`

JYOTHY INSTITUTE OF TECHNOLOGY

AFFILIATED TO VTU, BELAGAVI

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

ACCREDITED BY NBA, NEW DELHI

LAB MANUAL

FOR

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LABORATORY

(18CSL76)

`

Course Details

Course Name	**:**	**Artificial Intelligence & Machine Learning Lab**
Course Code	**:**	**18CSL76**
Course prerequisite	**:**	**Basic Knowledge of Python Programming**

Course Objectives

1. **Implement and evaluate AI and ML algorithms in and Python programming language**

Course Outcomes

• Implement and demonstrate AI and ML algorithms

• Evaluate different algorithms

Conduction 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 (Courseed to change in accoradance with university regulations)

q) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks

r) 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

`

LAB EXPERIMENTS

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 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.

`

Program 1 : Implement A* Search Algorithm

Source Code:

```
def A_star(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
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    #n is set its parent
                    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:    # if better cost found, then update the existing cost 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('Optimal Path :', 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 = {
            'S': 8,
            'A': 8,
            'B': 4,
            'C': 3,
            'D': 1000,
            'E': 1000,
            'G': 0,

        }


        return H_dist[n]
```

`

```
#Describe your graph here
Graph_nodes = {'S': [['A', 1], ['B', 5], ['C', 8]],
        'A': [['D', 3], ['E', 7], ['G', 9]],
        'B': [['G', 4]],
        'C': [['G', 5]],
        'D': None,
        'E': None}

A_star('S', 'G')
```

Output:

Optimal Path : ['S', 'B', 'G']

`

Program 2 : Implement AO* Search Algorithm

Source Code:

```
def recAOStar(n):
    global finalPath
    print("Expanding Node : ", n)
    and_nodes = []
    or_nodes = []

    #Segregation of AND and OR nodes
    if (n in allNodes):
        if 'AND' in allNodes[n]:
            and_nodes = allNodes[n]['AND']
        if 'OR' in allNodes[n]:
            or_nodes = allNodes[n]['OR']

    # If leaf node then return
    if len(and_nodes) == 0 and len(or_nodes) == 0:
        return

    solvable = False
    marked = {}

    while not solvable:
        # If all the child nodes are visited and expanded, take the least cost of all the child nodes
        if len(marked) == len(and_nodes) + len(or_nodes):
            min_cost_least, min_cost_group_least = least_cost_group(and_nodes, or_nodes, {})
            solvable = True
            change_heuristic(n, min_cost_least)
            optimal_child_group[n] = min_cost_group_least
            continue

        # Least cost of the unmarked child nodes
        min_cost, min_cost_group = least_cost_group(and_nodes, or_nodes, marked)

        is_expanded = False

        # If the child nodes have sub trees then recursively visit them to recalculate the heuristic of the child node
        if len(min_cost_group) > 1:
            if (min_cost_group[0] in allNodes):
                is_expanded = True
                recAOStar(min_cost_group[0])
            if (min_cost_group[1] in allNodes):
                is_expanded = True
                recAOStar(min_cost_group[1])
        else:
            if (min_cost_group in allNodes):
                is_expanded = True
                recAOStar(min_cost_group)


        # If the child node had any subtree and expanded, verify if the new heuristic value is still the least among all nodes
        if is_expanded:
            min_cost_verify, min_cost_group_verify = least_cost_group(and_nodes, or_nodes, {})

            if min_cost_group == min_cost_group_verify:
                solvable = True
```

`

```
            change_heuristic(n, min_cost_verify)
            optimal_child_group[n] = min_cost_group

    # If the child node does not have any subtrees then no change in heuristic, so update the min cost of the current node
    else:
        solvable = True
        change_heuristic(n, min_cost)
        optimal_child_group[n] = min_cost_group

    #Mark the child node which was expanded
    marked[min_cost_group] = 1
  return heuristic(n)


# Function to calculate the min cost among all the child nodes
def least_cost_group(and_nodes, or_nodes, marked):
  node_wise_cost = {}

  for node_pair in and_nodes:
    if not node_pair[0] + node_pair[1] in marked:
      cost = 0
      cost = cost + heuristic(node_pair[0]) + heuristic(node_pair[1]) + 2
      node_wise_cost[node_pair[0] + node_pair[1]] = cost

  for node in or_nodes:
    if not node in marked:
      cost = 0
      cost = cost + heuristic(node) + 1
      node_wise_cost[node] = cost

  min_cost = 999999
  min_cost_group = None

  # Calculates the min heuristic
  for costKey in node_wise_cost:
    if node_wise_cost[costKey] < min_cost:
      min_cost = node_wise_cost[costKey]
      min_cost_group = costKey

  return [min_cost, min_cost_group]


# Returns heuristic of a node
def heuristic(n):
  return H_dist[n]

# Updates the heuristic of a node
def change_heuristic(n, cost):
  H_dist[n] = cost
  return

# Function to print the optimal cost nodes
def print_path(node):
  print(optimal_child_group[node], end="")
  node = optimal_child_group[node]

  if len(node) > 1:
    if node[0] in optimal_child_group:
      print("->", end="")
```

`

```
            print_path(node[0])
        if node[1] in optimal_child_group:
            print("->", end="")
            print_path(node[1])
    else:
        if node in optimal_child_group:
            print("->", end="")
            print_path(node)


#Describe the heuristic here
H_dist = {
    'A': -1,
    'B': 4,
    'C': 2,
    'D': 3,
    'E': 6,
    'F': 8,
    'G': 2,
    'H': 0,
    'I': 0,
    'J': 0
}


#Describe your graph here
allNodes = {
    'A': {'AND': [('C', 'D')], 'OR': ['B']},
    'B': {'OR': ['E', 'F']},
    'C': {'OR': ['G'], 'AND': [('H', 'I')]},
    'D': {'OR': ['J']}
}

optimal_child_group = {}
optimal_cost = recAOStar('A')

print('Nodes which gives optimal cost are')
print_path('A')
print('\nOptimal Cost is  :: ', optimal_cost)
```

`

Output:

Expanding Node : A

Expanding Node : B

Expanding Node : C

Expanding Node : D

Nodes which gives optimal cost are

CD->HI->J

Optimal Cost is :: 5

`

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.

Source Code:

```
import numpy as np
import pandas as pd
# Loading Data from a CSV File
data1= pd.read_csv('Training_examples_1.csv')
#data = pd.DataFrame(data=data1)
# Separating concept features from Target
concepts = np.array(data1.iloc[:,0:-1])
# Isolating target into a separate DataFrame
#copying last column to target array
target = np.array(data1.iloc[:,-1])
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("initialization of specific_h and general_h")
    print("\n specific_h :")
    print(specific_h)

    general_h = [["?" for i in range(len(specific_h))]
                for i in range(len(specific_h))]
    print("\n general_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(" \n steps of Candidate Elimination Algorithm",i+1)
        print("\n specific_h:")
        print(specific_h)
        print("\n general_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("\n Final Specific_h:", s_final, sep="\n")
print("\n Final General_h:", g_final, sep="\n")
```

Input csv file:

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

Output:

initialization of specific_h and general_h

specific_h :

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

general_h:

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of Candidate Elimination Algorithm 1

specific_h:

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

general_h:

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of Candidate Elimination Algorithm 2

specific_h:

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

general_h:

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

`

steps of Candidate Elimination Algorithm 3

specific_h:

['Sunny' 'Warm' 'High' 'Strong' '?' '?']

general_h:

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

Final Specific_h:

['Sunny' 'Warm' 'High' '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.

Source Code:

```
import pandas as pd
import numpy as np
#Import the dataset and define the feature as well as the target datasets / columns#
dataset = pd.read_csv('playtennis.csv',
                      names=['outlook','temperature','humidity','wind','class',])
#Import all columns omitting the fist which consists the names of the animals
#We drop the animal names since this is not a good feature to split the data on
attributes =('Outlook','Temperature','Humidity','Wind','PlayTennis')
def entropy(target_col):
    """
    Calculate the entropy of a dataset.
    The only parameter of this function is the target_col parameter which specifies
    the target column
    """
    elements,counts = np.unique(target_col,return_counts = True)
    total_count = np.sum(counts)
    entropy = np.sum([(-counts[i]/total_count)*np.log2(counts[i]/total_count) for i in range(len(elements))])
    #print('Entropy =', entropy)
    return entropy
def InfoGain(data,split_attribute_name,target_name="class"):
    #Calculate the entropy of the total dataset
    total_entropy = entropy(data[target_name])

    ##Calculate the entropy of the dataset

    #Calculate the values and the corresponding counts for the split attribute
    vals,counts= np.unique(data[split_attribute_name],return_counts=True)

    #Calculate the weighted entropy
    Weighted_Entropy =
np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[targe
t_name]) for i in range(len(vals))])

    #Calculate the information gain
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain

def ID3(data,originaldata,features,target_attribute_name="class",parent_node_class = None):
    #Define the stopping criteria --> If one of this is satisfied, we want to return a leaf node#

    #If all target_values have the same value, return this value

    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
```

`

```
    #If the dataset is empty, return the mode target feature value in the original dataset
    elif len(data)==0:
        return
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attribute_name],ret
urn_counts=True)[1])]

    elif len(features) ==0:
        #return parent_node_class
        return
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attribute_name],ret
urn_counts=True)[1])]


    #If none of the above holds true, grow the tree!

    else:
        #Set the default value for this node --> The mode target feature value of the current node
        parent_node_class =
np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name],return_counts=Tr
ue)[1])]

        #Select the feature which best splits the dataset
        item_values = [InfoGain(data,feature,target_attribute_name) for feature in features] #Return the
information gain values for the features in the dataset
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]

        #Create the tree structure. The root gets the name of the feature (best_feature) with the maximum
information
        #gain in the first run
        tree = {best_feature:{}}


        #Remove the feature with the best inforamtion gain from the feature space
        features = [i for i in features if i != best_feature]

        #Grow a branch under the root node for each possible value of the root node feature

        for value in np.unique(data[best_feature]):
            value = value
            #Split the dataset along the value of the feature with the largest information gain and therwith create
sub_datasets
            sub_data = data.where(data[best_feature] == value).dropna()

            #Call the ID3 algorithm for each of those sub_datasets with the new parameters --> Here the
recursion comes in!
            subtree = ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)

            #Add the sub tree, grown from the sub_dataset to the tree under the root node
            tree[best_feature][value] = subtree

        return(tree)
```

`

```
def predict(query,tree,default = 1):

    #1.
    for key in list(query.keys()):
        if key in list(tree.keys()):
            #2.
            try:
                result = tree[key][query[key]]
            except:
                return default

            #3.
            result = tree[key][query[key]]
            #4.
            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)
    #We drop the index respectively relabel the index
    #starting form 0, because we do not want to run into errors regarding the row labels / indexes
    #testing_data = dataset.iloc[10:].reset_index(drop=True)
    return training_data  #,testing_data
def test(data,tree):
    #Create new query instances by simply removing the target feature column from the original dataset and
    #convert it to a dictionary
    queries = data.iloc[:,:-1].to_dict(orient = "records")

    #Create a empty DataFrame in whose columns the prediction of the tree are stored
    predicted = pd.DataFrame(columns=["predicted"])

    #Calculate the prediction accuracy
    for i in range(len(data)):
        predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)

    print('\n The prediction accuracy is: ',(np.sum(predicted["predicted"] ==
data["class"])/len(data))*100,'%')
"""
Train the tree, Print the tree and predict the accuracy
"""
XX = train_test_split(dataset)
training_data=XX
#elements,counts = np.unique(training_data["class"],return_counts = True)

"""
i=0
for value in np.unique(training_data["outlook"]):
            value = value
            sub_data = training_data.where(training_data["outlook"] == value).dropna()
```

```
`
        print(i+1, "Subdata for value=", value, "is:\n", sub_data)
        i+=1
"""

#testing_data=XX[1]
tree = ID3(training_data,training_data,training_data.columns[:-1])
print(' \n Display Tree',tree)
print('\n len of training data =',len(training_data))
test(training_data,tree)
```

Input csv file:

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
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	Normal	Strong	Yes

Output:

Display Tree {'outlook': {'Overcast': 'Yes', 'Rain': {'wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}}}

len of training data = 14

The prediction accuracy is: 100.0 %

`

Program 5 : Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Source Code:

```
from math import exp
from random import seed
from random import random
# Initialize a network
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
# Calculate neuron activation for an input
def activate(weights, inputs):
  activation = weights[-1]
  for i in range(len(weights)-1):
    activation += weights[i] * inputs[i]
  return activation
# Transfer neuron activation
def transfer(activation):
  return 1.0 / (1.0 + exp(-activation))
# Forward propagate input to a network output
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
# Calculate the derivative of an neuron output
def transfer_derivative(output):
  return output * (1.0 - output)
# Backpropagate error and store in neurons
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'])
# Update network weights with error
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']
# Train a network for a fixed number of epochs
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))
# Test training backprop algorithm
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=6.350

>epoch=1, lrate=0.500, error=5.531

>epoch=2, lrate=0.500, error=5.221

>epoch=3, lrate=0.500, error=4.951

>epoch=4, lrate=0.500, error=4.519

>epoch=5, lrate=0.500, error=4.173

>epoch=6, lrate=0.500, error=3.835

>epoch=7, lrate=0.500, error=3.506

>epoch=8, lrate=0.500, error=3.192

>epoch=9, lrate=0.500, error=2.898

>epoch=10, lrate=0.500, error=2.626

>epoch=11, lrate=0.500, error=2.377

>epoch=12, lrate=0.500, error=2.153

>epoch=13, lrate=0.500, error=1.953

>epoch=14, lrate=0.500, error=1.774

>epoch=15, lrate=0.500, error=1.614

>epoch=16, lrate=0.500, error=1.472

>epoch=17, lrate=0.500, error=1.346

>epoch=18, lrate=0.500, error=1.233

>epoch=19, lrate=0.500, error=1.132

`

[{'delta': -0.0059546604162323625, 'output': 0.029980305604426185, 'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297]}, {'delta': 0.0026279652850863837, 'output': 0.9456229000211323, 'weights': [0.37711098142462157, -0.0625909894552989, 0.2765123702642716]}]

[{'delta': -0.04270059278364587, 'output': 0.23648794202357587, 'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275]}, {'delta': 0.03803132596437354, 'output': 0.7790535202438367, 'weights': [-2.5584149848484263, 1.0036422106209202, 0.42383086467582715]}]

`

Program 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.

Source Code:

```
import csv
import random
import math

#1.Load Data
def loadCsv(filename):
    filename="diabetes1.csv"
    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

#Split the data into Training and Testing  randomly
def splitDataset(dataset, splitRatio):
    #splitRatio = 0.7
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    copy = list(dataset)
    while len(trainSet) < trainSize:
        #Using randrange() to generate numbers 0 to len(copy)=length of dataset
        index = random.randrange(len(copy))
        # pop: removes and returns the element at
        #the given index (passed as an argument) from the list,
        trainSet.append(copy.pop(index))
    return [trainSet, copy]

#Seperatedata by Class
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

#Calculate Mean
def mean(numbers):
   return sum(numbers)/float(len(numbers))

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

#Summarize the data
def summarize(dataset):
   summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
   del summaries[-1]
```

```
`
   return summaries

#Summarize Attributes by Class
def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    print(len(separated))
    summaries = {}
    # dictionary.items returns a copy of the
    #dictionary’s list of (key, value) pairs
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    print(summaries)
    return summaries

#Calculate Gaussian Probability Density Function
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

#Calculate Class Probabilities
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

#Make a Prediction
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

#return a list of predictions for each test instance.
def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
        print(i+1,': ', testSet[i],"--",result)
    return predictions

#calculate accuracy ratio.
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 = 'diabetes1.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)))

# prepare model
summaries = summarizeByClass(trainingSet)

# test model
predictions = getPredictions(summaries, testSet)
accuracy = getAccuracy(testSet, predictions)
print('Accuracy: {0}%'.format(accuracy))
```

Input csv file:

diabetes1.csv

Output:

Split 768 rows into train=537 and test=231 rows

2

{1.0: [(4.818681318681318, 3.7512550264640403), (141.35164835164835, 31.11158801588462), (71.25824175824175, 20.515573203126248), (22.47252747252747, 18.0504759793018), (102.52197802197803, 144.26348308900586), (35.201098901098916, 6.850736856048957), (0.5703791208791211, 0.39969461298602904), (36.92307692307692, 11.11142040462941)], 0.0: [(3.174647887323944, 3.0211924868060125), (109.43943661971831, 25.717470149066894), (68.8056338028169, 16.84857692631019), (19.952112676056338, 14.756611648068148), (69.27323943661972, 96.11463988052692), (30.475492957746507, 7.393503177365069), (0.43643943661971835, 0.29078308186730295), (30.977464788732394, 11.59653899740735)]}

1 : [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0] -- 0.0

2 : [10.0, 139.0, 80.0, 0.0, 0.0, 27.1, 1.441, 57.0, 0.0] -- 1.0

3 : [7.0, 100.0, 0.0, 0.0, 0.0, 30.0, 0.484, 32.0, 1.0] -- 1.0

4 : [0.0, 118.0, 84.0, 47.0, 230.0, 45.8, 0.551, 31.0, 1.0] -- 1.0

5 : [7.0, 107.0, 74.0, 0.0, 0.0, 29.6, 0.254, 31.0, 1.0] -- 0.0

6 : [11.0, 143.0, 94.0, 33.0, 146.0, 36.6, 0.254, 51.0, 1.0] -- 1.0

7 : [10.0, 125.0, 70.0, 26.0, 115.0, 31.1, 0.205, 41.0, 1.0] -- 1.0

8 : [1.0, 97.0, 66.0, 15.0, 140.0, 23.2, 0.487, 22.0, 0.0] -- 0.0

`

9 : [5.0, 117.0, 92.0, 0.0, 0.0, 34.1, 0.337, 38.0, 0.0] -- 0.0

10 : [11.0, 138.0, 76.0, 0.0, 0.0, 33.2, 0.42, 35.0, 0.0] -- 1.0

11 : [4.0, 111.0, 72.0, 47.0, 207.0, 37.1, 1.39, 56.0, 1.0] -- 1.0

12 : [3.0, 180.0, 64.0, 25.0, 70.0, 34.0, 0.271, 26.0, 0.0] -- 1.0

13 : [7.0, 106.0, 92.0, 18.0, 0.0, 22.7, 0.235, 48.0, 0.0] -- 0.0

14 : [2.0, 71.0, 70.0, 27.0, 0.0, 28.0, 0.586, 22.0, 0.0] -- 0.0

15 : [8.0, 176.0, 90.0, 34.0, 300.0, 33.7, 0.467, 58.0, 1.0] -- 1.0

16 : [1.0, 73.0, 50.0, 10.0, 0.0, 23.0, 0.248, 21.0, 0.0] -- 0.0

17 : [2.0, 84.0, 0.0, 0.0, 0.0, 0.0, 0.304, 21.0, 0.0] -- 0.0

18 : [7.0, 114.0, 66.0, 0.0, 0.0, 32.8, 0.258, 42.0, 1.0] -- 0.0

19 : [1.0, 0.0, 48.0, 20.0, 0.0, 24.7, 0.14, 22.0, 0.0] -- 0.0

20 : [2.0, 112.0, 66.0, 22.0, 0.0, 25.0, 0.307, 24.0, 0.0] -- 0.0

21 : [3.0, 113.0, 44.0, 13.0, 0.0, 22.4, 0.14, 22.0, 0.0] -- 0.0

22 : [2.0, 74.0, 0.0, 0.0, 0.0, 0.0, 0.102, 22.0, 0.0] -- 0.0

23 : [2.0, 100.0, 68.0, 25.0, 71.0, 38.5, 0.324, 26.0, 0.0] -- 0.0

24 : [7.0, 81.0, 78.0, 40.0, 48.0, 46.7, 0.261, 42.0, 0.0] -- 0.0

25 : [4.0, 134.0, 72.0, 0.0, 0.0, 23.8, 0.277, 60.0, 1.0] -- 0.0

26 : [1.0, 126.0, 56.0, 29.0, 152.0, 28.7, 0.801, 21.0, 0.0] -- 0.0

27 : [4.0, 144.0, 58.0, 28.0, 140.0, 29.5, 0.287, 37.0, 0.0] -- 0.0

28 : [3.0, 83.0, 58.0, 31.0, 18.0, 34.3, 0.336, 25.0, 0.0] -- 0.0

29 : [7.0, 160.0, 54.0, 32.0, 175.0, 30.5, 0.588, 39.0, 1.0] -- 1.0

30 : [4.0, 146.0, 92.0, 0.0, 0.0, 31.2, 0.539, 61.0, 1.0] -- 1.0

`

31 : [5.0, 124.0, 74.0, 0.0, 0.0, 34.0, 0.22, 38.0, 1.0] -- 0.0

32 : [5.0, 78.0, 48.0, 0.0, 0.0, 33.7, 0.654, 25.0, 0.0] -- 0.0

33 : [4.0, 99.0, 76.0, 15.0, 51.0, 23.2, 0.223, 21.0, 0.0] -- 0.0

34 : [0.0, 113.0, 76.0, 0.0, 0.0, 33.3, 0.278, 23.0, 1.0] -- 0.0

35 : [1.0, 88.0, 30.0, 42.0, 99.0, 55.0, 0.496, 26.0, 1.0] -- 0.0

36 : [3.0, 120.0, 70.0, 30.0, 135.0, 42.9, 0.452, 30.0, 0.0] -- 0.0

37 : [9.0, 122.0, 56.0, 0.0, 0.0, 33.3, 1.114, 33.0, 1.0] -- 1.0

38 : [5.0, 106.0, 82.0, 30.0, 0.0, 39.5, 0.286, 38.0, 0.0] -- 0.0

39 : [4.0, 154.0, 62.0, 31.0, 284.0, 32.8, 0.237, 23.0, 0.0] -- 1.0

40 : [5.0, 147.0, 78.0, 0.0, 0.0, 33.7, 0.218, 65.0, 0.0] -- 1.0

41 : [1.0, 136.0, 74.0, 50.0, 204.0, 37.4, 0.399, 24.0, 0.0] -- 1.0

42 : [1.0, 153.0, 82.0, 42.0, 485.0, 40.6, 0.687, 23.0, 0.0] -- 1.0

43 : [8.0, 188.0, 78.0, 0.0, 0.0, 47.9, 0.137, 43.0, 1.0] -- 1.0

44 : [7.0, 152.0, 88.0, 44.0, 0.0, 50.0, 0.337, 36.0, 1.0] -- 1.0

45 : [0.0, 114.0, 80.0, 34.0, 285.0, 44.2, 0.167, 27.0, 0.0] -- 1.0

46 : [6.0, 104.0, 74.0, 18.0, 156.0, 29.9, 0.722, 41.0, 1.0] -- 0.0

47 : [4.0, 120.0, 68.0, 0.0, 0.0, 29.6, 0.709, 34.0, 0.0] -- 0.0

48 : [4.0, 110.0, 66.0, 0.0, 0.0, 31.9, 0.471, 29.0, 0.0] -- 0.0

49 : [2.0, 87.0, 0.0, 23.0, 0.0, 28.9, 0.773, 25.0, 0.0] -- 0.0

50 : [8.0, 179.0, 72.0, 42.0, 130.0, 32.7, 0.719, 36.0, 1.0] -- 1.0

51 : [0.0, 129.0, 110.0, 46.0, 130.0, 67.1, 0.319, 26.0, 1.0] -- 1.0

52 : [8.0, 109.0, 76.0, 39.0, 114.0, 27.9, 0.64, 31.0, 1.0] -- 0.0

`

53 : [7.0, 159.0, 66.0, 0.0, 0.0, 30.4, 0.383, 36.0, 1.0] -- 1.0

54 : [1.0, 105.0, 58.0, 0.0, 0.0, 24.3, 0.187, 21.0, 0.0] -- 0.0

55 : [4.0, 109.0, 64.0, 44.0, 99.0, 34.8, 0.905, 26.0, 1.0] -- 0.0

56 : [4.0, 148.0, 60.0, 27.0, 318.0, 30.9, 0.15, 29.0, 1.0] -- 1.0

57 : [0.0, 113.0, 80.0, 16.0, 0.0, 31.0, 0.874, 21.0, 0.0] -- 0.0

58 : [6.0, 103.0, 72.0, 32.0, 190.0, 37.7, 0.324, 55.0, 0.0] -- 1.0

59 : [5.0, 111.0, 72.0, 28.0, 0.0, 23.9, 0.407, 27.0, 0.0] -- 0.0

60 : [1.0, 96.0, 64.0, 27.0, 87.0, 33.2, 0.289, 21.0, 0.0] -- 0.0

61 : [7.0, 184.0, 84.0, 33.0, 0.0, 35.5, 0.355, 41.0, 1.0] -- 1.0

62 : [0.0, 147.0, 85.0, 54.0, 0.0, 42.8, 0.375, 24.0, 0.0] -- 1.0

63 : [12.0, 151.0, 70.0, 40.0, 271.0, 41.8, 0.742, 38.0, 1.0] -- 1.0

64 : [6.0, 125.0, 68.0, 30.0, 120.0, 30.0, 0.464, 32.0, 0.0] -- 0.0

65 : [1.0, 100.0, 66.0, 15.0, 56.0, 23.6, 0.666, 26.0, 0.0] -- 0.0

66 : [1.0, 87.0, 78.0, 27.0, 32.0, 34.6, 0.101, 22.0, 0.0] -- 0.0

67 : [0.0, 101.0, 76.0, 0.0, 0.0, 35.7, 0.198, 26.0, 0.0] -- 0.0

68 : [3.0, 162.0, 52.0, 38.0, 0.0, 37.2, 0.652, 24.0, 1.0] -- 1.0

69 : [4.0, 197.0, 70.0, 39.0, 744.0, 36.7, 2.329, 31.0, 0.0] -- 1.0

70 : [0.0, 117.0, 80.0, 31.0, 53.0, 45.2, 0.089, 24.0, 0.0] -- 0.0

71 : [6.0, 134.0, 80.0, 37.0, 370.0, 46.2, 0.238, 46.0, 1.0] -- 1.0

72 : [4.0, 122.0, 68.0, 0.0, 0.0, 35.0, 0.394, 29.0, 0.0] -- 0.0

73 : [7.0, 181.0, 84.0, 21.0, 192.0, 35.9, 0.586, 51.0, 1.0] -- 1.0

74 : [0.0, 179.0, 90.0, 27.0, 0.0, 44.1, 0.686, 23.0, 1.0] -- 1.0

`

75 : [6.0, 119.0, 50.0, 22.0, 176.0, 27.1, 1.318, 33.0, 1.0] -- 1.0

76 : [2.0, 146.0, 76.0, 35.0, 194.0, 38.2, 0.329, 29.0, 0.0] -- 1.0

77 : [9.0, 124.0, 70.0, 33.0, 402.0, 35.4, 0.282, 34.0, 0.0] -- 1.0

78 : [9.0, 106.0, 52.0, 0.0, 0.0, 31.2, 0.38, 42.0, 0.0] -- 0.0

79 : [12.0, 92.0, 62.0, 7.0, 258.0, 27.6, 0.926, 44.0, 1.0] -- 1.0

80 : [3.0, 111.0, 56.0, 39.0, 0.0, 30.1, 0.557, 30.0, 0.0] -- 0.0

81 : [2.0, 114.0, 68.0, 22.0, 0.0, 28.7, 0.092, 25.0, 0.0] -- 0.0

82 : [3.0, 191.0, 68.0, 15.0, 130.0, 30.9, 0.299, 34.0, 0.0] -- 1.0

83 : [3.0, 122.0, 78.0, 0.0, 0.0, 23.0, 0.254, 40.0, 0.0] -- 0.0

84 : [13.0, 106.0, 70.0, 0.0, 0.0, 34.2, 0.251, 52.0, 0.0] -- 1.0

85 : [7.0, 106.0, 60.0, 24.0, 0.0, 26.5, 0.296, 29.0, 1.0] -- 0.0

86 : [5.0, 114.0, 74.0, 0.0, 0.0, 24.9, 0.744, 57.0, 0.0] -- 0.0

87 : [2.0, 108.0, 62.0, 10.0, 278.0, 25.3, 0.881, 22.0, 0.0] -- 0.0

88 : [0.0, 146.0, 70.0, 0.0, 0.0, 37.9, 0.334, 28.0, 1.0] -- 0.0

89 : [0.0, 107.0, 62.0, 30.0, 74.0, 36.6, 0.757, 25.0, 1.0] -- 0.0

90 : [8.0, 112.0, 72.0, 0.0, 0.0, 23.6, 0.84, 58.0, 0.0] -- 0.0

91 : [2.0, 144.0, 58.0, 33.0, 135.0, 31.6, 0.422, 25.0, 1.0] -- 0.0

92 : [3.0, 150.0, 76.0, 0.0, 0.0, 21.0, 0.207, 37.0, 0.0] -- 0.0

93 : [0.0, 137.0, 68.0, 14.0, 148.0, 24.8, 0.143, 21.0, 0.0] -- 0.0

94 : [2.0, 124.0, 68.0, 28.0, 205.0, 32.9, 0.875, 30.0, 1.0] -- 0.0

95 : [2.0, 155.0, 74.0, 17.0, 96.0, 26.6, 0.433, 27.0, 1.0] -- 0.0

96 : [7.0, 109.0, 80.0, 31.0, 0.0, 35.9, 1.127, 43.0, 1.0] -- 1.0

`

97 : [2.0, 112.0, 68.0, 22.0, 94.0, 34.1, 0.315, 26.0, 0.0] -- 0.0

98 : [3.0, 182.0, 74.0, 0.0, 0.0, 30.5, 0.345, 29.0, 1.0] -- 1.0

99 : [3.0, 115.0, 66.0, 39.0, 140.0, 38.1, 0.15, 28.0, 0.0] -- 0.0

100 : [13.0, 152.0, 90.0, 33.0, 29.0, 26.8, 0.731, 43.0, 1.0] -- 1.0

101 : [6.0, 105.0, 70.0, 32.0, 68.0, 30.8, 0.122, 37.0, 0.0] -- 0.0

102 : [1.0, 180.0, 0.0, 0.0, 0.0, 43.3, 0.282, 41.0, 1.0] -- 1.0

103 : [12.0, 106.0, 80.0, 0.0, 0.0, 23.6, 0.137, 44.0, 0.0] -- 1.0

104 : [1.0, 130.0, 70.0, 13.0, 105.0, 25.9, 0.472, 22.0, 0.0] -- 0.0

105 : [1.0, 95.0, 74.0, 21.0, 73.0, 25.9, 0.673, 36.0, 0.0] -- 0.0

106 : [8.0, 95.0, 72.0, 0.0, 0.0, 36.8, 0.485, 57.0, 0.0] -- 0.0

107 : [3.0, 116.0, 0.0, 0.0, 0.0, 23.5, 0.187, 23.0, 0.0] -- 0.0

108 : [3.0, 99.0, 62.0, 19.0, 74.0, 21.8, 0.279, 26.0, 0.0] -- 0.0

109 : [5.0, 0.0, 80.0, 32.0, 0.0, 41.0, 0.346, 37.0, 1.0] -- 0.0

110 : [1.0, 125.0, 50.0, 40.0, 167.0, 33.3, 0.962, 28.0, 1.0] -- 1.0

111 : [13.0, 129.0, 0.0, 30.0, 0.0, 39.9, 0.569, 44.0, 1.0] -- 1.0

112 : [1.0, 196.0, 76.0, 36.0, 249.0, 36.5, 0.875, 29.0, 1.0] -- 1.0

113 : [4.0, 146.0, 78.0, 0.0, 0.0, 38.5, 0.52, 67.0, 1.0] -- 1.0

114 : [5.0, 99.0, 54.0, 28.0, 83.0, 34.0, 0.499, 30.0, 0.0] -- 0.0

115 : [6.0, 124.0, 72.0, 0.0, 0.0, 27.6, 0.368, 29.0, 1.0] -- 0.0

116 : [3.0, 81.0, 86.0, 16.0, 66.0, 27.5, 0.306, 22.0, 0.0] -- 0.0

117 : [1.0, 133.0, 102.0, 28.0, 140.0, 32.8, 0.234, 45.0, 1.0] -- 1.0

118 : [2.0, 122.0, 52.0, 43.0, 158.0, 36.2, 0.816, 28.0, 0.0] -- 0.0

`

119 : [0.0, 93.0, 100.0, 39.0, 72.0, 43.4, 1.021, 35.0, 0.0] -- 1.0

120 : [1.0, 119.0, 54.0, 13.0, 50.0, 22.3, 0.205, 24.0, 0.0] -- 0.0

121 : [8.0, 105.0, 100.0, 36.0, 0.0, 43.3, 0.239, 45.0, 1.0] -- 1.0

122 : [0.0, 131.0, 66.0, 40.0, 0.0, 34.3, 0.196, 22.0, 1.0] -- 0.0

123 : [4.0, 95.0, 64.0, 0.0, 0.0, 32.0, 0.161, 31.0, 1.0] -- 0.0

124 : [5.0, 136.0, 84.0, 41.0, 88.0, 35.0, 0.286, 35.0, 1.0] -- 1.0

125 : [9.0, 72.0, 78.0, 25.0, 0.0, 31.6, 0.28, 38.0, 0.0] -- 0.0

126 : [5.0, 168.0, 64.0, 0.0, 0.0, 32.9, 0.135, 41.0, 1.0] -- 1.0

127 : [4.0, 115.0, 72.0, 0.0, 0.0, 28.9, 0.376, 46.0, 1.0] -- 0.0

128 : [8.0, 197.0, 74.0, 0.0, 0.0, 25.9, 1.191, 39.0, 1.0] -- 1.0

129 : [1.0, 172.0, 68.0, 49.0, 579.0, 42.4, 0.702, 28.0, 1.0] -- 1.0

130 : [3.0, 173.0, 84.0, 33.0, 474.0, 35.7, 0.258, 22.0, 1.0] -- 1.0

131 : [2.0, 94.0, 68.0, 18.0, 76.0, 26.0, 0.561, 21.0, 0.0] -- 0.0

132 : [8.0, 151.0, 78.0, 32.0, 210.0, 42.9, 0.516, 36.0, 1.0] -- 1.0

133 : [1.0, 95.0, 82.0, 25.0, 180.0, 35.0, 0.233, 43.0, 1.0] -- 0.0

134 : [2.0, 99.0, 0.0, 0.0, 0.0, 22.2, 0.108, 23.0, 0.0] -- 0.0

135 : [0.0, 189.0, 104.0, 25.0, 0.0, 34.3, 0.435, 41.0, 1.0] -- 1.0

136 : [2.0, 83.0, 66.0, 23.0, 50.0, 32.2, 0.497, 22.0, 0.0] -- 0.0

137 : [4.0, 117.0, 64.0, 27.0, 120.0, 33.2, 0.23, 24.0, 0.0] -- 0.0

138 : [0.0, 95.0, 80.0, 45.0, 92.0, 36.5, 0.33, 26.0, 0.0] -- 0.0

139 : [2.0, 134.0, 70.0, 0.0, 0.0, 28.9, 0.542, 23.0, 1.0] -- 0.0

140 : [1.0, 135.0, 54.0, 0.0, 0.0, 26.7, 0.687, 62.0, 0.0] -- 0.0

`

141 : [5.0, 86.0, 68.0, 28.0, 71.0, 30.2, 0.364, 24.0, 0.0] -- 0.0

142 : [8.0, 74.0, 70.0, 40.0, 49.0, 35.3, 0.705, 39.0, 0.0] -- 0.0

143 : [0.0, 124.0, 56.0, 13.0, 105.0, 21.8, 0.452, 21.0, 0.0] -- 0.0

144 : [7.0, 136.0, 90.0, 0.0, 0.0, 29.9, 0.21, 50.0, 0.0] -- 1.0

145 : [7.0, 114.0, 76.0, 17.0, 110.0, 23.8, 0.466, 31.0, 0.0] -- 0.0

146 : [3.0, 158.0, 70.0, 30.0, 328.0, 35.5, 0.344, 35.0, 1.0] -- 1.0

147 : [0.0, 123.0, 88.0, 37.0, 0.0, 35.2, 0.197, 29.0, 0.0] -- 0.0

148 : [0.0, 84.0, 82.0, 31.0, 125.0, 38.2, 0.233, 23.0, 0.0] -- 0.0

149 : [0.0, 145.0, 0.0, 0.0, 0.0, 44.2, 0.63, 31.0, 1.0] -- 1.0

150 : [4.0, 99.0, 72.0, 17.0, 0.0, 25.6, 0.294, 28.0, 0.0] -- 0.0

151 : [3.0, 80.0, 0.0, 0.0, 0.0, 0.0, 0.174, 22.0, 0.0] -- 0.0

152 : [6.0, 166.0, 74.0, 0.0, 0.0, 26.6, 0.304, 66.0, 0.0] -- 1.0

153 : [5.0, 110.0, 68.0, 0.0, 0.0, 26.0, 0.292, 30.0, 0.0] -- 0.0

154 : [3.0, 84.0, 72.0, 32.0, 0.0, 37.2, 0.267, 28.0, 0.0] -- 0.0

155 : [10.0, 75.0, 82.0, 0.0, 0.0, 33.3, 0.263, 38.0, 0.0] -- 0.0

156 : [0.0, 180.0, 90.0, 26.0, 90.0, 36.5, 0.314, 35.0, 1.0] -- 1.0

157 : [8.0, 120.0, 78.0, 0.0, 0.0, 25.0, 0.409, 64.0, 0.0] -- 0.0

158 : [0.0, 139.0, 62.0, 17.0, 210.0, 22.1, 0.207, 21.0, 0.0] -- 0.0

159 : [9.0, 91.0, 68.0, 0.0, 0.0, 24.2, 0.2, 58.0, 0.0] -- 0.0

160 : [2.0, 91.0, 62.0, 0.0, 0.0, 27.3, 0.525, 22.0, 0.0] -- 0.0

161 : [13.0, 76.0, 60.0, 0.0, 0.0, 32.8, 0.18, 41.0, 0.0] -- 1.0

162 : [3.0, 124.0, 80.0, 33.0, 130.0, 33.2, 0.305, 26.0, 0.0] -- 0.0

`

163 : [6.0, 114.0, 0.0, 0.0, 0.0, 0.0, 0.189, 26.0, 0.0] -- 0.0

164 : [3.0, 87.0, 60.0, 18.0, 0.0, 21.8, 0.444, 21.0, 0.0] -- 0.0

165 : [1.0, 86.0, 66.0, 52.0, 65.0, 41.3, 0.917, 29.0, 0.0] -- 0.0

166 : [4.0, 84.0, 90.0, 23.0, 56.0, 39.5, 0.159, 25.0, 0.0] -- 0.0

167 : [5.0, 187.0, 76.0, 27.0, 207.0, 43.6, 1.034, 53.0, 1.0] -- 1.0

168 : [4.0, 189.0, 110.0, 31.0, 0.0, 28.5, 0.68, 37.0, 0.0] -- 1.0

169 : [3.0, 84.0, 68.0, 30.0, 106.0, 31.9, 0.591, 25.0, 0.0] -- 0.0

170 : [1.0, 88.0, 62.0, 24.0, 44.0, 29.9, 0.422, 23.0, 0.0] -- 0.0

171 : [1.0, 97.0, 70.0, 40.0, 0.0, 38.1, 0.218, 30.0, 0.0] -- 0.0

172 : [6.0, 99.0, 60.0, 19.0, 54.0, 26.9, 0.497, 32.0, 0.0] -- 0.0

173 : [2.0, 130.0, 96.0, 0.0, 0.0, 22.6, 0.268, 21.0, 0.0] -- 0.0

174 : [2.0, 98.0, 60.0, 17.0, 120.0, 34.7, 0.198, 22.0, 0.0] -- 0.0

175 : [6.0, 108.0, 44.0, 20.0, 130.0, 24.0, 0.813, 35.0, 0.0] -- 0.0

176 : [2.0, 118.0, 80.0, 0.0, 0.0, 42.9, 0.693, 21.0, 1.0] -- 0.0

177 : [6.0, 96.0, 0.0, 0.0, 0.0, 23.7, 0.19, 28.0, 0.0] -- 0.0

178 : [1.0, 124.0, 74.0, 36.0, 0.0, 27.8, 0.1, 30.0, 0.0] -- 0.0

179 : [4.0, 183.0, 0.0, 0.0, 0.0, 28.4, 0.212, 36.0, 1.0] -- 1.0

180 : [1.0, 111.0, 62.0, 13.0, 182.0, 24.0, 0.138, 23.0, 0.0] -- 0.0

181 : [11.0, 138.0, 74.0, 26.0, 144.0, 36.1, 0.557, 50.0, 1.0] -- 1.0

182 : [2.0, 92.0, 76.0, 20.0, 0.0, 24.2, 1.698, 28.0, 0.0] -- 1.0

183 : [6.0, 183.0, 94.0, 0.0, 0.0, 40.8, 1.461, 45.0, 0.0] -- 1.0

184 : [4.0, 94.0, 65.0, 22.0, 0.0, 24.7, 0.148, 21.0, 0.0] -- 0.0

`

185 : [0.0, 102.0, 78.0, 40.0, 90.0, 34.5, 0.238, 24.0, 0.0] -- 0.0

186 : [10.0, 92.0, 62.0, 0.0, 0.0, 25.9, 0.167, 31.0, 0.0] -- 0.0

187 : [0.0, 102.0, 86.0, 17.0, 105.0, 29.3, 0.695, 27.0, 0.0] -- 0.0

188 : [2.0, 157.0, 74.0, 35.0, 440.0, 39.4, 0.134, 30.0, 0.0] -- 1.0

189 : [1.0, 167.0, 74.0, 17.0, 144.0, 23.4, 0.447, 33.0, 1.0] -- 0.0

190 : [0.0, 179.0, 50.0, 36.0, 159.0, 37.8, 0.455, 22.0, 1.0] -- 1.0

191 : [0.0, 107.0, 60.0, 25.0, 0.0, 26.4, 0.133, 23.0, 0.0] -- 0.0

192 : [2.0, 120.0, 54.0, 0.0, 0.0, 26.8, 0.455, 27.0, 0.0] -- 0.0

193 : [2.0, 101.0, 58.0, 35.0, 90.0, 21.8, 0.155, 22.0, 0.0] -- 0.0

194 : [1.0, 199.0, 76.0, 43.0, 0.0, 42.9, 1.394, 22.0, 1.0] -- 1.0

195 : [9.0, 145.0, 80.0, 46.0, 130.0, 37.9, 0.637, 40.0, 1.0] -- 1.0

196 : [1.0, 112.0, 80.0, 45.0, 132.0, 34.8, 0.217, 24.0, 0.0] -- 0.0

197 : [10.0, 111.0, 70.0, 27.0, 0.0, 27.5, 0.141, 40.0, 1.0] -- 0.0

198 : [6.0, 98.0, 58.0, 33.0, 190.0, 34.0, 0.43, 43.0, 0.0] -- 0.0

199 : [9.0, 154.0, 78.0, 30.0, 100.0, 30.9, 0.164, 45.0, 0.0] -- 1.0

200 : [8.0, 91.0, 82.0, 0.0, 0.0, 35.6, 0.587, 68.0, 0.0] -- 1.0

201 : [6.0, 195.0, 70.0, 0.0, 0.0, 30.9, 0.328, 31.0, 1.0] -- 1.0

202 : [9.0, 156.0, 86.0, 0.0, 0.0, 24.8, 0.23, 53.0, 1.0] -- 1.0

203 : [5.0, 136.0, 82.0, 0.0, 0.0, 0.0, 0.64, 69.0, 0.0] -- 0.0

204 : [2.0, 129.0, 74.0, 26.0, 205.0, 33.2, 0.591, 25.0, 0.0] -- 0.0

205 : [1.0, 140.0, 74.0, 26.0, 180.0, 24.1, 0.828, 23.0, 0.0] -- 0.0

206 : [13.0, 158.0, 114.0, 0.0, 0.0, 42.3, 0.257, 44.0, 1.0] -- 1.0

`

207 : [7.0, 142.0, 90.0, 24.0, 480.0, 30.4, 0.128, 43.0, 1.0] -- 1.0

208 : [4.0, 118.0, 70.0, 0.0, 0.0, 44.5, 0.904, 26.0, 0.0] -- 0.0

209 : [1.0, 168.0, 88.0, 29.0, 0.0, 35.0, 0.905, 52.0, 1.0] -- 1.0

210 : [2.0, 129.0, 0.0, 0.0, 0.0, 38.5, 0.304, 41.0, 0.0] -- 1.0

211 : [10.0, 115.0, 0.0, 0.0, 0.0, 0.0, 0.261, 30.0, 1.0] -- 0.0

212 : [2.0, 93.0, 64.0, 32.0, 160.0, 38.0, 0.674, 23.0, 1.0] -- 0.0

213 : [5.0, 126.0, 78.0, 27.0, 22.0, 29.6, 0.439, 40.0, 0.0] -- 0.0

214 : [3.0, 102.0, 74.0, 0.0, 0.0, 29.5, 0.121, 32.0, 0.0] -- 0.0

215 : [4.0, 83.0, 86.0, 19.0, 0.0, 29.3, 0.317, 34.0, 0.0] -- 0.0

216 : [1.0, 149.0, 68.0, 29.0, 127.0, 29.3, 0.349, 42.0, 1.0] -- 0.0

217 : [5.0, 117.0, 86.0, 30.0, 105.0, 39.1, 0.251, 42.0, 0.0] -- 0.0

218 : [1.0, 111.0, 94.0, 0.0, 0.0, 32.8, 0.265, 45.0, 0.0] -- 0.0

219 : [4.0, 112.0, 78.0, 40.0, 0.0, 39.4, 0.236, 38.0, 0.0] -- 0.0

220 : [0.0, 141.0, 84.0, 26.0, 0.0, 32.4, 0.433, 22.0, 0.0] -- 0.0

221 : [2.0, 175.0, 88.0, 0.0, 0.0, 22.9, 0.326, 22.0, 0.0] -- 0.0

222 : [2.0, 106.0, 56.0, 27.0, 165.0, 29.0, 0.426, 22.0, 0.0] -- 0.0

223 : [2.0, 99.0, 60.0, 17.0, 160.0, 36.6, 0.453, 21.0, 0.0] -- 0.0

224 : [1.0, 102.0, 74.0, 0.0, 0.0, 39.5, 0.293, 42.0, 1.0] -- 0.0

225 : [11.0, 120.0, 80.0, 37.0, 150.0, 42.3, 0.785, 48.0, 1.0] -- 1.0

226 : [3.0, 102.0, 44.0, 20.0, 94.0, 30.8, 0.4, 26.0, 0.0] -- 0.0

227 : [1.0, 81.0, 74.0, 41.0, 57.0, 46.3, 1.096, 32.0, 0.0] -- 1.0

228 : [8.0, 154.0, 78.0, 32.0, 0.0, 32.4, 0.443, 45.0, 1.0] -- 1.0

`

229 : [7.0, 137.0, 90.0, 41.0, 0.0, 32.0, 0.391, 39.0, 0.0] -- 1.0

230 : [6.0, 190.0, 92.0, 0.0, 0.0, 35.5, 0.278, 66.0, 1.0] -- 1.0

231 : [10.0, 101.0, 76.0, 48.0, 180.0, 32.9, 0.171, 63.0, 0.0] -- 1.0

Accuracy: 74.45887445887446%

`

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.

Source Code:

```
import numpy as np
import math
import matplotlib.pyplot as plt
import csv
def get_binomial_log_likelihood(obs,probs):
    """ Return the (log)likelihood of obs, given the probs"""
    # Binomial Distribution Log PDF
    # ln (pdf)      = Binomial Coeff * product of probabilities
    # ln[f(x|n, p)] =   comb(N,k)    * num_heads*ln(pH) + (N-num_heads) * ln(1-pH)
    N = sum(obs);#number of trials
    k = obs[0] # number of heads
    binomial_coeff = math.factorial(N) / (math.factorial(N-k) * math.factorial(k))
    prod_probs = obs[0]*math.log(probs[0]) + obs[1]*math.log(1-probs[0])
    log_lik = binomial_coeff + prod_probs
    return log_lik
# 1st:  Coin B, {HTTTHHTHTH}, 5H,5T
# 2nd:  Coin A, {HHHHTHHHHH}, 9H,1T
# 3rd:  Coin A, {HTHHHHHTHH}, 8H,2T
# 4th:  Coin B, {HTHTTTHHTT}, 4H,6T
# 5th:  Coin A, {THHHTHHHTH}, 7H,3T
# so, from MLE: pA(heads) = 0.80 and pB(heads)=0.45
data=[]
with open("cluster.csv") as tsv:
    for line in csv.reader(tsv):
        data=[int(i) for i in line]

# represent the experiments
head_counts = np.array(data)
tail_counts = 10-head_counts
experiments = list(zip(head_counts,tail_counts))
# initialise the pA(heads) and pB(heads)
pA_heads = np.zeros(100); pA_heads[0] = 0.60
pB_heads = np.zeros(100); pB_heads[0] = 0.50
# E-M begins!
delta = 0.001
j = 0 # iteration counter
improvement = float('inf')
while (improvement>delta):
    expectation_A = np.zeros((len(experiments),2), dtype=float)
    expectation_B = np.zeros((len(experiments),2), dtype=float)
    for i in range(0,len(experiments)):
        e = experiments[i] # i'th experiment
          # loglikelihood of e given coin A:
        ll_A = get_binomial_log_likelihood(e,np.array([pA_heads[j],1-pA_heads[j]]))
```

`

```
        # loglikelihood of e given coin B
        ll_B = get_binomial_log_likelihood(e,np.array([pB_heads[j],1-pB_heads[j]]))
        # corresponding weight of A proportional to likelihood of A , ex. .45
        weightA = math.exp(ll_A) / ( math.exp(ll_A) + math.exp(ll_B) )
        # corresponding weight of B proportional to likelihood of B , ex. .55
        weightB = math.exp(ll_B) / ( math.exp(ll_A) + math.exp(ll_B) )
        expectation_A[i] = np.dot(weightA, e) #multiply weightA * e .45xNo. of heads and 45xNo. of tails for coin A
        expectation_B[i] = np.dot(weightB, e) #multiply weightB * e .45xNo. of heads and 45xNo. of Tails for coin B
    pA_heads[j+1] = sum(expectation_A)[0] / sum(sum(expectation_A)); #summing up the data no. of heads and tails for coin A
    pB_heads[j+1] = sum(expectation_B)[0] / sum(sum(expectation_B)); #summing up the data no. of heads and tails for coin B
    #checking the improvement to maximise the accuracy.
    improvement = ( max( abs(np.array([pA_heads[j+1],pB_heads[j+1]]) -
                np.array([pA_heads[j],pB_heads[j]]) )) )
    print(np.array([pA_heads[j+1],pB_heads[j+1]]) -
                np.array([pA_heads[j],pB_heads[j]]) )
    j = j+1
plt.figure();
plt.plot(range(0,j),pA_heads[0:j])#for plotting the graph coin A
plt.plot(range(0,j),pB_heads[0:j])#for plotting the graph coin B
plt.show()
```

Output:

```
[ 0.00796672 -0.09125939]

[ 0.04620638 -0.05680878]

[ 0.02203957 -0.02519619]

[ 0.00533685 -0.00675812]

[ 0.00090446 -0.00162885]

[ 6.34794565e-05 -4.42987679e-04]
```

`

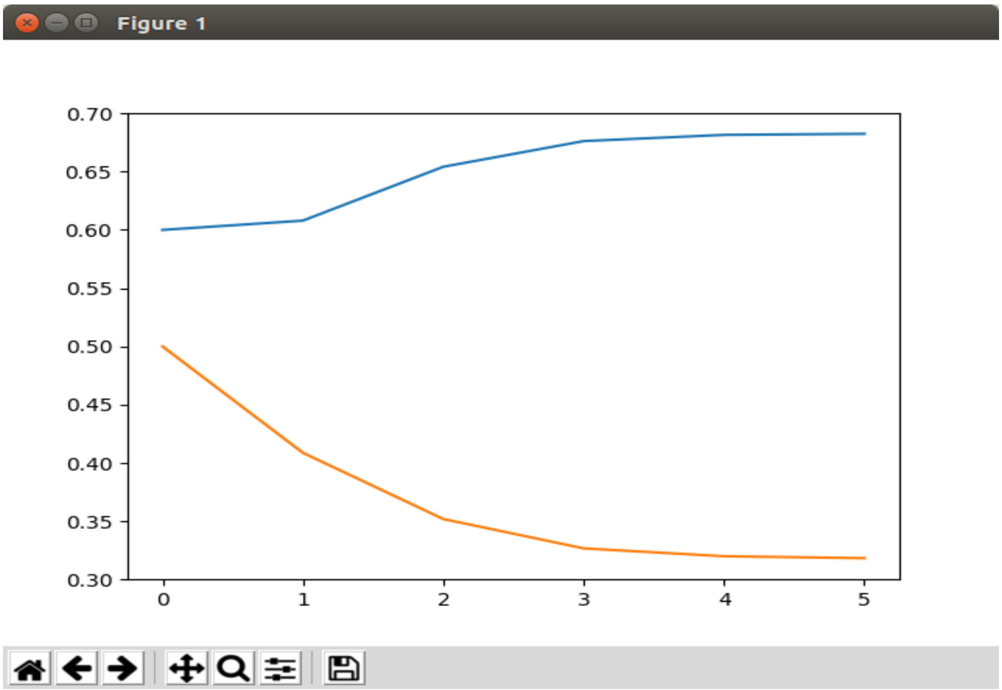


Figure 1
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
1
2
3
4
5

`

K-Means:

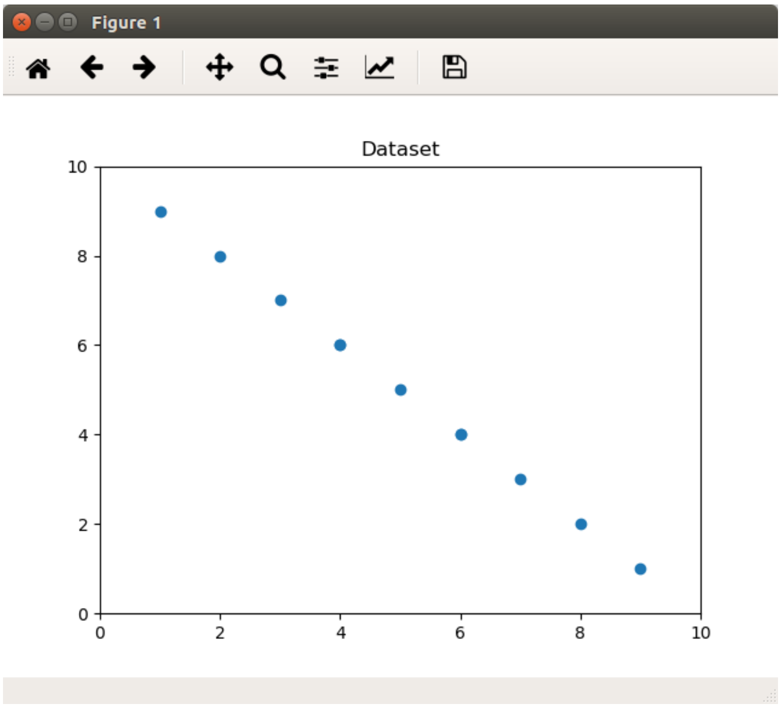
 # clustering dataset

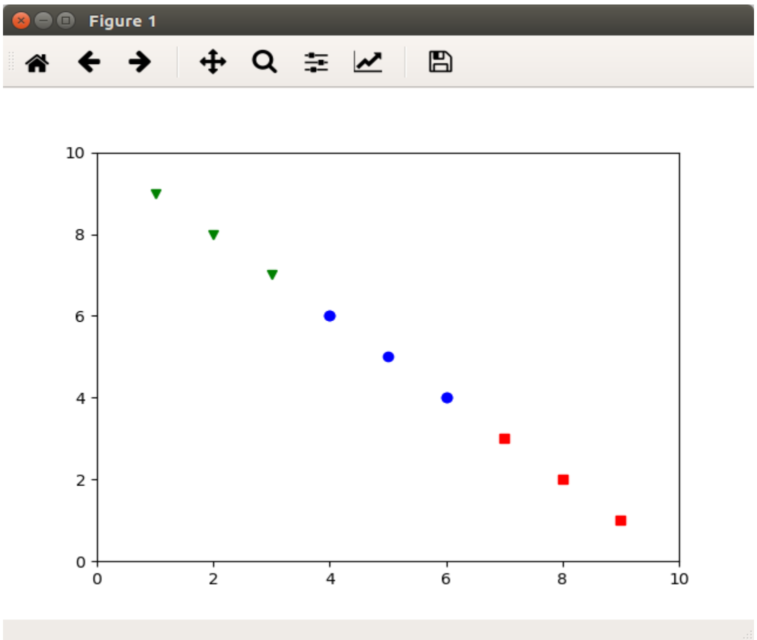
```
from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import csv
data=[]
ydata=[]
with open("cluster.csv") as tsv:
    for line in csv.reader(tsv):
        data=[int(i) for i in line]
        ydata=[10-int(i) for i in line]

x1 = np.array(data)#np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array(ydata)#np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])
print(x1)
plt.plot()
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()
# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
# KMeans algorithm
K = 3
kmeans_model = KMeans(n_clusters=K).fit(X)
plt.plot()
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')
    plt.xlim([0, 10])
    plt.ylim([0, 10])
plt.show()
```

`

Output:





`

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.

Source Code:

```
#import numpy as np
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('iris.csv')
#dataset.groupby('species').size()
#Dividing data into features and labels
feature_columns = ['sepal_length', 'sepal_width', 'petal_length','petal_width']
X = dataset[feature_columns].values
y = dataset['species'].values
"""
KNeighborsClassifier does not accept string labels.
We need to use LabelEncoder to transform them into numbers.
Iris-setosa correspond to 0,
Iris-versicolor correspond to 1 and
Iris-virginica correspond to 2.
"""
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
#Spliting dataset into training set and test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3)
# Fitting the model
classifier.fit(X_train, y_train)
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print("y_pred    y_test")
for i in range(len(y_pred)):
    print(y_pred[i], "    ", y_test[i])
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

Input csv file:

iris_data.csv

`

Output:

y_pred	*y_test*
2	*2*
1	*1*
0	*0*
2	*2*
0	*0*
2	*2*
0	*0*
1	*1*
1	*1*
1	*1*
2	*2*
1	*1*
1	*1*
1	*1*
2	*1*
0	*0*
1	*1*
1	*1*
0	*0*
0	*0*
2	*2*
1	*1*
0	*0*

`

0	*0*
2	*2*
0	*0*
0	*0*
1	*1*
1	*1*
0	*0*

Confusion Matrix:

[[11 0 0]

[0 12 1]

[0 0 6]]

Accuracy of our model is equal 96.67 %.

`

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.

Source Code:

```
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)
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('colA')
plt.ylabel('colB')
```

Input csv file:

LR.csv

Output: