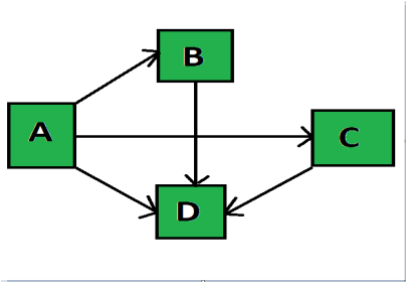
PROGRAM 1

Implement A search Algorithm*

```
class  Graph:
   def __init__(self, adjac_lis):
      self.adjac_lis = adjac_lis
   def get_neighbors(self, v):
      return self.adjac_lis[v]
   # This is heuristic function which is having equal values for all nodes
   def  h(self, n):
      H = {
         'A': 1,
         'B': 1,
         'C': 1,
         'D': 1
      }
      return H[n]
   def a_star_algorithm(self, start, stop):
      # In this open_lst is a lisy of nodes which have been visited, but who's
      # neighbours haven't all been always inspected, It starts off with the start
 #node
      # And closed_lst is a list of nodes which have been visited
      # and who's neighbors have been always inspected
      open_lst = set([start])
      closed_lst = set([])

      # poo has present distances from start to all other nodes
      # the default value is +infinity
      poo = {}
      poo[start] = 0

      # par contains an adjac mapping of all nodes
      par = {}
      par[start] = start

      while len(open_lst) > 0:
         n = None

         # it will find a node with the lowest value of f() -
         for v in open_lst:
            if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
               n = v;

         if n == None:
            print('Path does not exist!')
            return None

         # if the current node is the stop
         # then we start again from start
```

```
        if n == stop:
            reconst_path = []

            while par[n] != n:
                reconst_path.append(n)
                n = par[n]

            reconst_path.append(start)

            reconst_path.reverse()

            print('Path found: {}'.format(reconst_path))
            return reconst_path

        # for all the neighbors of the current node do
        for (m, weight) in self.get_neighbors(n):
          # if the current node is not presentin both open_lst and closed_lst
            # add it to open_lst and note n as it's par
            if m not in open_lst and m not in closed_lst:
                open_lst.add(m)
                par[m] = n
                poo[m] = poo[n] + weight

            # otherwise, check if it's quicker to first visit n, then m
            # and if it is, update par data and poo data
            # and if the node was in the closed_lst, move it to open_lst
            else:
                if poo[m] > poo[n] + weight:
                    poo[m] = poo[n] + weight
                    par[m] = n

                    if m in closed_lst:
                        closed_lst.remove(m)
                        open_lst.add(m)

        # remove n from the open_lst, and add it to closed_lst
        # because all of his neighbors were inspected
        open_lst.remove(n)
        closed_lst.add(n)

    print('Path does not exist!')
    return None

adjac_lis = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
graph1.a_star_algorithm('A', 'D')
```



PROGRAM 3

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

```
import csv
a = [ ]
print("\n The Given Training Data Set \n")

with open('ws.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append (row)
        print(row)
num_attributes = len(a[0])-1

print("\n The initial value of hypothesis: ")
S = ['0'] * num_attributes
G = ['?'] * num_attributes
print ("\n The most specific hypothesis S0 : [0,0,0,0,0,0]\n")
print (" \n The most general hypothesis G0 : [?,?,?,?,?,?]\n")

# Comparing with First Training Example
for j in range(0,num_attributes):
       S[j] = a[0][j];

# Comparing with Remaining Training Examples of Given Data Set

print("\n Candidate Elimination algorithm  Hypotheses Version Space Computation\n")
temp=[]

for i in range(0,len(a)):
    print("-------------------------------------------------------------------------------")
    if a[i][num_attributes]=='Yes':
        for j in range(0,num_attributes):
            if a[i][j]!=S[j]:
                S[j]='?'

        for j in range(0,num_attributes):
            for k  in range(1,len(temp)):
                if temp[k][j]!= '?' and temp[k][j] !=S[j]:
                    del temp[k]

        print(" For Training Example No :{0} the hypothesis is S{0}  ".format(i+1),S)
        if (len(temp)==0):
            print(" For Training Example No :{0} the hypothesis is G{0} ".format(i+1),G)
        else:
```

```
        print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)

    if a[i][num_attributes]=='No':
        for j in range(0,num_attributes):
            if S[j] != a[i][j] and S[j]!= '?':
                G[j]=S[j]
                temp.append(G)
                G = ['?'] * num_attributes

        print(" For Training Example No :{0} the hypothesis is S{0} ".format(i+1),S)
        print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
```

Output

The Given Training Data Set

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

The initial value of hypothesis:

The most specific hypothesis S0 : [0,0,0,0,0,0]

The most general hypothesis G0 : [?,?,?,?,?,?]

Candidate Elimination algorithm Hypotheses Version Space Computation

```
--------------------------------------------------------------------------------
(' For Training Example No :1 the hypothesis is S1  ', ['Sunny', 'Warm', 'Normal', 'Strong', 'W
arm', 'Same'])
(' For Training Example No :1 the hypothesis is G1 ', ['?', '?', '?', '?', '?', '?'])
--------------------------------------------------------------------------------
(' For Training Example No :2 the hypothesis is S2  ', ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'S
ame'])
(' For Training Example No :2 the hypothesis is G2 ', ['?', '?', '?', '?', '?', '?'])
--------------------------------------------------------------------------------
(' For Training Example No :3 the hypothesis is S3 ', ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'S
ame'])
(' For Training Example No :3 the hypothesis is G3', [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '
?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']])
--------------------------------------------------------------------------------
(' For Training Example No :4 the hypothesis is S4  ', ['Sunny', 'Warm', '?', 'Strong', '?', '?'])
(' For Training Example No :4 the hypothesis is G4', [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '
?', '?', '?', '?']])
```

PROGRAM 4

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import sys
import numpy as np
from numpy import *
import csv

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

def read_data(filename):

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


    return (metadata, traindata)

def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col]) # get unique values in particular column

   count = np.zeros((items.shape[0], 1), dtype=np.int32)   #number of row = number of values

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1
    #count has the data of number of times each value is present in

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

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

```
            pos += 1

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

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

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

    for x in range(items.shape[0]):

        counts[x] = sum(S == items[x]) / (S.size)



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

def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)
    #item is the unique value and dict is the data corresponding to it
    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])


    total_entropy = entropy(data[:, -1])


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

    return total_entropy

def create_node(data, metadata):


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

```
    gains = np.zeros((data.shape[1] - 1, 1))
    #size of gains= number of attribute to calculate gain


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

    split = np.argmax(gains)


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


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

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

    return node

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

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

        print(empty(level), node.answer.item(0).decode("utf-8"))
        return

    print(empty(level), node.attribute)

    for value, n in node.children:
        print(empty(level + 1), value.tobytes().decode("utf-8"))
        print_tree(n, level + 2)


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

tennis.csv

outlook	temp	humidity	windy	play
sunny	hot	high	Weak	no
sunny	hot	high	Strong	no
overcast	hot	high	Weak	yes
rainy	mild	high	Weak	yes
rainy	cool	normal	Weak	yes
rainy	cool	normal	Strong	no
overcast	cool	normal	Strong	yes
sunny	mild	high	Weak	no
sunny	cool	normal	Weak	yes
rainy	mild	normal	Weak	yes
sunny	mild	normal	Strong	yes
overcast	mild	high	Strong	yes
overcast	hot	normal	Weak	yes
rainy	mild	high	Strong	no

PROGRAM 5

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

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
y = y/100
#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
#Variable initialization
epoch=10000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
#Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)
#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to e
rror
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and currentlayerop
    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output:\n" ,output)
```

Output:

Input:
[[2. 9.]
 [1. 5.]
 [3. 6.]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

('Predicted Output:\n', array([[0.89678563],
       [0.87179214],
       [0.89995692]]))

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

```
import numpy as np
import math
import csv

def read_data(filename):

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

   return (metadata, traindata)

def splitDataset(dataset, splitRatio):        #splits dataset to training set and test set base
d on split ratio
            trainSize = int(len(dataset) * splitRatio)
            trainSet = []
            testset = list(dataset)
            i=0
            while len(trainSet) < trainSize:
               trainSet.append(testset.pop(i))
            return [trainSet, testset]

def classify(data,test):

   total_size = data.shape[0]
   print("training data size=",total_size)
   print("test data sixe=",test.shape[0])
   target=np.unique(data[:,-1])
   count = np.zeros((target.shape[0]), dtype=np.int32)
   prob = np.zeros((target.shape[0]), dtype=np.float32)
   print("target   count  probability")
   for y in range(target.shape[0]):
      for x in range(data.shape[0]):
         if data[x,data.shape[1]-1] == target[y]:
            count[y] += 1
      prob[y]=count[y]/total_size    # comptes the probability of target
      print(target[y],"\t",count[y],"\t",prob[y])

   prob0 = np.zeros((test.shape[1]-1), dtype=np.float32)
```

```
    prob1 = np.zeros((test.shape[1]-1), dtype=np.float32)
    accuracy=0
    print("Instance prediction taget")
    for t in range(test.shape[0]):
        for k in range(test.shape[1]-1):  # for each attribute in column
            count1=count0=0
            for j in range(data.shape[0]):
                if test[t,k]== data[j,k] and data[j,data.shape[1]-1]== target[0]:
                    count0+=1
                elif test[t,k]== data[j,k] and data[j,data.shape[1]-1]== target[1]:
                    count1+=1
            prob0[k]= count0/count[0]       #Find no probability of each attribute
            prob1[k]= count1/count[1]      #Find yes probability of each attribute


        probno=prob[0]
        probyes=prob[1]
        for i in range(test.shape[1]-1):
            probno=probno*prob0[i]
            probyes=probyes*prob1[i]


        if probno>probyes:     # prediction
            predict='no'
        else:
            predict='yes'
        print(t+1,"\t",predict,"\t    ",test[t,test.shape[1]-1])


        if predict== test[t,test.shape[1]-1]:     # computing accuracy
            accuracy+=1

    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")

    return

metadata, traindata = read_data("tennis.csv")
splitRatio = 0.6
trainingset, testset = splitDataset(traindata, splitRatio)
training=np.array(trainingset)
testing=np.array(testset)
print("------------------Training Data--------------------")
print(trainingset)
print("--------------------Test Data--------------------")
print(testset)
classify(training,testing)
```

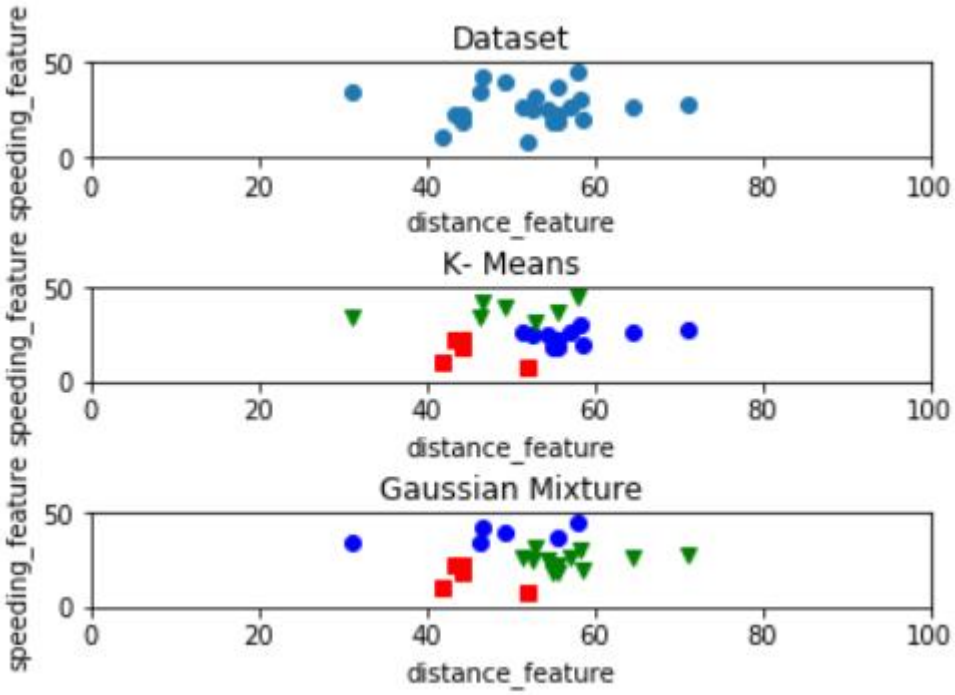
PROGRAM 7

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

```
from sklearn.cluster import KMeans

#from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.mixture import GaussianMixture

data=pd.read_csv("clusterdata.csv")
df1=pd.DataFrame(data)
print(df1)
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values

X=np.matrix(list(zip(f1,f2)))
plt.plot(1)
plt.subplot(511)
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')
plt.scatter(f1,f2)


colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# create new plot and data for K- means algorithm
plt.plot(2)
ax=plt.subplot(513)
kmeans_model = KMeans(n_clusters=3).fit(X)

for i, l in enumerate(kmeans_model.labels_):
    fig1=plt.plot(f1[i], f2[i], color=colors[l],marker=markers[l])

plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('K- Means')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')
```

```
# create new plot and data for gaussian mixture
plt.plot(3)
plt.subplot(515)
gmm=GaussianMixture(n_components=3).fit(X)
labels= gmm.predict(X)
for i, l in enumerate(labels):
   plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l])
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Gaussian Mixture')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')


plt.show()
```

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


Dataset
K- Means
Gaussian Mixture
speeding_feature
distance_feature
0
20
40
60
80
100
50
0

PROGRAM 8

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

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

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

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

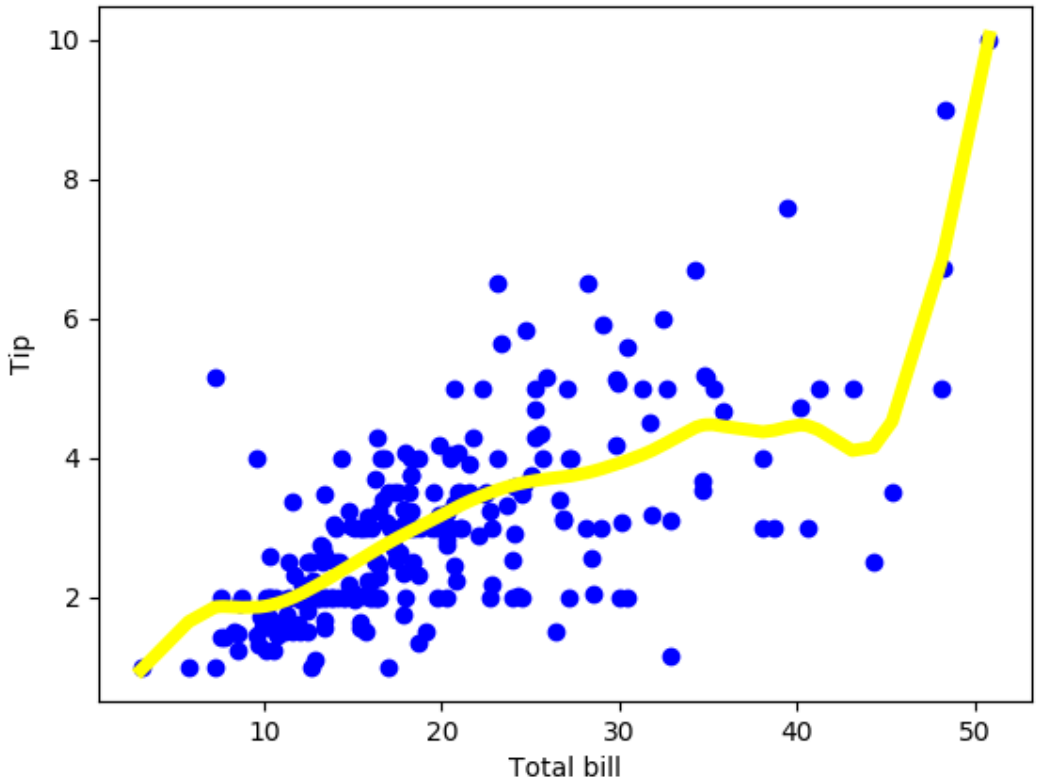
Output:

```
Confusion matrix is as follows
[[15  0  0]
 [ 0 15  1]
 [ 0  0 14]]
Accuracy Matrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      0.94      0.97        16
           2       0.93      1.00      0.97        14

avg / total       0.98      0.98      0.98        45
```

Program 09
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt



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

#preparing and add 1 in bill
mColA = np.mat(ColA)  # mat treats array as matrix
mColB = np.mat(ColB)
m= np.shape(mColB)[1]
#print(m)
one = np.ones((1,m) ,dtype=int)
#print(one)

#Horizontal Stacking
X= np.hstack((one.T,mColA.T))
#print(X.shape)
tem=np.hstack((mColA.T))
#print(tem)
 #Gaussiaon Kernel
def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    #print(weights)
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights
 #Define Local Weights
def localWeight(point,xmat,ymat,k):
    wt = kernel(point,xmat,k)
    W = (X.T*(wt*X)).I*(X.T*(wt*ymat.T))
    return W
```

```
#Final prediction value
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


#prediction value
ypred = localWeightRegression(X,mColB,2)

#Plot Regression Lines
Xvalues=X.copy()
print(Xvalues)
Xvalues.sort(axis=0)
plt.scatter(ColA, ColB, color='blue')

plt.plot(Xvalues[:,1], ypred[X[:,1].argsort(0)], color='yellow', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```



Program 02
IMPLEMENT AO* SEARCH ALGORITHM