']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

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

PROCESSING NODE : A

----

5 ['B', 'C']

Graph - 1

Graph - 1

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

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

----

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

***** Create Excel file Training_examples.csv and save it in same path**

Sky	Air	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Same	No
Sunny	Warm	High	Strong	Cool	Change	Yes

```
import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read_csv('Training_examples.csv'))

concepts = np.array(data.iloc[:,0:-1])

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

def learn(concepts, target):

  specific_h = concepts[0].copy()

  print("initialization of specific_h and general_h")

  print(specific_h)

  general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]

  print(general_h)

  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)):

         if h[x] != specific_h[x]:

          general_h[x][x] = specific_h[x]

         else:
```

```
                general_h[x][x] = '?'
                print(" steps of Candidate Elimination Algorithm",i+1)
                print(specific_h)
                print(general_h)
        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")
```

OUTPUT:-

```
initialization of specific_h and general_h
['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Steps of Candidate Elimination Algorithm 1
['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Steps of Candidate Elimination Algorithm 2
['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
Steps of Candidate Elimination Algorithm 3
['Sunny' 'Warm' 'High' 'Strong' '?' '?']
[['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.

***** Create Excel file 'playtennis.csv' and save it in same path**

Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

```
import pandas as pd

import numpy as np

dataset = pd.read_csv('playtennis.csv',names=['outlook','temperature','humidity','wind','class',])

attributes =('outlook','Temperature','Humidity','Wind','PlayTennis')

def entropy(target_col):

   elements,counts = np.unique(target_col,return_counts = True)

   entropy = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for
i in range(len(elements))])

   return entropy

def InfoGain(data,split_attribute_name,target_name="class"):

   total_entropy = entropy(data[target_name])

   vals,counts= np.unique(data[split_attribute_name],return_counts=True)

   Weighted_Entropy =
np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==vals[i]).dropn
a()[target_name]) for i in range(len(vals))])

   Information_Gain = total_entropy - Weighted_Entropy

   return Information_Gain

def ID3(data,originaldata,features,target_attribute_name="class",parent_node_class= None):
```

```
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data)==0:
        return
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attribute
_name],return_counts=True)[1])]
    elif len(features) ==0:
        return parent_node_class
    else:
        parent_node_class =
np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name],return
_counts=True)[1])]
    item_values = [InfoGain(data,feature,target_attribute_name) for feature in features]
    best_feature_index = np.argmax(item_values)
    best_feature = features[best_feature_index]
    tree = {best_feature:{}}
    features = [i for i in features if i != best_feature]
    for value in np.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()
            subtree =ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)
            tree[best_feature][value] = subtree
    return(tree)
def predict(query,tree,default = 1):
    for key in list(query.keys()):
        if key in list(tree.keys()):
          try:
            result = tree[key][query[key]]
          except:
            return default
          result = tree[key][query[key]]
          if isinstance(result,dict):
```

```
            return predict(query,result)
        else:
            return result
def train_test_split(dataset):
    training_data = dataset.iloc[:14].reset_index(drop=True)
    return training_data
def test(data,tree):
    queries = data.iloc[:,:-1].to_dict(orient = "records")
    predicted = pd.DataFrame(columns=["predicted"])
    for i in range(len(data)):
        predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)
    print('The prediction accuracy is: ',(np.sum(predicted["predicted"]==data["class"])/len(data))*100,'%')
XX = train_test_split(dataset)
training_data=XX
tree = ID3(training_data,training_data,training_data.columns[:-1])
print(' Display Tree',tree)
print('len=',len(training_data))
test(training_data,tree)
```

OUTPUT:-

Display Tree {'wind': {'Change': {'humidity': {'Cold': 'Yes', 'Warm': 'No'}}, 'Forecast': 'EnjoySport', 'Same': 'Yes'}}

len= 5

The prediction accuracy is: 100.0 %

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

```
from math import exp

from random import seed

from random import random

def initialize_network(n_inputs, n_hidden, n_outputs):

  network = list()

  hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]

  network.append(hidden_layer)

  output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]

  network.append(output_layer)

  return network

def activate(weights, inputs):

      activation = weights[-1]

      for i in range(len(weights)-1):

           activation += weights[i] * inputs[i]

      return activation


def transfer(activation):

  return 1.0 / (1.0 + exp(-activation))

def forward_propagate(network, row):

    inputs = row

    for layer in network:

          new_inputs = []

          for neuron in layer:

              activation = activate(neuron['weights'], inputs)

              neuron['output'] = transfer(activation)

              new_inputs.append(neuron['output'])

          inputs = new_inputs

    return inputs
```

```
def transfer_derivative(output):
    return output * (1.0 - output)
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
            layer = network[i]
            errors = list()
            if i != len(network)-1:
                    for j in range(len(layer)):
                            error = 0.0
                            for neuron in network[i + 1]:
                                error += (neuron['weights'][j] * neuron['delta'])
                                errors.append(error)
            else:
                for j in range(len(layer)):
                        neuron = layer[j]
                        errors.append(expected[j] - neuron['output'])
            for j in range(len(layer)):
                neuron = layer[j]
                neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
def update_weights(network, row, l_rate):
     for i in range(len(network)):
             inputs = row[:-1]
             if i != 0:
                    inputs = [neuron['output'] for neuron in network[i - 1]]
             for neuron in network[i]:
                 for j in range(len(inputs)):
                            neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
                 neuron['weights'][-1] += l_rate * neuron['delta']
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
```

```
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
seed(1)
dataset = [[2.7810836,2.550537003,0],
        [1.465489372,2.362125076,0],
        [3.396561688,4.400293529,0],
        [1.38807019,1.850220317,0],
        [3.06407232,3.005305973,0],
        [7.627531214,2.759262235,1],
        [5.332441248,2.088626775,1],
        [6.922596716,1.77106367,1],
        [8.675418651,-0.242068655,1],
        [7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

OUTPUT:-

>epoch=0, lrate=0.500, error=6.365

>epoch=1, lrate=0.500, error=5.557

>epoch=2, lrate=0.500, error=5.291

>epoch=3, lrate=0.500, error=5.262

>epoch=4, lrate=0.500, error=5.217

>epoch=5, lrate=0.500, error=4.899

>epoch=6, lrate=0.500, error=4.419

>epoch=7, lrate=0.500, error=3.900

>epoch=8, lrate=0.500, error=3.461

>epoch=9, lrate=0.500, error=3.087

>epoch=10, lrate=0.500, error=2.758

>epoch=11, lrate=0.500, error=2.468

>epoch=12, lrate=0.500, error=2.213

>epoch=13, lrate=0.500, error=1.989

>epoch=14, lrate=0.500, error=1.792

>epoch=15, lrate=0.500, error=1.621

>epoch=16, lrate=0.500, error=1.470

>epoch=17, lrate=0.500, error=1.339

>epoch=18, lrate=0.500, error=1.223

>epoch=19, lrate=0.500, error=1.122

[{'weights': [-0.9766426647918854, 1.0573043092399, 0.7999535671683315], 'output': 0.05429927062285241, 'delta': -0.0035328621774792703}, {'weights': [-1.2245133652927975, 1.4766900503308025, 0.7507113892487565], 'output': 0.03737569585208105, 'delta': -0.005989297622698788}]

[{'weights': [1.4965066037208181, 1.770264295168642, -1.28526000789383], 'output': 0.24698288711606625, 'delta': -0.04593445543099784}, {'weights': [-1.8260068779176126, -1.1775229580602165, 1.1610216434075609], 'output': 0.7292895947013409, 'delta': 0.05344534875231567}]

__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__

***** Create Excel file DBetes.csv and save it in same path**

6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	56	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	166	60	23	846	30.1	0.398	59	1
5	100	72	19	175	25.8	0.587	51	1
7	118	0	0	0	30	0.484	32	1
0	107	84	47	230	45.8	0.551	31	1
7	103	74	0	0	29.6	0.254	31	1
1	115	30	38	83	43.3	0.183	33	0
1	126	70	30	96	34.6	0.529	32	1

```
import csv
import random
import math
def loadCsv(filename):
    lines = csv.reader(open(filename, "rt"))
    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)
        while len(trainSet) < trainSize:
            index = random.randrange(len(copy))
            trainSet.append(copy.pop(index))
        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 sum(numbers)/float(len(numbers))
def stdev(numbers):
    avg = mean(numbers)
    variance = 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)
    print(len(separated))
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
```

```
    print(summaries)
    return summaries
def calculateProbability(x, mean, stdev):
        exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
        return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
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
    return (correct/float(len(testSet))) * 100.0
filename = 'DBetes.csv'
splitRatio = 0.70
dataset = loadCsv(filename)
trainingSet, testSet = splitDataset(dataset, splitRatio)
print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset),
len(trainingSet), len(testSet)))
summaries = summarizeByClass(trainingSet)
predictions = getPredictions(summaries, testSet)
accuracy = getAccuracy(testSet, predictions)
print('Accuracy: {0}%'.format(accuracy))
```

OUTPUT:

Split 20 rows into train=14 and test=6 rows

2

{1.0: [(3.7777777777777777, 3.632415786283895), (122.55555555555556, 30.23289231578378), (58.22222222222222, 25.699113689861843), (20.666666666666668, 17.334935823359714), (178.11111111111111, 264.3182004915877), (34.22222222222222, 6.752550958300458), (0.6528888888888889, 0.6258622940480686), (39.888888888888886, 11.794537341969422)], 0.0: [(5.2, 4.54972526643093), (112.8, 19.21457779916072), (53.6, 37.93151723830725), (13.4, 18.62256695517565), (16.6, 37.11872842649651), (33.98, 7.132811507393138), (0.45999999999999996, 0.5544384546547976), (36.0, 11.832159566199232)]}

Accuracy: 66.66666666666666%

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 algorithmsand comment on the quality of clustering. You can add JAVA/Python Libarary classes for ML program

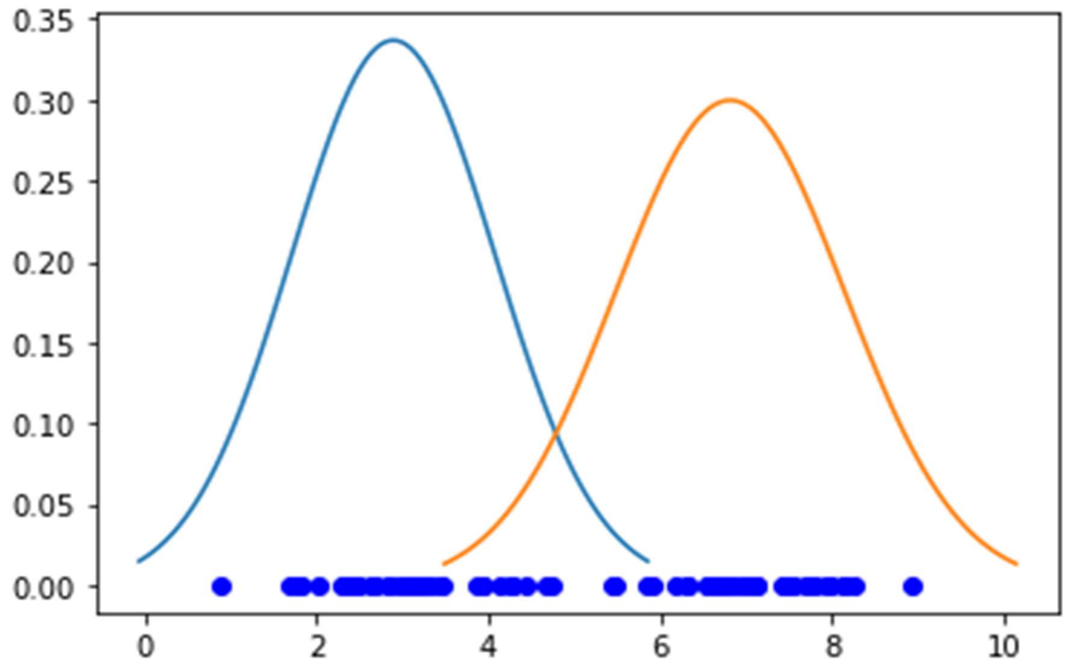
EM algorithm

```
import numpy as np

from scipy import stats

np.random.seed(110)

red_mean = 3

red_std = 0.8

blue_mean = 7

blue_std = 1

red = np.random.normal(red_mean, red_std, size=40)

blue = np.random.normal(blue_mean, blue_std, size=40)

both_colours = np.sort(np.concatenate((red, blue)))

red_mean_guess = 2.1

blue_mean_guess = 6

red_std_guess = 1.5

blue_std_guess = 0.8

for i in range(10):

  likelihood_of_red = stats.norm(red_mean_guess, red_std_guess).pdf(both_colours)

  likelihood_of_blue = stats.norm(blue_mean_guess,blue_std_guess).pdf(both_colours)

  likelihood_total = likelihood_of_red + likelihood_of_blue

  red_weight = likelihood_of_red / likelihood_total

  blue_weight = likelihood_of_blue / likelihood_total

def estimate_mean(data, weight):

   return np.sum(data * weight) / np.sum(weight)

def estimate_std(data, weight, mean):

   variance = np.sum(weight * (data - mean)**2) / np.sum(weight)
```

```
    return np.sqrt(variance)
blue_std_guess = estimate_std(both_colours, blue_weight, blue_mean_guess)
red_std_guess = estimate_std(both_colours, red_weight, red_mean_guess)
red_mean_guess = estimate_mean(both_colours, red_weight)
blue_mean_guess = estimate_mean(both_colours, blue_weight)
print("red mean:", red_mean_guess, ":::::::::", "blue mean:", blue_mean_guess)
print("red std:", red_std_guess, ":::::::::", "blue std:", blue_std_guess)

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
y = np.zeros(len(both_colours))
mured = red_mean_guess
sigmared = red_std_guess
x = np.linspace(mured - 2.5*sigmared, mured + 2.5*sigmared, 100)
plt.plot(x,norm.pdf(x, mured, sigmared))
mublue = blue_mean_guess
sigmablue = blue_std_guess
y = np.linspace(mublue - 2.5*sigmablue, mublue + 2.5*sigmablue, 100)
plt.plot(y,norm.pdf(y, mublue, sigmablue))
for i in range(len(both_colours)):
    plt.plot(both_colours[i],0,"bo")
plt.show()
```

OUTPUT:

```
red mean: 2.8939486098495264 ::::::::: blue mean: 6.817385954777204
red std: 1.1842294755422575 ::::::::: blue std: 1.3307903761287299
```

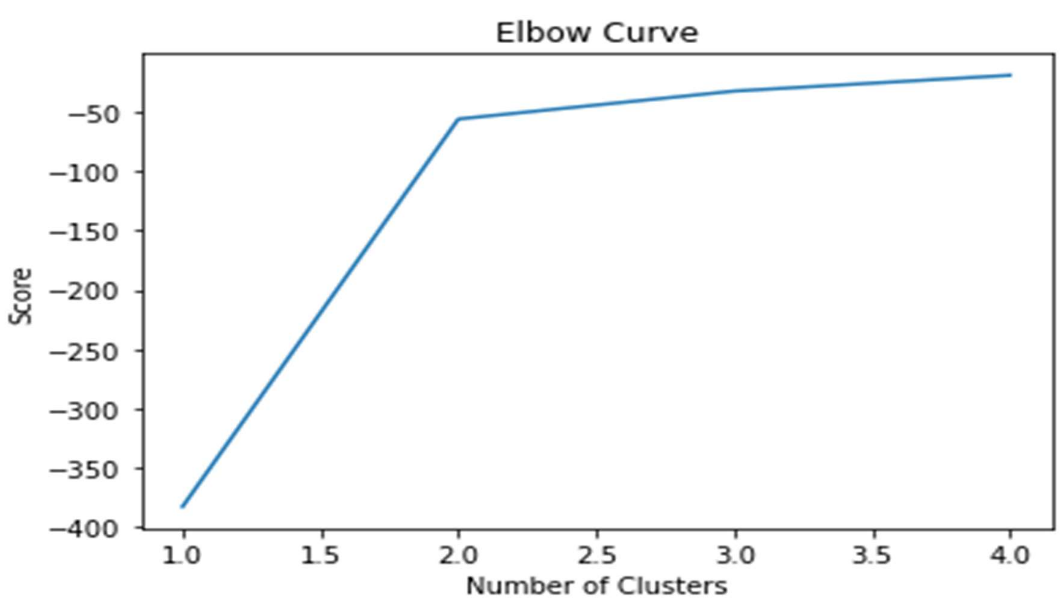


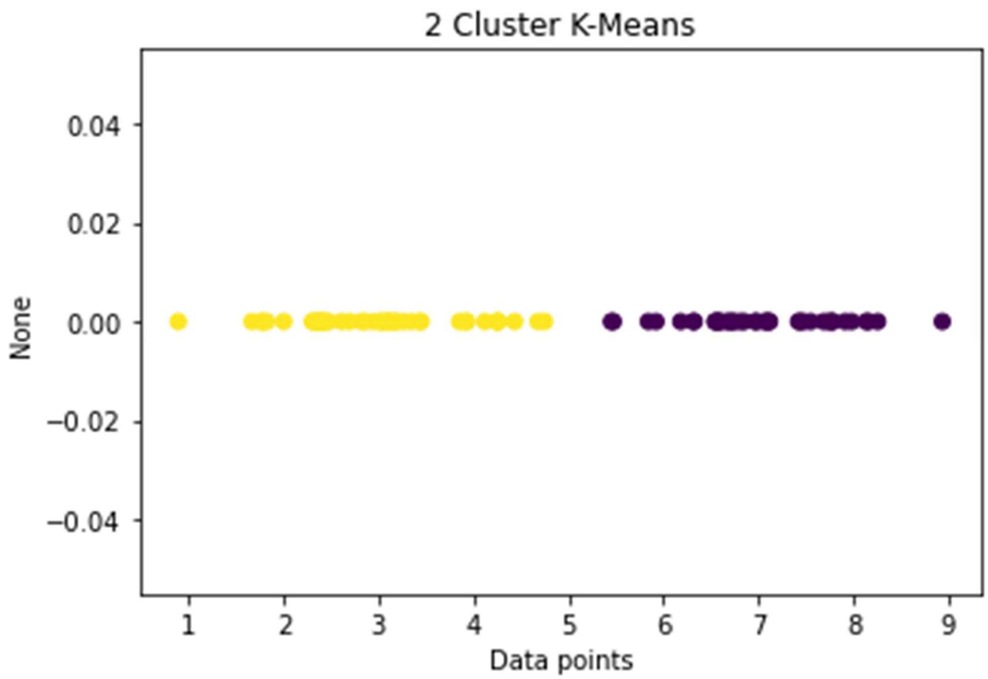
K-MEANS

```
import pylab as pl

import numpy as np

from sklearn.cluster import KMeans

np.random.seed(110)


red_mean = 3

red_std = 0.8

blue_mean = 7

blue_std = 1


red = np.random.normal(red_mean, red_std, size=40)

blue = np.random.normal(blue_mean, blue_std, size=40)

both_colours = np.sort(np.concatenate((red, blue)))

y = np.zeros(len(both_colours))
```

```
kmeans=KMeans(n_clusters=2)
kmeansoutput=kmeans.fit(both_colours.reshape(-1,1))
Nc = range(1, 5)
kmeans = [KMeans(n_clusters=i) for i in Nc]
score = [kmeans[i].fit(both_colours.reshape(-1,1)).score(both_colours.reshape(-1,1)) for
i in range(len(kmeans))]
pl.plot(Nc,score)
pl.xlabel('Number of Clusters')
pl.ylabel('Score')
pl.title('Elbow Curve')
pl.show()
pl.scatter(both_colours,y,c=kmeansoutput.labels_)
pl.xlabel('Data points')
pl.ylabel('None')
pl.title('2 Cluster K-Means')
pl.show()
```

OUPUT:-




2 Cluster K-Means
0.04
0.02
0.00
−0.02
−0.04
None
1
2
3
4
5
6
7
8
9
Data points

8. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
from sklearn.datasets import load_iris

iris = load_iris()

print("Feature Names:",iris.feature_names,"Iris Data:",iris.data,"Target
Names:",iris.target_names,"Target:",iris.target)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(

iris.data, iris.target, test_size = .25)

from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier()

clf.fit(X_train, y_train)

print(" Accuracy=",clf.score(X_test, y_test))

print("Predicted Data")

print(clf.predict(X_test))

prediction=clf.predict(X_test)

print("Test data :")

print(y_test)

diff=prediction-y_test

print("Result is ")

print(diff)

print('Total no of samples misclassied =', sum(abs(diff)))
```

OUTPUT:-

```
Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Iris Data: [[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 0.2]

[5. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

```
[4.9 3.6 1.4 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]] Target Names: ['setosa' 'versicolor' 'virginica'] Target: [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]

Accuracy= 0.9473684210526315

Predicted Data

[0 0 2 1 2 1 1 0 1 2 1 1 2 0 1 1 0 2 0 2 0 0 1 0 2 1 1 0 0 0 1 2 0 0 2 1 2

2]

Test data :

[0 0 2 1 2 1 1 0 1 2 1 1 2 0 1 1 0 2 0 1 0 0 1 0 2 1 1 0 0 0 1 2 0 0 2 2 2

2]

Result is

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

  0 0 0 0 0 0 0 0 0 0 0 -1 0 0]

Total no of samples misclassied = 2

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.

***** Create Excel file LR.csv and save it in same path**

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
      diff = point - X[j]
      weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
     wei = kernel(point,xmat,k)
     W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
     return W
def localWeightRegression(xmat,ymat,k):
   m,n = np.shape(xmat)
   ypred = np.zeros(m)
   for i in range(m):


      ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
      return ypred

data = pd.read_csv('LR.csv')
colA = np.array(data.colA)
```

```
colB = np.array(data.colB)
mcolA = np.mat(colA)
mcolB = np.mat(colB)
m= np.shape(mcolA)[1]
one = np.ones((1,m),dtype=int)

X= np.hstack((one.T,mcolA.T))
print(X.shape)
#set k here (0.5)
ypred = localWeightRegression(X,mcolB,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(colA,colB, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('TV')
plt.ylabel('radio')
plt.show();
```

OUTPUT:-