PROGRAMMING ASSIGNMENT3

Question1 and Question2 both are based on same function and interlinked to each other so ouput are provided at last.

In [1]:

```
#Importing all necessary libraries to implement our model
import numpy as np                                  # Python library for data
import pandas as pd                                 # Python library for numerical c
omputation
import matplotlib.pyplot as plt                     # Python library for plotting da
ta
from sklearn.preprocessing import StandardScaler
```

In [2]:

```
class Logistic_Regression(object):

#Below function Intialize the value for learning rate and number of interation in l
ogistic regression.
#The learning rate is a hyperparameter that controls change in the model in respons
e to the estimated error each time the model weights are updated.
    def __init__(Logistic_reg, Learning_rate=0.01, number_of_iteration=100):
        Logistic_reg.Learning_rate = Learning_rate
        Logistic_reg.number_iteration = number_of_iteration

#Below function Computing the sigmoid value of the given paramters for this logisti
c regrssion model.
    def sigm_fun(Logistic_reg, x):
        output = 1 / (1 + np.exp(-x))
        return output

#Below function computing the cost value of the logistic regression model.
    def cost_fun(Logistic_reg,h,model_weights, y): # The fuctions calculates the co
st value
        m = len(y)
        cost = (1 / m) * (np.sum(-y.T.dot(np.log(h)) - (1 - y).T.dot(np.log(1 - h
))))
        return cost

#After getting the cost value of the function using gradient descent method below f
unction computing the model weights(Theta).
    def gradient_fun(Logistic_reg,X,h,model_weights,y,m):
        gradient_output = np.dot(X.T, (h - y)) / m
        model_weights -= Logistic_reg.Learning_rate * gradient_output
        return model_weights

#We need to use predict data to build our model so below function first computing t
he optimal weights for the model based on predicted data.
#Every time function takes different values.
    def fit(Logistic_reg, X, y):
        print("Running...Please Wait")
        Logistic_reg.model_weights = []
        Logistic_reg.cost = []
        X = np.insert(X, 0, 1, axis=1)
        m = len(y)
        for i in np.unique(y):
            y_onevsall = np.where(y == i, 1, 0)
            model_weights = np.zeros(X.shape[1])
            cost = []
            for _ in range(Logistic_reg.number_iteration):
                z = X.dot(model_weights)
                h = Logistic_reg.sigm_fun(z)
                model_weights = Logistic_reg.gradient_fun(X,h,model_weights,y_onevs
all,m)
                cost.append(Logistic_reg.cost_fun(h,model_weights,y_onevsall))
            Logistic_reg.model_weights.append((model_weights, i))
            Logistic_reg.cost.append((cost,i))
        return Logistic_reg,Logistic_reg.model_weights

#After calcultaing the model weights value we need to classify the invidual feature
```

```
s based on our needs.
#Below function classfied the individal features for predicted value.
    def predict_fun(Logistic_reg, X):
        X = np.insert(X, 0, 1, axis=1)
        X_predicted = [max((Logistic_reg.sigm_fun(i.dot(model_weights)), c) for mod
el_weights, c in Logistic_reg.model_weights)[1] for i in X ]
        return X_predicted

#Below function comparing the values between the predicted label and actual label.I
t gives model Accuracy.
# To find the model performace based on the data.
    def score_fun(Logistic_reg,X, y):
        score = sum(Logistic_reg.predict_fun(X) == y) / len(y)
        return score

#Below funtion plotting cost function graph for its different value.
#This function plots converge garph.
    def plt_cost(Logistic_reg,costh):
        for cost,c in costh   :
                plt.plot(range(len(cost)),cost,'g')
                plt.title("Gradient Graph for species -" + str(c) +" vs All")
                plt.xlabel("Number of Iterations")
                plt.ylabel("Cost(Error)")
                plt.show()
```

In [3]:

```
#Reading data from CSV file
Dataset = pd.read_csv('Fish.csv')
print(Dataset)
```

```
    Species  Weight  Length1  Length2  Length3   Height   Width
0     Bream   242.0     23.2     25.4     30.0  11.5200  4.0200
1     Bream   290.0     24.0     26.3     31.2  12.4800  4.3056
2     Bream   340.0     23.9     26.5     31.1  12.3778  4.6961
3     Bream   363.0     26.3     29.0     33.5  12.7300  4.4555
4     Bream   430.0     26.5     29.0     34.0  12.4440  5.1340
..      ...     ...      ...      ...      ...      ...     ...
154   Smelt    12.2     11.5     12.2     13.4   2.0904  1.3936
155   Smelt    13.4     11.7     12.4     13.5   2.4300  1.2690
156   Smelt    12.2     12.1     13.0     13.8   2.2770  1.2558
157   Smelt    19.7     13.2     14.3     15.2   2.8728  2.0672
158   Smelt    19.9     13.8     15.0     16.2   2.9322  1.8792

[159 rows x 7 columns]
```

In [4]:

```
#Split the data in X and Y.
#Y lable take output as species and X label takes all other columns from the datase
t.
y_Dataset = Dataset['Species'].values
X = Dataset.drop(['Species'],axis=1).values
```

In [5]:

```
#Below Function normalizing data.
Normalizer = StandardScaler()
X= Normalizer.fit_transform(X)
#print(X)
```

In [6]:

```
#Library for spliting the data into train data and test data.
from sklearn.model_selection import train_test_split
#X_train.shape
```

In [7]:

```
#y_test.shape
```

In [8]:

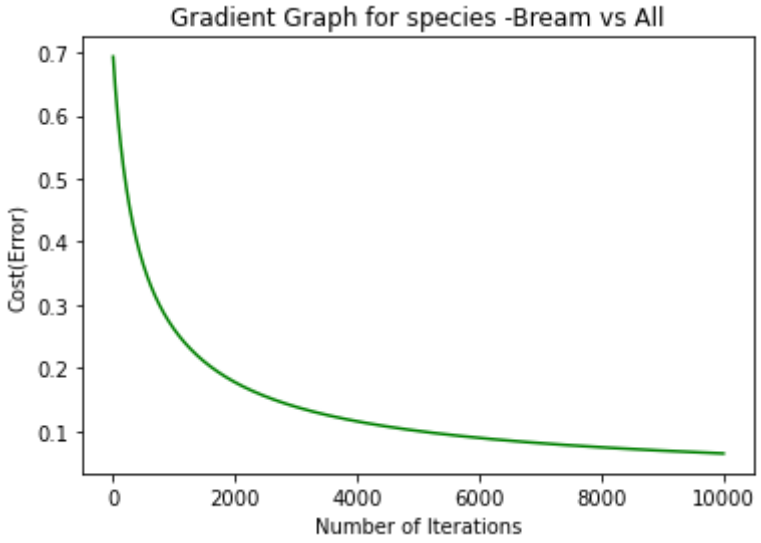
```
#Train onevsall model.
for _ in range (6):
#Here we are taking Weight,Lenght1,Length2,Length3,Width and Height as an input fea
tures and Every time model plots onevsall graph for one inputfeature and one specie
s.
    X_train,X_test,y_train,y_test = train_test_split(X,y_Dataset,test_size = 0.6)
    log_reg, model_coefficients= Logistic_Regression(number_of_iteration=10000).fit
(X_train, y_train)
    prediction_output = log_reg.predict_fun(X_test)
    Train_Accuracy = log_reg.score_fun(X_train,y_train)
    Test_Accuracy = log_reg.score_fun(X_test,y_test)

    print("\n")
    print("Train accuracy of the model is: ",Train_Accuracy)
    print("Test accuracy of the model is: ",Test_Accuracy)
    print("\n")
```

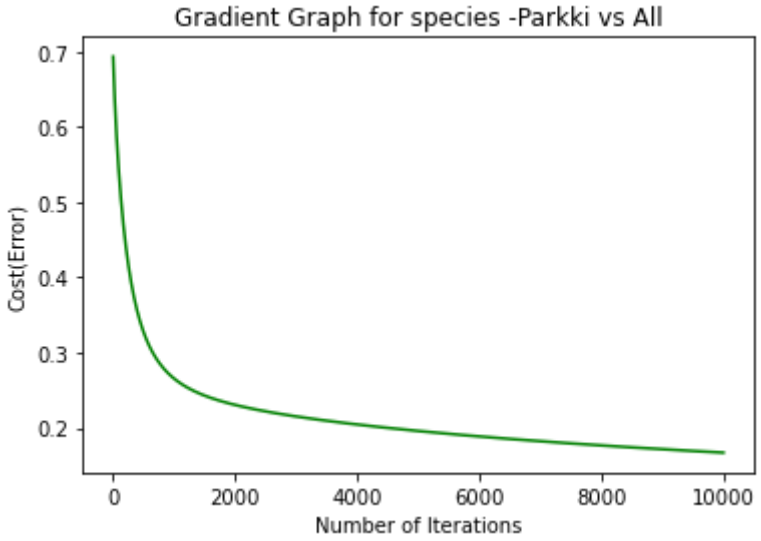
```
Running...Please Wait


Train accuracy of the model is:  0.7936507936507936
Test accuracy of the model is:  0.8125


Running...Please Wait


Train accuracy of the model is:  0.8412698412698413
Test accuracy of the model is:  0.6666666666666666


Running...Please Wait


Train accuracy of the model is:  0.8095238095238095
Test accuracy of the model is:  0.7395833333333334


Running...Please Wait


Train accuracy of the model is:  0.8571428571428571
Test accuracy of the model is:  0.7708333333333334


Running...Please Wait


Train accuracy of the model is:  0.8888888888888888
Test accuracy of the model is:  0.7395833333333334


Running...Please Wait


Train accuracy of the model is:  0.8253968253968254
Test accuracy of the model is:  0.78125
```

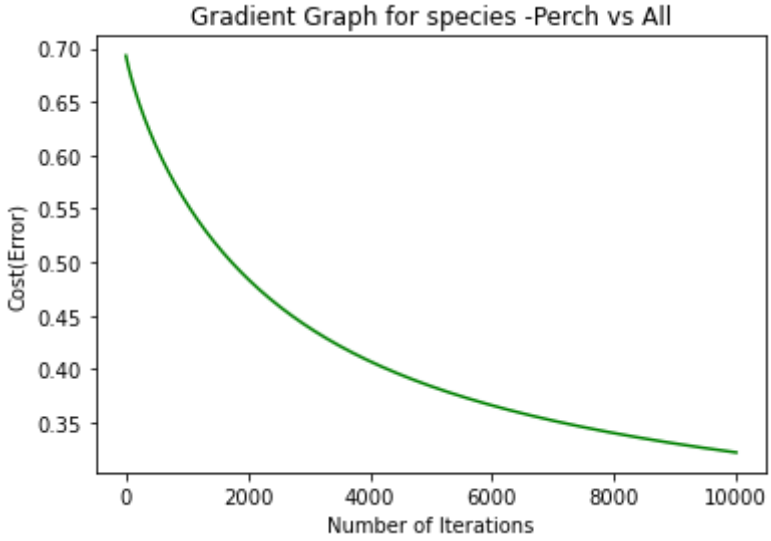
In [9]:

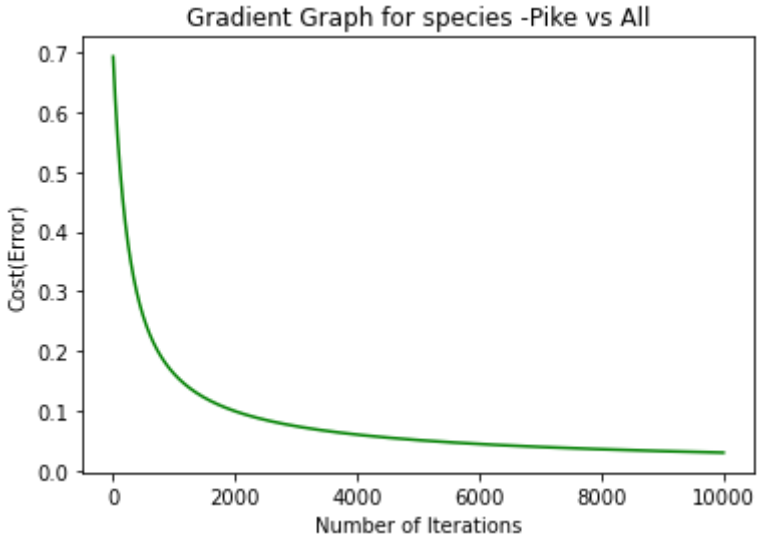
```
#Plotting onevsall graph for each species.
print("Plotted OneVsAll Graph For Each Sepcies")
log_reg.plt_cost(log_reg.cost)
```

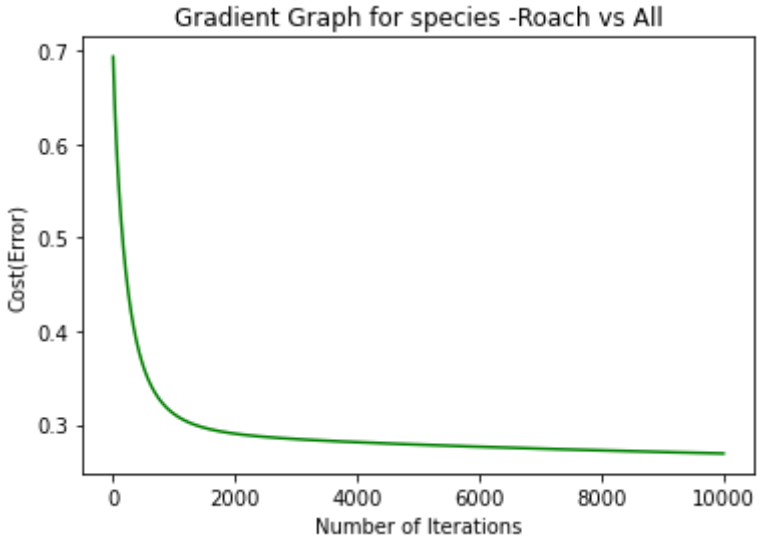
```
Plotted OneVsAll Graph For Each Sepcies
```

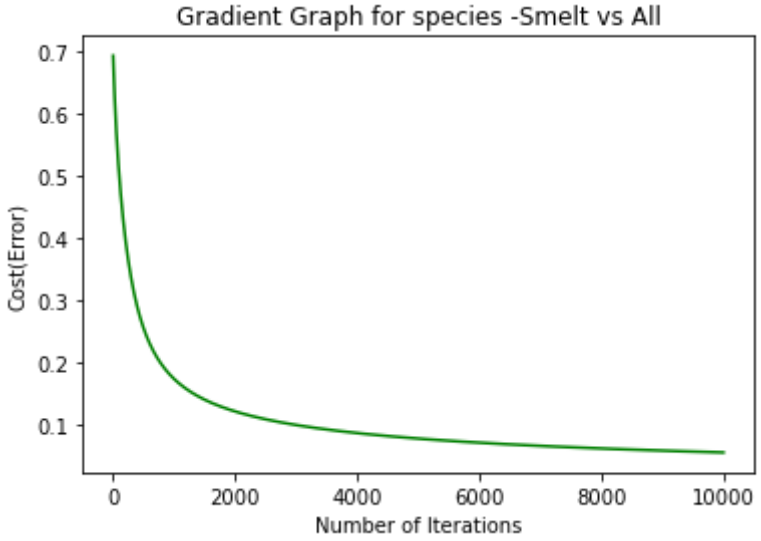


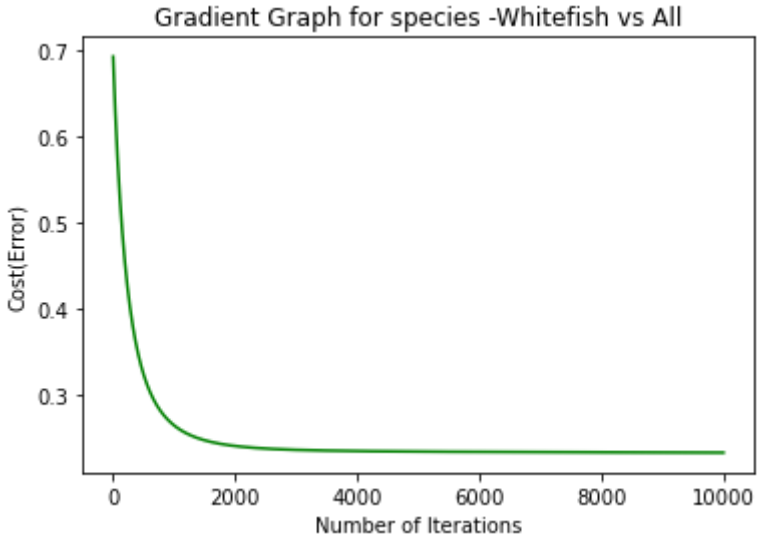



Gradient Graph for species -Perch vs All
Cost(Error)
Number of Iterations
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
2000
4000
6000
8000
10000


Gradient Graph for species -Pike vs All
Cost(Error)
Number of Iterations
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2000
4000
6000
8000
10000


Gradient Graph for species -Roach vs All
Cost(Error)
Number of Iterations
0.7
0.6
0.5
0.4
0.3
0
2000
4000
6000
8000
10000


Gradient Graph for species -Smelt vs All
Cost(Error)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2000
4000
6000
8000
10000
Number of Iterations


Gradient Graph for species -Whitefish vs All
Cost(Error)
0.7
0.6
0.5
0.4
0.3
0
2000
4000
6000
8000
10000
Number of Iterations

In [10]:

```
#Print model coefficient for each
print("Model Coefficient:", model_coefficients)
```

```
Model Coefficient: [(array([-2.90104452, -0.60416603, -0.57471651, -0.4
460134 ,  0.73660962,
        4.49669526, -0.97473469]), 'Bream'), (array([-3.37296206, -0.90
761546, -0.56605202, -0.61741354, -0.58236414,
        2.00456865, -0.83108324]), 'Parkki'), (array([-0.71949988,  0.0
412865 , -0.31716079, -0.14418072, -1.98554205,
       -2.61627026,  4.20970488]), 'Perch'), (array([-3.71933656, -0.40
240045,  1.34052759,  1.27852989,  1.39567178,
       -1.66697047, -1.39107341]), 'Pike'), (array([-2.49992086, -1.368
1404 ,  0.0174022 , -0.16603317,  0.23253066,
       -0.32214485,  0.53671199]), 'Roach'), (array([-4.39021389,  0.80
508244, -0.25367776, -0.39618064, -0.33054178,
       -0.95420767, -1.40742034]), 'Smelt'), (array([-2.71337069, -0.15
291736, -0.10520693, -0.02739548,  0.06935928,
       -0.13318833,  0.40208171]), 'Whitefish')]
```

In [11]:

```
#Below Function Gives output for actual value and predicted value.
Output_Prediction = pd.DataFrame({
    'TestValue':y_test,
    'PredictedValue':prediction_output
})
print(Output_Prediction)
```

```
    TestValue PredictedValue
0       Bream          Bream
1       Bream          Bream
2       Smelt          Perch
3       Smelt          Smelt
4       Perch          Perch
..        ...            ...
91      Smelt          Smelt
92  Whitefish          Perch
93       Pike           Pike
94      Perch          Perch
95      Roach          Perch

[96 rows x 2 columns]
```