

# Logistic Regression with non-linear features

## import library

```
In [ ]: import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.colors as colors  
from matplotlib import ticker, cm
```

## load training data

```
In [ ]: fname_data1 = 'assignment_09_data1.txt'  
fname_data2 = 'assignment_09_data2.txt'  
  
data1      = np.genfromtxt(fname_data1, delimiter=',')  
data2      = np.genfromtxt(fname_data2, delimiter=',')  
  
number_data1 = data1.shape[0]  
number_data2 = data2.shape[0]  
  
data1_point    = data1[:, 0:2]  
data1_point_x  = data1_point[:, 0]  
data1_point_y  = data1_point[:, 1]  
data1_label    = data1[:, 2]  
  
data2_point    = data2[:, 0:2]  
data2_point_x  = data2_point[:, 0]  
data2_point_y  = data2_point[:, 1]  
data2_label    = data2[:, 2]  
  
data1_label_class_0    = (data1_label == 0)  
data1_label_class_1    = (data1_label == 1)  
  
data2_label_class_0    = (data2_label == 0)  
data2_label_class_1    = (data2_label == 1)  
  
data1_point_x_class_0  = data1_point_x[data1_label_class_0]  
data1_point_y_class_0  = data1_point_y[data1_label_class_0]  
  
data1_point_x_class_1  = data1_point_x[data1_label_class_1]  
data1_point_y_class_1  = data1_point_y[data1_label_class_1]  
  
data2_point_x_class_0  = data2_point_x[data2_label_class_0]  
data2_point_y_class_0  = data2_point_y[data2_label_class_0]  
  
data2_point_x_class_1  = data2_point_x[data2_label_class_1]  
data2_point_y_class_1  = data2_point_y[data2_label_class_1]  
  
print('shape of point in data1 = ', data1_point.shape)  
print('shape of point in data2 = ', data2_point.shape)  
  
print('shape of label in data1 = ', data1_label.shape)  
print('shape of label in data2 = ', data2_label.shape)  
  
print('data type of point x in data1 = ', data1_point_x.dtype)
```

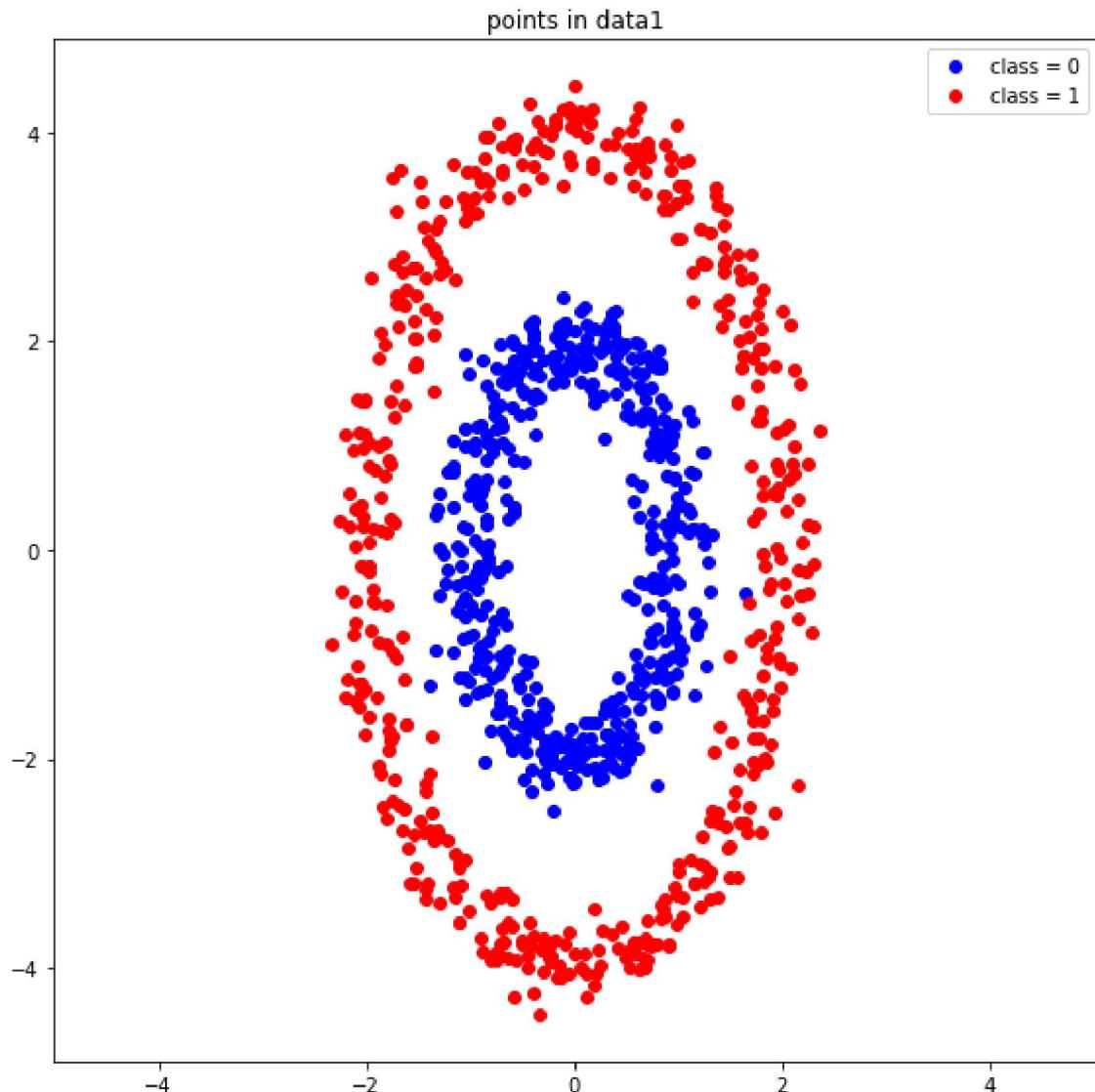
```
print('data type of point y in data1 = ', data1_point_y.dtype)
print('data type of point x in data2 = ', data2_point_x.dtype)
print('data type of point y in data2 = ', data2_point_y.dtype)

shape of point in data1 = (1000, 2)
shape of point in data2 = (1000, 2)
shape of label in data1 = (1000,)
shape of label in data2 = (1000,)
data type of point x in data1 = float64
data type of point y in data1 = float64
data type of point x in data2 = float64
data type of point y in data2 = float64
```

## plot the data

```
In [ ]: f = plt.figure(figsize=(8,8))

plt.title('points in data1')
plt.plot(data1_point_x_class_0, data1_point_y_class_0, 'o', color='blue', label='class 0')
plt.plot(data1_point_x_class_1, data1_point_y_class_1, 'o', color='red', label='class 1')
plt.axis('equal')
plt.legend()
plt.tight_layout()
plt.show()
```

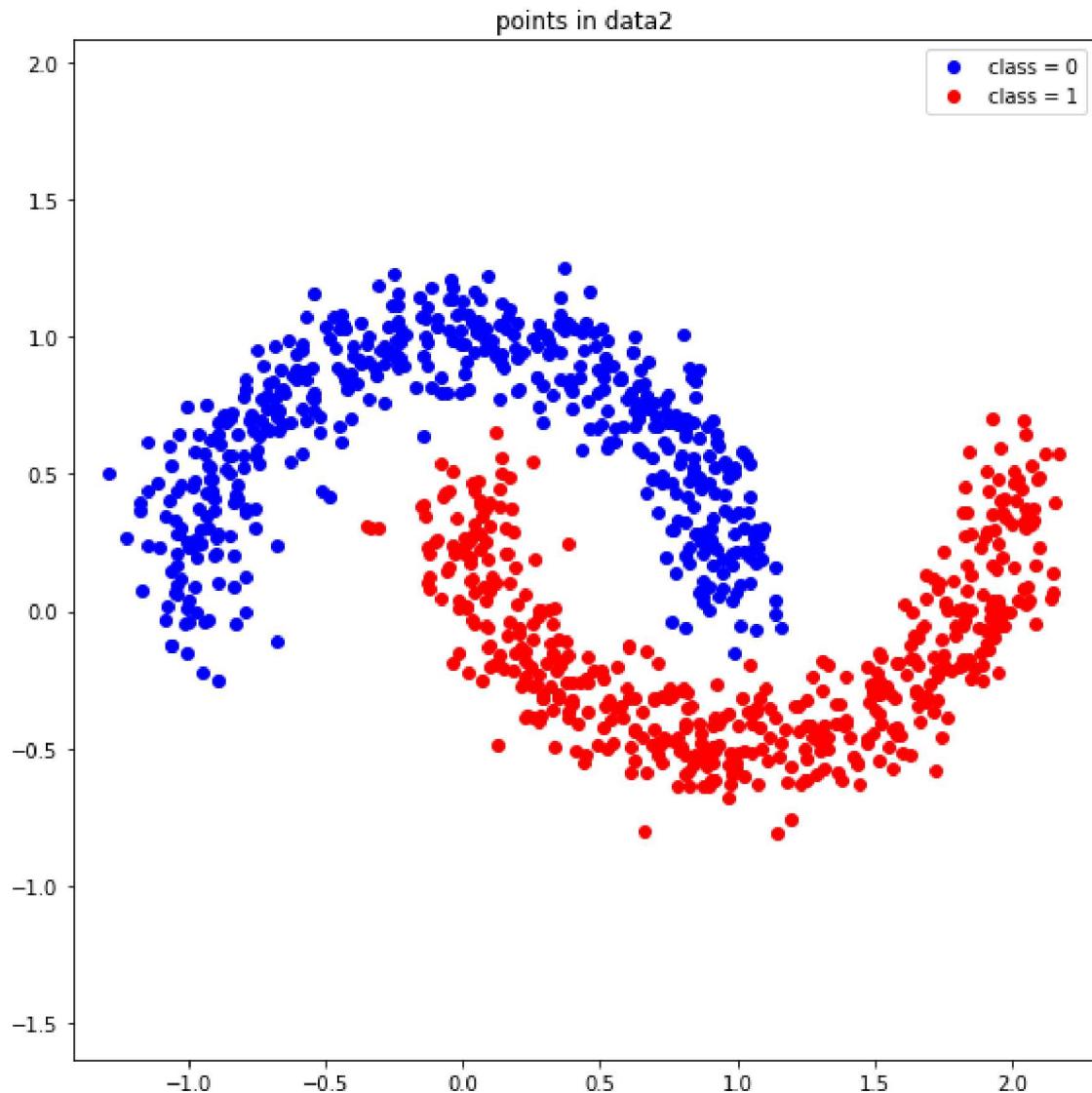


```
In [ ]: f = plt.figure(figsize=(8,8))
```

```

plt.title('points in data2')
plt.plot(data2_point_x_class_0, data2_point_y_class_0, 'o', color='blue', label='class 0')
plt.plot(data2_point_x_class_1, data2_point_y_class_1, 'o', color='red', label='class 1')
plt.axis('equal')
plt.legend()
plt.tight_layout()
plt.show()

```



## define the feature functions

- feature vector is defined by  $(1, f_1(x, y), f_2(x, y), \dots, f_{k-1}(x, y)) \in \mathbb{R}^k$

```

In [ ]: def compute_feature1(point):

    # ++++++#
    # complete the blanks
    #

    x = point[:, 0]
    y = point[:, 1]
    feature = np.array([np.ones(x.shape), np.sqrt(2)*x*y, x**2, y**2,]).T

    #
    # ++++++#

```

```
    return feature
```

```
In [ ]: def compute_feature2(point):  
    # ++++++  
    # complete the blanks  
    #  
  
    x = point[:, 0]  
    y = point[:, 1]  
    feature = np.array([np.ones(x.shape), x, y, x**2, y**2, x**3, y**3, x*y]).T  
    #  
    # ++++++  
  
    return feature
```

## define the linear regression function

- $\theta = (\theta_0, \theta_1, \dots, \theta_{k-1}) \in \mathbb{R}^k$
- feature =  $(1, f_1(x, y), \dots, f_{k-1}(x, y)) \in \mathbb{R}^k$

```
In [ ]: def compute_linear_regression(theta, feature):  
    # ++++++  
    # complete the blanks  
    #  
  
    value = np.inner(theta, feature)  
    #  
    # ++++++  
  
    return value
```

## define sigmoid function with input

- $z \in \mathbb{R}$

```
In [ ]: def sigmoid(z):  
    # ++++++  
    # complete the blanks  
    #  
  
    value = 1 / (1 + np.exp(-z))  
    #  
    # ++++++  
  
    return value
```

## define the logistic regression function

- $\theta = (\theta_0, \theta_1, \dots, \theta_{k-1}) \in \mathbb{R}^k$
- feature =  $(1, f_1(x, y), \dots, f_{k-1}(x, y)) \in \mathbb{R}^k$

```
In [ ]: def compute_logistic_regression(theta, feature):
    # ++++++
    # complete the blanks
    #

    f = compute_linear_regression(theta, feature)
    value = sigmoid(f)

    #
    # +++++

    return value
```

## define the residual function

- $\theta = (\theta_0, \theta_1, \dots, \theta_{k-1}) \in \mathbb{R}^k$
- feature =  $(1, f_1(x, y), \dots, f_{k-1}(x, y)) \in \mathbb{R}^k$
- label =  $l \in \{0, 1\}^k$

```
In [ ]: def compute_residual(theta, feature, label):
    # ++++++
    # complete the blanks
    #

    h = compute_logistic_regression(theta, feature)
    residual = -(label * np.log(h) + ((1 - label) * np.log(1 - h)))

    #
    # +++++

    return residual
```

## define the loss function for the logistic regression

- $\theta = (\theta_0, \theta_1, \dots, \theta_{k-1}) \in \mathbb{R}^k$
- feature =  $(1, f_1(x, y), \dots, f_{k-1}(x, y)) \in \mathbb{R}^k$
- label =  $l \in \{0, 1\}^k$

```
In [ ]: def compute_loss(theta, feature, label):
    # ++++++
    # complete the blanks
    #

    n = feature.shape[0]
    r = compute_residual(theta, feature, label)
    loss = np.sum(r) / n

    #
    # +++++
```

```
    return loss
```

## define the gradient of the loss with respect to the model parameter $\theta$

- $\theta = (\theta_0, \theta_1, \dots, \theta_{k-1}) \in \mathbb{R}^k$
- feature =  $(1, f_1(x, y), \dots, f_{k-1}(x, y)) \in \mathbb{R}^k$
- label =  $l \in \{0, 1\}^k$

```
In [ ]: def compute_gradient(theta, feature, label):  
  
    # ++++++  
    # complete the blanks  
    #  
  
    n = feature.shape[0]  
    h = compute_logistic_regression(theta, feature)  
    gradient = np.sum((h.reshape(n,1) - label.reshape(n,1)) * feature, axis = 0) /  
  
    #  
    # ++++++  
  
    return gradient
```

## compute the accuracy of the prediction for point with a given model parameter

```
In [ ]: def compute_accuracy(theta, feature, label):  
  
    # ++++++  
    # complete the blanks  
    #  
  
    n = feature.shape[0]  
    h = compute_logistic_regression(theta, feature)  
    correct = 0  
  
    for i in range(n) :  
        if (h[i] >= 0.5 and label[i] == 1) or (h[i] < 0.5 and label[i] == 0) :  
            correct = correct + 1  
  
    accuracy = correct / n  
  
    #  
    # ++++++  
  
    return accuracy
```

## initialize the gradient descent algorithm

```
In [ ]: data1_number_iteration = 10000  
data2_number_iteration = 10000  
  
data1_learning_rate = 0.3
```

```

data2_learning_rate = 1.0

data1_number_feature = 4
data2_number_feature = 8

theta1 = np.zeros(data1_number_feature)
theta2 = np.zeros(data2_number_feature)

data1_loss_iteration = np.zeros(data1_number_iteration)
data2_loss_iteration = np.zeros(data2_number_iteration)

data1_accuracy_iteration = np.zeros(data1_number_iteration)
data2_accuracy_iteration = np.zeros(data2_number_iteration)

```

## run the gradient descent algorithm to optimize the loss function with respect to the model parameter

```

In [ ]: for i in range(data1_number_iteration):

    # ++++++
    # complete the blanks
    #
    theta1 = theta1 - (data1_learning_rate * compute_gradient(theta1, compute_f
    loss1 = compute_loss(theta1, compute_feature1(data1_point), data1_label)
    accuracy1 = compute_accuracy(theta1, compute_feature1(data1_point), data1_labe

    #
    # ++++++
    data1_loss_iteration[i] = loss1
    data1_accuracy_iteration[i] = accuracy1

data1_theta_optimal = theta1

```

```

In [ ]: for i in range(data2_number_iteration):

    # ++++++
    # complete the blanks
    #
    theta2 = theta2 - (data2_learning_rate * compute_gradient(theta2, compute_f
    loss2 = compute_loss(theta2, compute_feature2(data2_point), data2_label)
    accuracy2 = compute_accuracy(theta2, compute_feature2(data2_point), data2_labe

    #
    # ++++++
    data2_loss_iteration[i] = loss2
    data2_accuracy_iteration[i] = accuracy2

data2_theta_optimal = theta2

```

---



---

## functions for presenting the results

---

```
In [ ]: def function_result_01():

    print("final loss for data1 = {:.13.10f}".format(data1_loss_iteration[-1]))
```

```
In [ ]: def function_result_02():

    print("final loss for data2 = {:.13.10f}".format(data2_loss_iteration[-1]))
```

```
In [ ]: def function_result_03():

    print("final accuracy for data1 = {:.13.10f}".format(data1_accuracy_iteration[-1]))
```

```
In [ ]: def function_result_04():

    print("final accuracy for data2 = {:.13.10f}".format(data2_accuracy_iteration[-1]))
```

```
In [ ]: def function_result_05():

    plt.figure(figsize=(8,6))
    plt.title('loss for data1')

    plt.plot(data1_loss_iteration, '--', color='red')
    plt.xlabel('iteration')
    plt.ylabel('loss')

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

```
In [ ]: def function_result_06():

    plt.figure(figsize=(8,6))
    plt.title('loss for data2')

    plt.plot(data2_loss_iteration, '--', color='red')
    plt.xlabel('iteration')
    plt.ylabel('loss')

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

```
In [ ]: def function_result_07():

    plt.figure(figsize=(8,6))
    plt.title('accuracy for data1')

    plt.plot(data1_accuracy_iteration, '--', color='red')
    plt.xlabel('iteration')
    plt.ylabel('accuracy')

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

```
In [ ]: def function_result_08():

    plt.figure(figsize=(8,6))
    plt.title('accuracy for data2')

    plt.plot(data2_accuracy_iteration, '--', color='red')
    plt.xlabel('iteration')
    plt.ylabel('accuracy')
```

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

## plot the linear regression values over the 2-dimensional Euclidean space and superimpose the training data

```
In [ ]: def function_result_09():  
  
    plt.figure(figsize=(8,8)) # USE THIS VALUE for the size of the figure  
    plt.title('linear regression values')  
  
    min_x = np.min(data1_point_x)  
    max_x = np.max(data1_point_x)  
    min_y = np.min(data1_point_y)  
    max_y = np.max(data1_point_y)  
  
    X = np.arange(min_x - 0.5, max_x + 0.5, 0.1) # USE THIS VALUE for the range of x  
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1) # USE THIS VALUE for the range of y  
  
    [XX, YY] = np.meshgrid(X, Y)  
  
    # ++++++  
    # complete the blanks  
    #  
    plt.title('linear regression values')  
  
    ZZ = np.array([np.ones(XX.shape), np.sqrt(2)*XX*YY, XX**2, YY**2])  
    linear_regression = data1_theta_optimal[0]*ZZ[0] + data1_theta_optimal[1]*ZZ[1]  
    plt.scatter(XX, YY, c=linear_regression, cmap=plt.cm.bwr)  
    plt.grid()  
  
    classifier = np.where((linear_regression < 0.5) & (linear_regression > -0.5))  
    plt.plot(XX[classifier], YY[classifier], 'x', color='black')  
  
    plt.plot(data1_point_x_class_0, data1_point_y_class_0, '.', color='red', label='red')  
    plt.plot(data1_point_x_class_1, data1_point_y_class_1, '.', color='blue', label='blue')  
    plt.colorbar()  
    plt.clim(-60, 60)  
    #  
    # ++++++  
  
    plt.axis('equal')  
    plt.legend()  
    plt.tight_layout()  
    plt.show()
```

```
In [ ]: def function_result_10():  
  
    plt.figure(figsize=(8,8)) # USE THIS VALUE for the size of the figure  
    plt.title('linear regression values')  
  
    min_x = np.min(data2_point_x)  
    max_x = np.max(data2_point_x)  
    min_y = np.min(data2_point_y)  
    max_y = np.max(data2_point_y)  
  
    X = np.arange(min_x - 0.5, max_x + 0.5, 0.1) # USE THIS VALUE for the range of x  
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1) # USE THIS VALUE for the range of y
```

```

[XX, YY] = np.meshgrid(X, Y)

# ++++++
# complete the blanks
#
plt.title('linear regression values')

ZZ = np.array([np.ones(XX.shape), XX, YY, XX**2, YY**2, XX**3, YY**3, XX*YY])
linear_regression = data2_theta_optimal[0]*ZZ[0] + data2_theta_optimal[1]*ZZ[1]
    + data2_theta_optimal[4]*ZZ[4] + data2_theta_optimal[5]*ZZ[5] + data2_theta_
plt.scatter(XX, YY, c=linear_regression, cmap=plt.cm.bwr)
plt.grid()

classifier = np.where((linear_regression < 0.8) & (linear_regression > -0.8))
plt.plot(XX[classifier], YY[classifier], 'x', color='black')

plt.plot(data2_point_x_class_0, data2_point_y_class_0, '.', color='blue', label=
plt.plot(data2_point_x_class_1, data2_point_y_class_1, '.', color='red', label=
plt.colorbar()
plt.clim(-50, 50)
#
# ++++++
plt.axis('equal')
plt.legend()
plt.tight_layout()
plt.show()

```

## plot the logistic regression values over the 2-dimensional Euclidean space

```

In [ ]: def function_result_11():

    plt.figure(figsize=(8,8)) # USE THIS VALUE for the size of the figure
    plt.title('logistic regression values')

    min_x = np.min(data1_point_x)
    max_x = np.max(data1_point_x)
    min_y = np.min(data1_point_y)
    max_y = np.max(data1_point_y)

    X = np.arange(min_x - 0.5, max_x + 0.5, 0.1) # USE THIS VALUE for the range of x
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1) # USE THIS VALUE for the range of y

    [XX, YY] = np.meshgrid(X, Y)

    # ++++++
    # complete the blanks
    #
    plt.title('logistic regression values')

    ZZ = np.array([np.ones(XX.shape), np.sqrt(2)*XX*YY, XX**2, YY**2])
z = data1_theta_optimal[0]*ZZ[0] + data1_theta_optimal[1]*ZZ[1] + data1_theta_opt
logistic_regression = 1 / (1 + np.exp(-z))
plt.scatter(XX, YY, c=logistic_regression, cmap=plt.cm.bwr)
plt.grid()

plt.plot(data1_point_x_class_0, data1_point_y_class_0, '.', color='red', label=
plt.plot(data1_point_x_class_1, data1_point_y_class_1, '.', color='blue', label=
plt.colorbar()
#

```

```
# ++++++
plt.axis('equal')
plt.legend()
plt.tight_layout()
plt.show()
```

```
In [ ]: def function_result_12():

    plt.figure(figsize=(8,8)) # USE THIS VALUE for the size of the figure
    plt.title('logistic regression values')

    min_x = np.min(data2_point_x)
    max_x = np.max(data2_point_x)
    min_y = np.min(data2_point_y)
    max_y = np.max(data2_point_y)

    X = np.arange(min_x - 0.5, max_x + 0.5, 0.1) # USE THIS VALUE for the range of x
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1) # USE THIS VALUE for the range of y

    [XX, YY] = np.meshgrid(X, Y)

    # ++++++
    # complete the blanks
    #
    plt.title('logistic regression values')

    ZZ = np.array([np.ones(XX.shape), XX, YY, XX**2, YY**2, XX**3, YY**3, XX*YY])
    z = data2_theta_optimal[0]*ZZ[0] + data2_theta_optimal[1]*ZZ[1] + data2_theta_optimal[2]*ZZ[2] + data2_theta_optimal[3]*ZZ[3] + data2_theta_optimal[4]*ZZ[4] + data2_theta_optimal[5]*ZZ[5] + data2_theta_optimal[6]
    logistic_regression = 1 / (1 + np.exp(-z))
    plt.scatter(XX, YY, c=logistic_regression, cmap=plt.cm.bwr)
    plt.grid()

    plt.plot(data2_point_x_class_0, data2_point_y_class_0, '.', color='blue', label='Class 0')
    plt.plot(data2_point_x_class_1, data2_point_y_class_1, '.', color='red', label='Class 1')
    plt.colorbar()
    #
    # ++++++
    plt.axis('equal')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

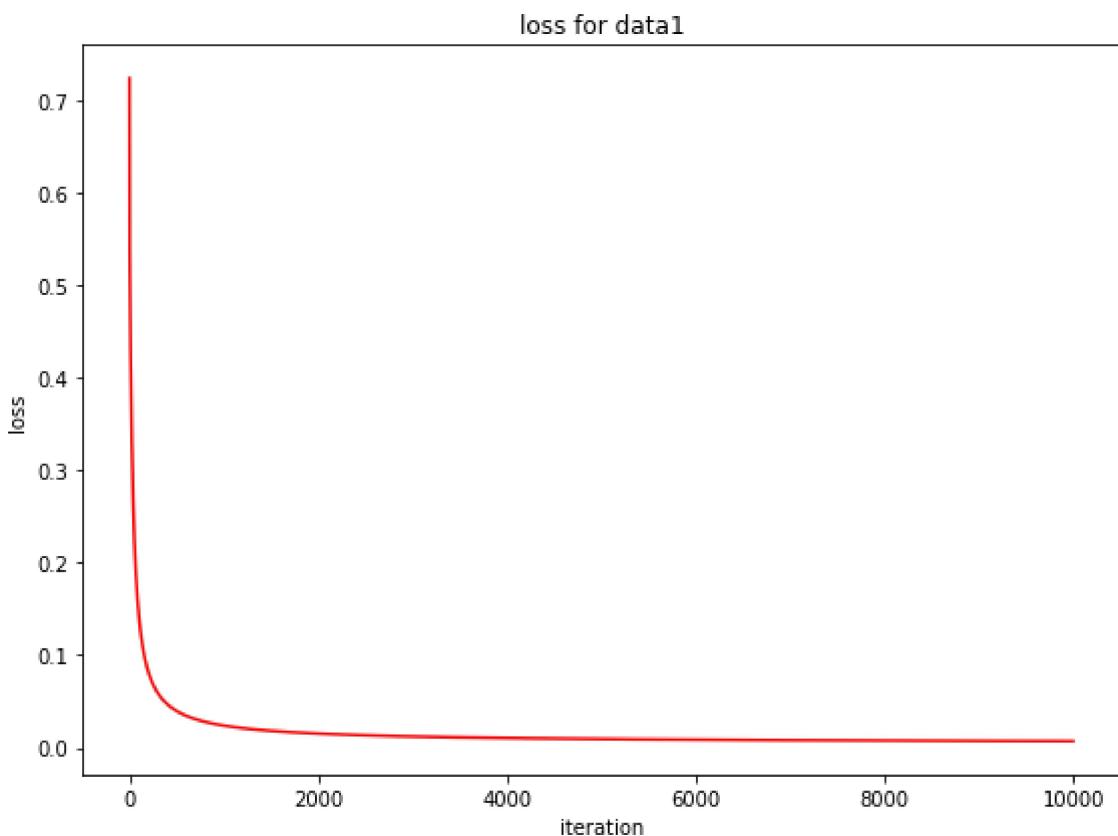
## results

```
In [ ]: number_result = 12

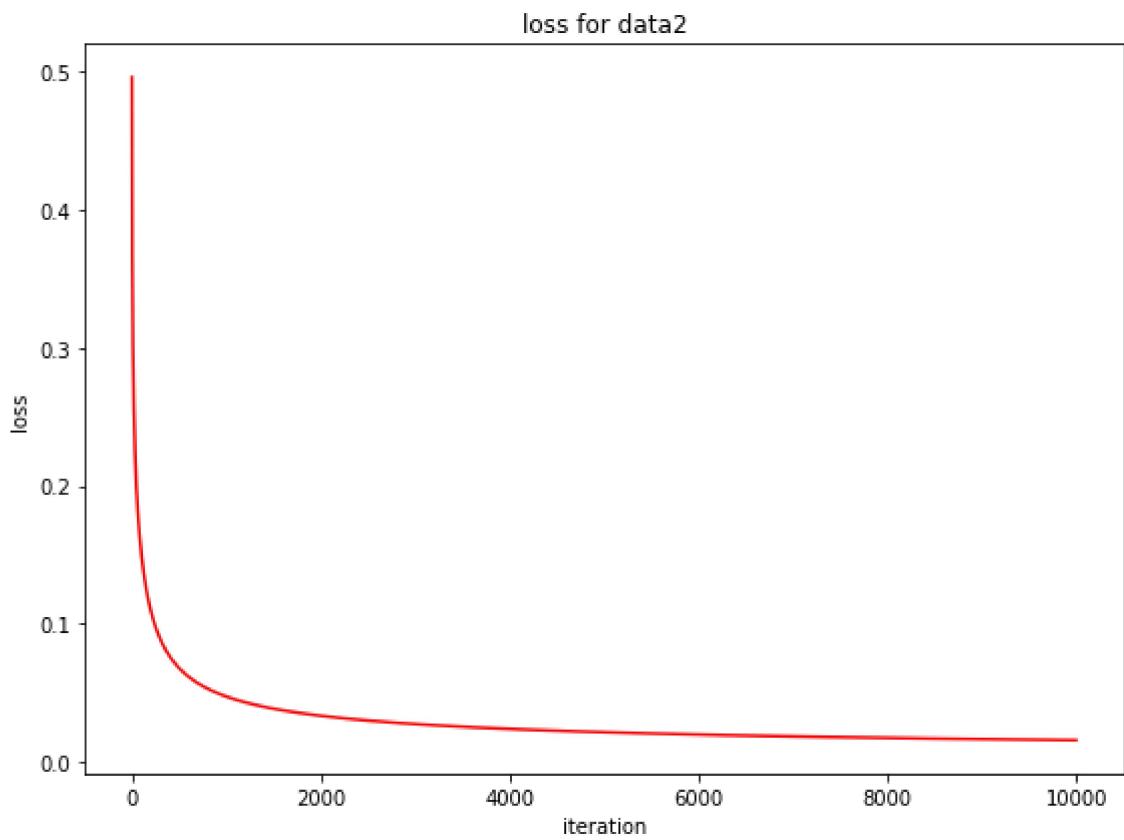
for i in range(number_result):
    title = '## [RESULT {:02d}]'.format(i+1)
    name_function = 'function_result_{:02d}()'.format(i+1)
```

```
print('*****')
print(title)
print('*****')
eval(name_function)
```

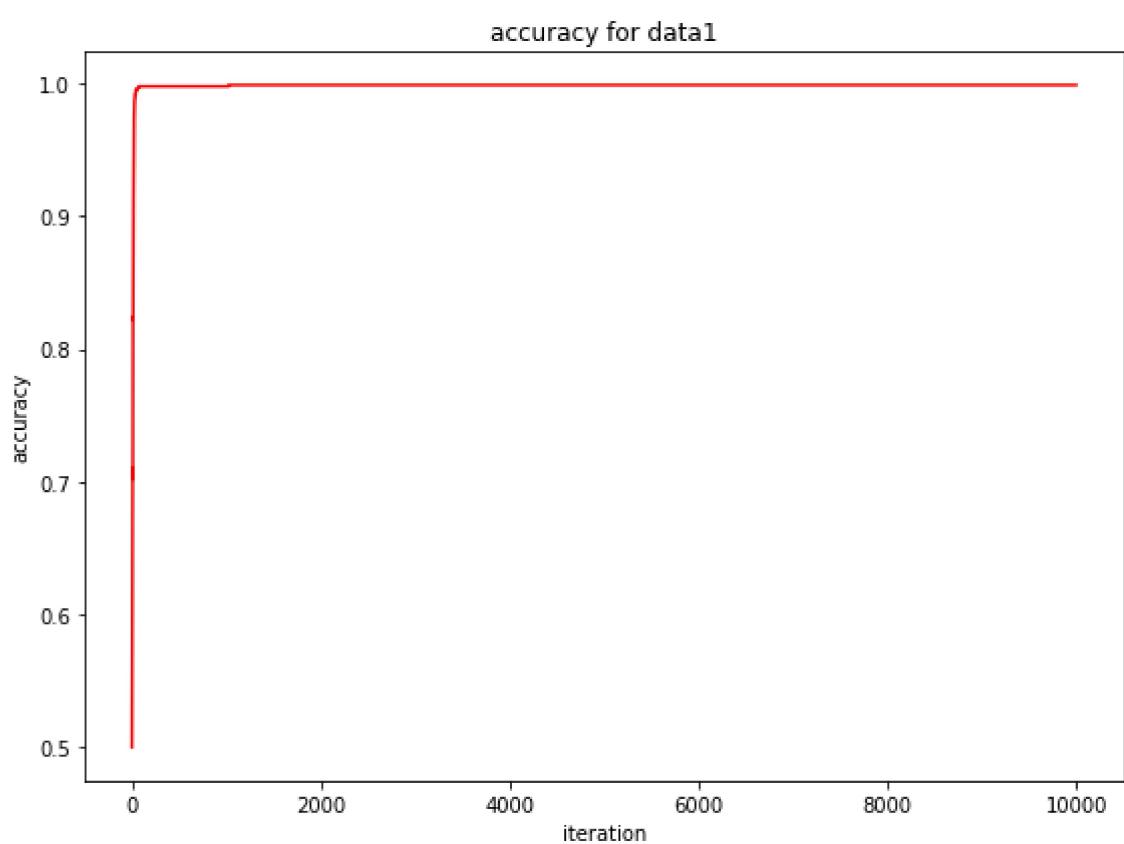
```
*****
## [RESULT 01]
*****
final loss for data1 = 0.0070289462
*****
## [RESULT 02]
*****
final loss for data2 = 0.0157305907
*****
## [RESULT 03]
*****
final accuracy for data1 = 0.9990000000
*****
## [RESULT 04]
*****
final accuracy for data2 = 0.9940000000
*****
## [RESULT 05]
*****
```



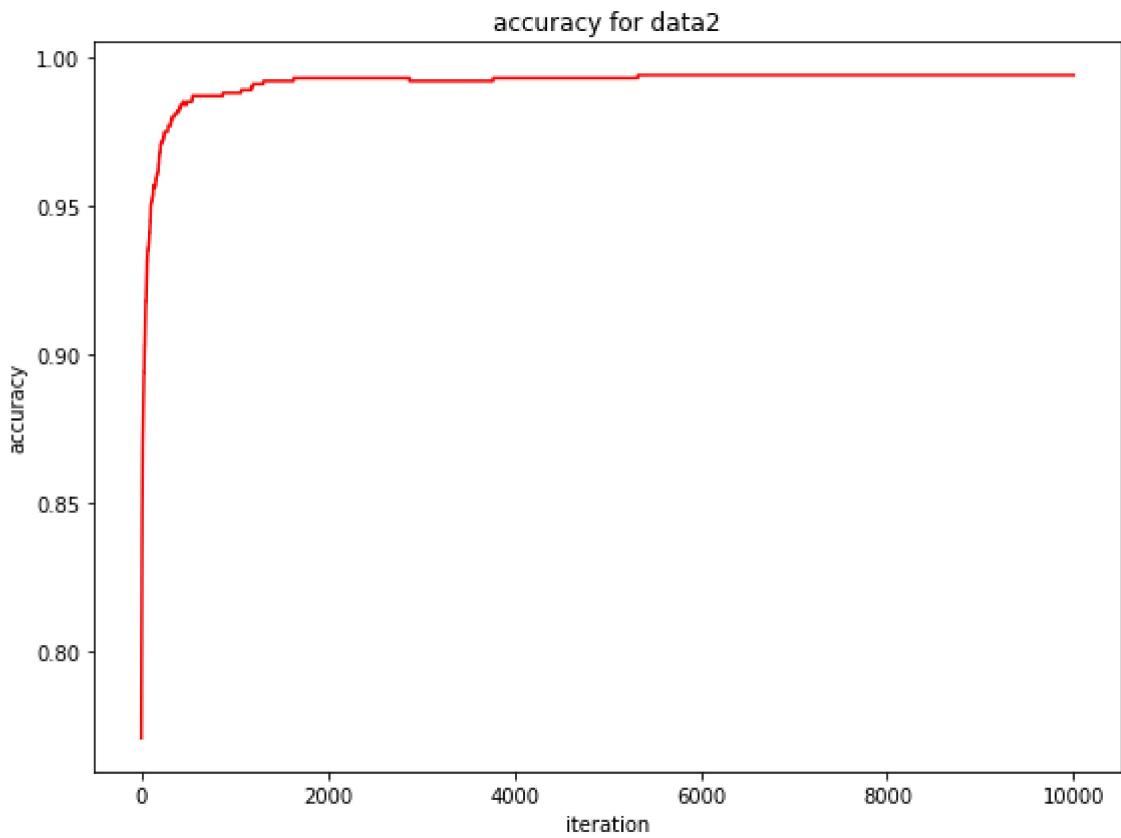
```
*****
## [RESULT 06]
*****
```



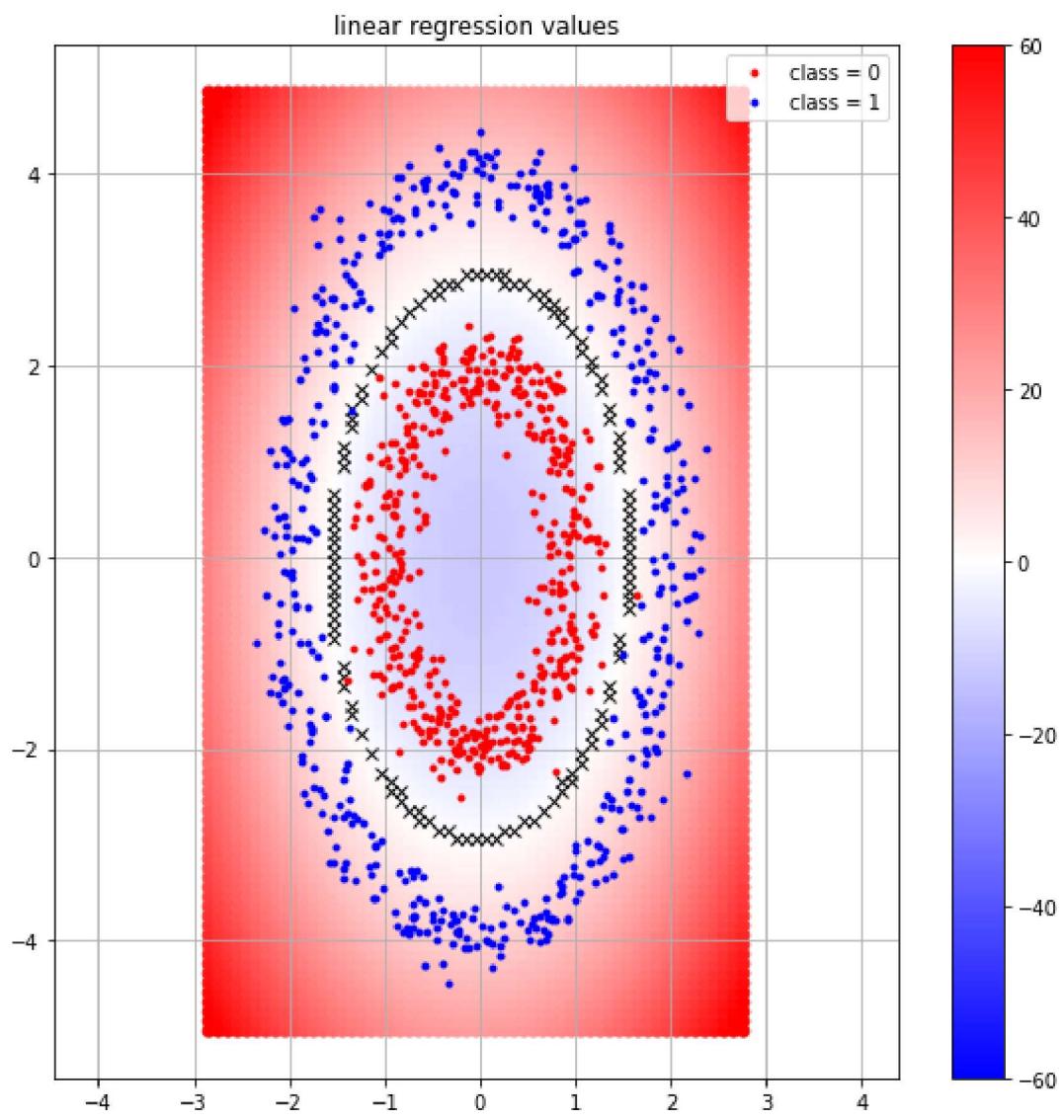
```
*****  
## [RESULT 07]  
*****
```



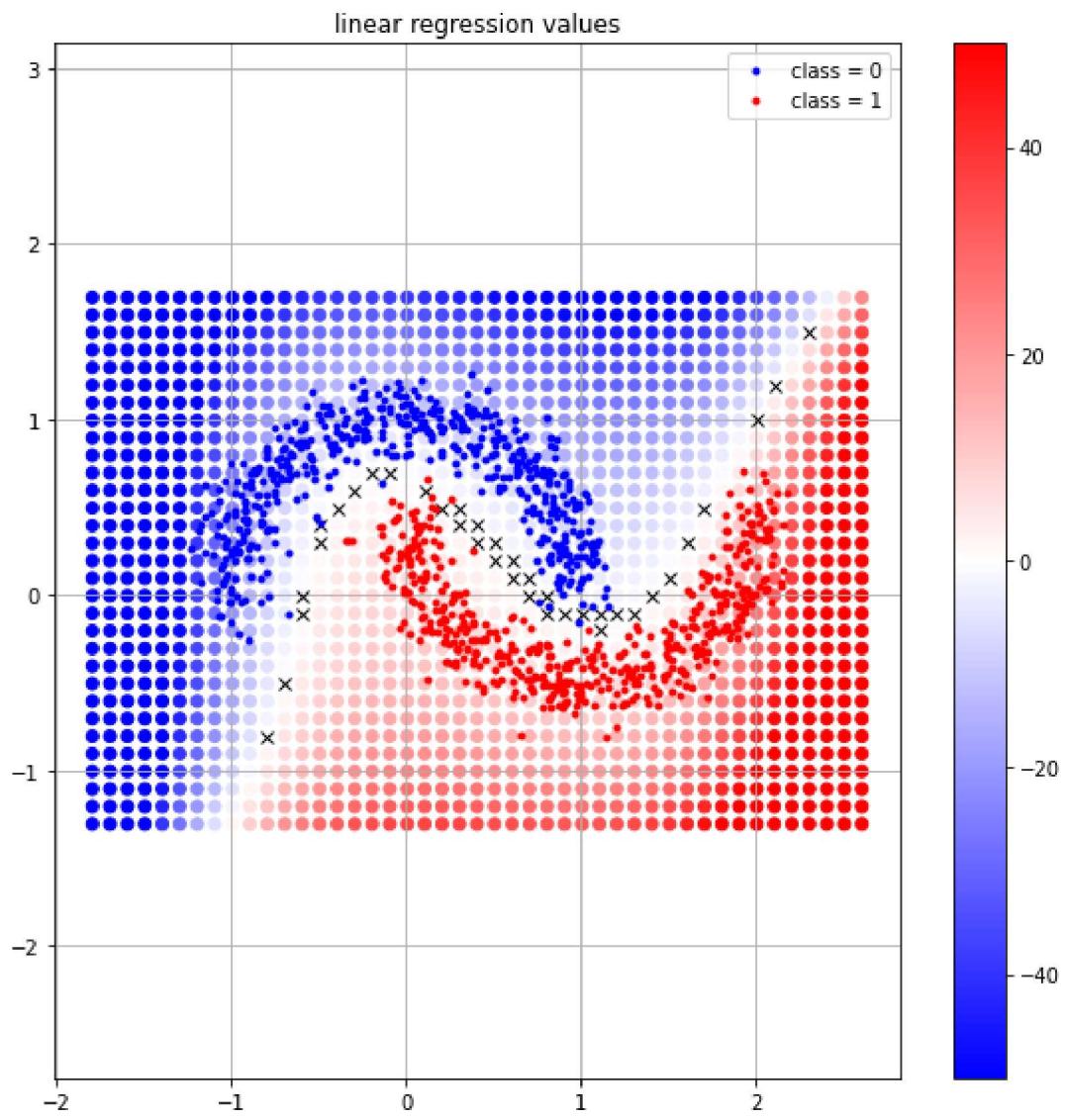
```
*****  
## [RESULT 08]  
*****
```



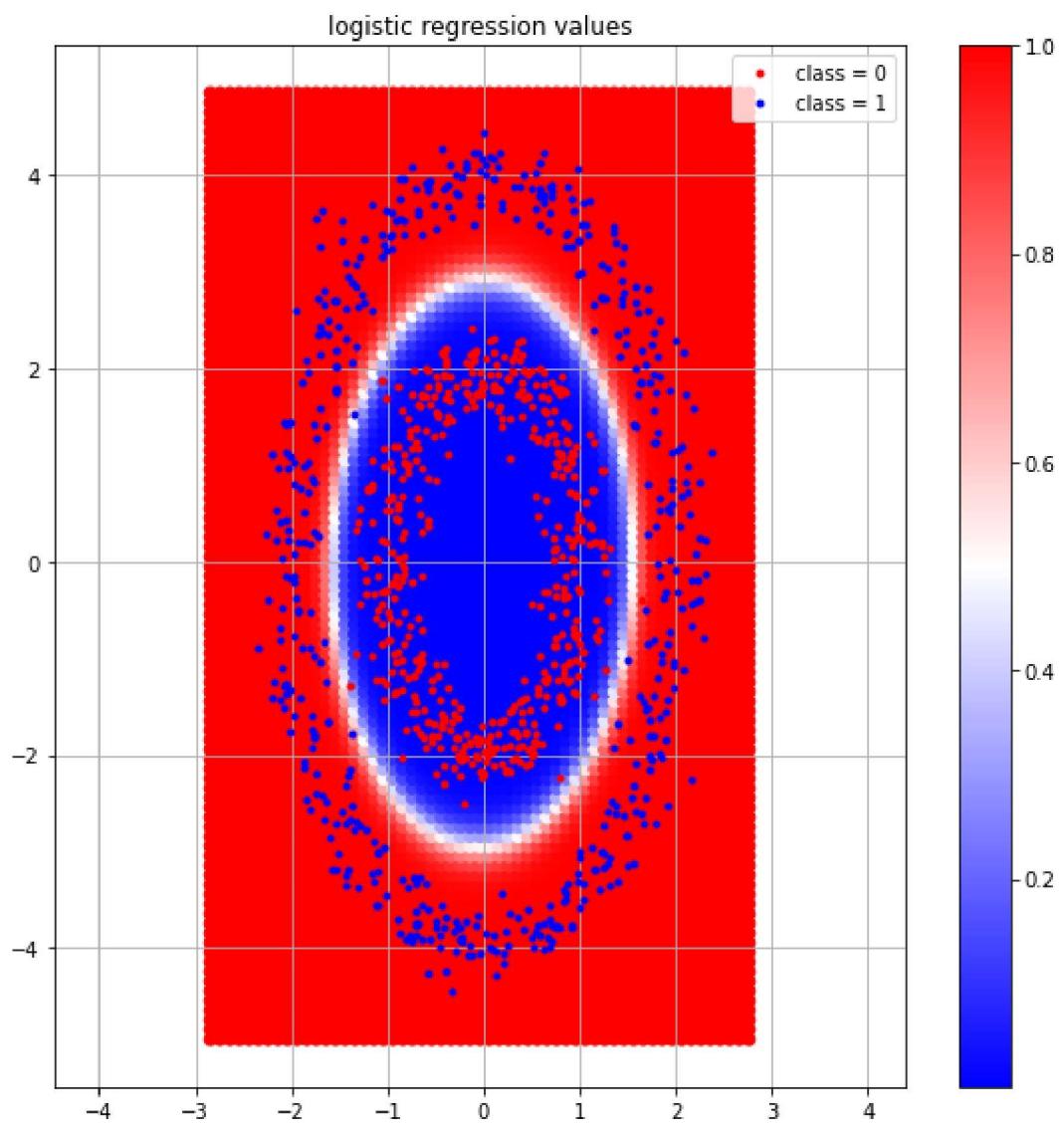
```
*****  
## [RESULT 09]  
*****
```



```
*****  
## [RESULT 10]  
*****
```



```
*****  
## [RESULT 11]  
*****
```



```
*****  
## [RESULT 12]  
*****
```

