**PROGRAM1:FINDS**

**AIM:**

Implement and demonstrate the FINDS algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

**PROGRAM:**

```
import csv
a = []
with open('enjoysport.csv', 'r') as csvfile:
   for row in csv.reader(csvfile):
      a.append(row)
   print(a)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
 if a[i][num_attribute] == 'yes':
  for j in range(0, num_attribute):
     if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
        hypothesis[j] = a[i][j]
     else:
        hypothesis[j] = '?'
 print("\n The hypothesis for the training instance {} is :\n" .format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

**RESULT**

```
[['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport'], ['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 total number of training instances are :  5

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 1 is :
 ['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 2 is :
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

 The hypothesis for the training instance 3 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

 The hypothesis for the training instance 4 is :
```

['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 5 is :
['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is
['sunny', 'warm', '?', 'strong', '?', '?']

## PROGRAM:2 CE ALGORITHM

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

```
import numpy as np
import pandas as pd

data = pd.read_csv('D:/GEO/BE COURSES/2022 dec/LAB/DATASET/enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")
```

**RESULT:**

Instances are:
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
['rainy' 'cold' 'high' 'strong' 'warm' 'change']
['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are: ['yes' 'yes' 'no' 'yes']

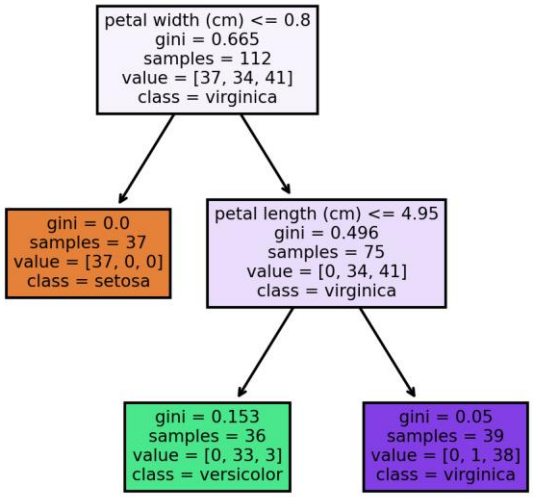
## PROGRAM :3 DECISION TREE

**AIM: 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.**

```
@author: GEOMOL GEORGE
"""
#Import Libraries
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
#from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier
#from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn import tree
#Load the Dataset
import pandas as pd
from sklearn.datasets import load_iris
data = load_iris()
#data= pd.read_csv("D:/GEO/BE COURSES/LAB/DATASET/pima-indians-diabetes.csv")
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
#Splitting Data into Training and Test Sets
X_train, X_test, Y_train, Y_test = train_test_split(df[data.feature_names], df['target'],
random_state=0)
#Scikit-learn 4-Step Modeling Pattern
# Step 1: Import the model you want to use
# This was already imported earlier in the notebook so commenting out
#from sklearn.tree import DecisionTreeClassifier
# Step 2: Make an instance of the Model
clf = DecisionTreeClassifier(max_depth = 2,random_state = 0)
# Step 3: Train the model on the data
```

```
clf.fit(X_train, Y_train)
# Step 4: Predict labels of unseen (test) data
# Not doing this step in the tutorial
# clf.predict(X_test)
#How to Visualize Decision Trees using Matplotlib
tree.plot_tree(clf);

fn=['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']
cn=['setosa', 'versicolor', 'virginica']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename.png')
```

**RESULT:**




# PROGRAM4:BACK PROPAGATION

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

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
```

```
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5 #Setting training iterations
lr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)

    #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad

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

    print ("-----------Epoch-", i+1, "Starts----------")
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends----------\n")

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

**RESULT:**

**Input:**
**[[0.66666667 1.       ]**
**[0.33333333 0.55555556]**
**[1.       0.66666667]]**
**Actual Output:**
**[[0.92]**
**[0.86]**
**[0.89]]**
**Predicted Output:**
**[[0.85680382]**
**[0.83717506]**
**[0.86055174]]**
**----------Epoch- 3 Ends----------**

**----------Epoch- 4 Starts----------**
**Input:**
**[[0.66666667 1.       ]**
**[0.33333333 0.55555556]**
**[1.       0.66666667]]**
**Actual Output:**
**[[0.92]**
**[0.86]**
**[0.89]]**
**Predicted Output:**
**[[0.85716867]**
**[0.83754295]**
**[0.86091463]]**
**----------Epoch- 4 Ends----------**

**----------Epoch- 5 Starts----------**
**Input:**
**[[0.66666667 1.       ]**
**[0.33333333 0.55555556]**
**[1.       0.66666667]]**
**Actual Output:**
**[[0.92]**
**[0.86]**
**[0.89]]**
**Predicted Output:**
**[[0.85752859]**
**[0.83790597]**
**[0.86127258]]**
**----------Epoch- 5 Ends----------**

**Input:**
**[[0.66666667 1.       ]**
**[0.33333333 0.55555556]**
**[1.       0.66666667]]**
**Actual Output:**
**[[0.92]**
**[0.86]**

**[0.89]]**
**Predicted Output:**
**[[0.85752859]**
**[0.83790597]**
**[0.86127258]]**

# PROGRAM5: CLASSIFICATION OF IRIS (IN-BUILT) USING KNN

AIM: **Write a program for Implementation of K-Nearest Neighbors (K-NN) in Python**

```
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()

x = iris.data
y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print(' classification_report)
print(classification_report(y_test,y_pred))
```

**RESULT**
**Confusion Matrix**
**[[17 0 0]**
**[ 0 10 1]**
**[ 0 3 14]]**
**classification_report**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **0** | **1.00** | **1.00** | **1.00** | **17** |
| **1** | **0.77** | **0.91** | **0.83** | **11** |
| **2** | **0.93** | **0.82** | **0.87** | **17** |

**accuracy 0.91 45**
**macro avg 0.90 0.91 0.90 45**
**weighted avg 0.92 0.91 0.91 45**

**Accuracy Score: 0.9111111111111111**

## PROGRAM 6: NAÏVE BAYES

**AIM: Write a program to implement Naïve Bayes algorithm in python and to display the results using confusion matrix and accuracy. Java/Python ML library classes can be used for this problem.**

```
import numpy as np
import pandas as pd
#Importing the dataset

dataset = pd.read_csv("breastcancer.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

#Naive Bayes classifier model
GaussianNB(priors=None, var_smoothing=1e-09)

#Display the results (confusion matrix and accuracy)

from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

**RESULT:**
**[[99 8]**
**[ 2 62]]**

# PROGRAM :7-LINEAR REGRESSION

AIM:Write a program to implement Linear Regression (LR) algorithm in python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
# Load the dataset
dataset = pd.read_csv('Salary_Data.csv')

# Split the dataset into independent variables (X) and dependent variable (y)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create an instance of the Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Predict the salaries for the test data
y_pred = model.predict(X_test)

#model good or not
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)


#Visualising the Training set results Here scatter plot is used to visualize the results.

plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, model.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

**RESULT**


## PROGRAM:8 LOGISTIC REGRESSION (Brestcancerdataset)

**AIM: Write a program to implement Logistic Regression (LR) algorithm in python**

```
import numpy as np
import pandas as pd

#"Importing the dataset



# divide the dataset into concepts and targets. Store the concepts into X and targets into y.
dataset = pd.read_csv("D:/GEO/BE COURSES/2022 dec/LAB/DATASET/breastcancer.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

#Splitting the dataset into the Training set and Test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 2)

#Feature Scaling

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.linear_model import LogisticRegression
```

```
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
#Logistic Regression (LR) classifier model

#Display the results (confusion matrix and accuracy)

from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy Score:confusion matrix')
accuracy_score(y_test, y_pred)
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

**RESULT**

**[[117 8]**
**[ 6 74]]**
**Accuracy Score:confusion matrix**
**Accuracy: 0.9317073170731708**

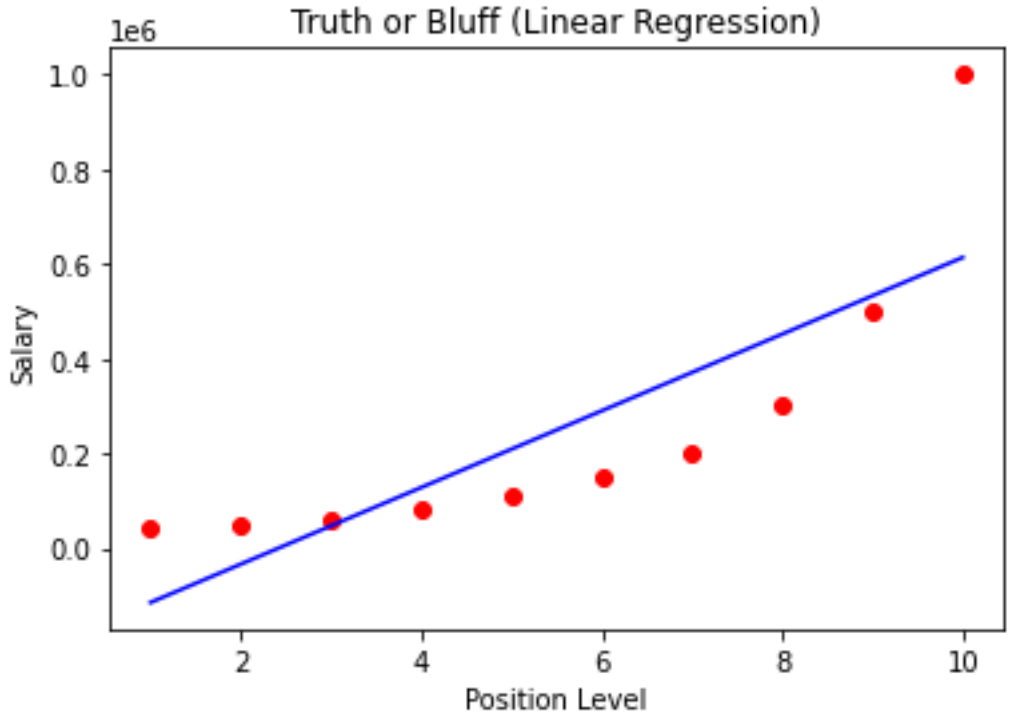
## PROGRAM 8 POLYNOMIAL REGRESSION

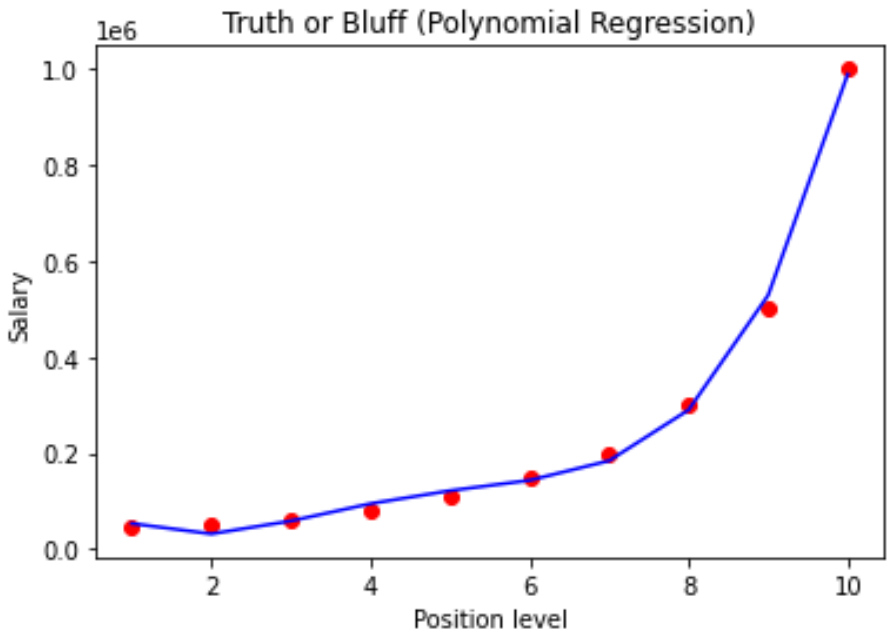
AIM: **Implementation Of Linear And Polynomial Regression In Python**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values


from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)

#Linear Regression classifier model
#(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)


lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
#Polynomial Regression classifier model
#LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
#Visualising the Polynomial Regression results

plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

**RESULT**






# PROGRAM9 EM algorithm

**AIM: Python Program to Implement Estimation & MAximization Algorithm**

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

# Read the dataset
dataset = pd.read_csv("D:/GEO/BE COURSES/2022 dec/LAB/IRIS.csv")

X = dataset.iloc[:, :-1]
label = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
y = dataset.iloc[:, -1].map(label)  # Convert class labels to integer values

plt.figure(figsize=(14, 7))
colormap = np.array(['red', 'lime', 'black'])

# REAL PLOT
plt.subplot(1, 3, 1)
plt.title('Real')
plt.scatter(X.petal_length, X.petal_width, c=colormap[y])  # Use y as the class indices

gmm = GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm = gmm.predict(X)

# GMM Classification PLOT
plt.subplot(1, 3, 3)
plt.title('GMM Classification')
```

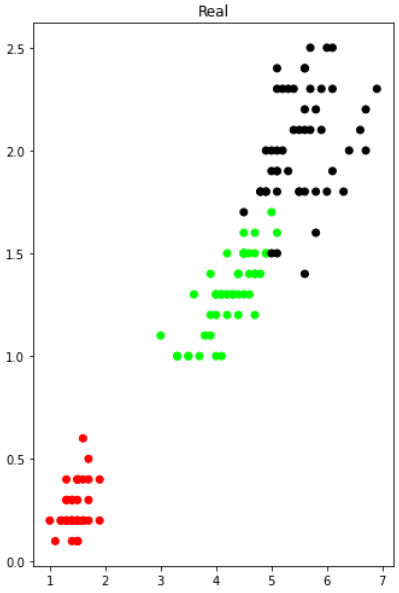
```
plt.scatter(X.petal_length, X.petal_width, c=colormap[y_cluster_gmm])  # Use y_cluster_gmm for colors

# Print metrics
print('The accuracy score of GMM:', metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of GMM:\n', metrics.confusion_matrix(y, y_cluster_gmm))

plt.tight_layout()
plt.show()
```

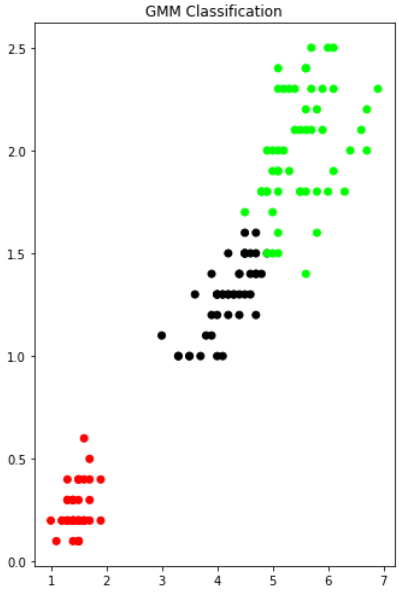
RESULT:

The accuracy score of GMM: 0.36666666666666664

The Confusion matrix of GMM:

```
 [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
```




# PROGRAM 10:PERCEPTRON IRIS CLASSIFCATION

AIM:Write python code for PERCEPTRON IRIS CLASSIFCATION

```
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
```

```
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

ppn = Perceptron(eta0=0.1, random_state=1)
ppn.fit(X_train_std, y_train)

y_pred = ppn.predict(X_test_std)

print('Accuracy:' % accuracy_score(y_test, y_pred))
```

## RESULT

## Accuracy:

## perceptron IRIS classification Accuracy: 0.978