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 [ ]:

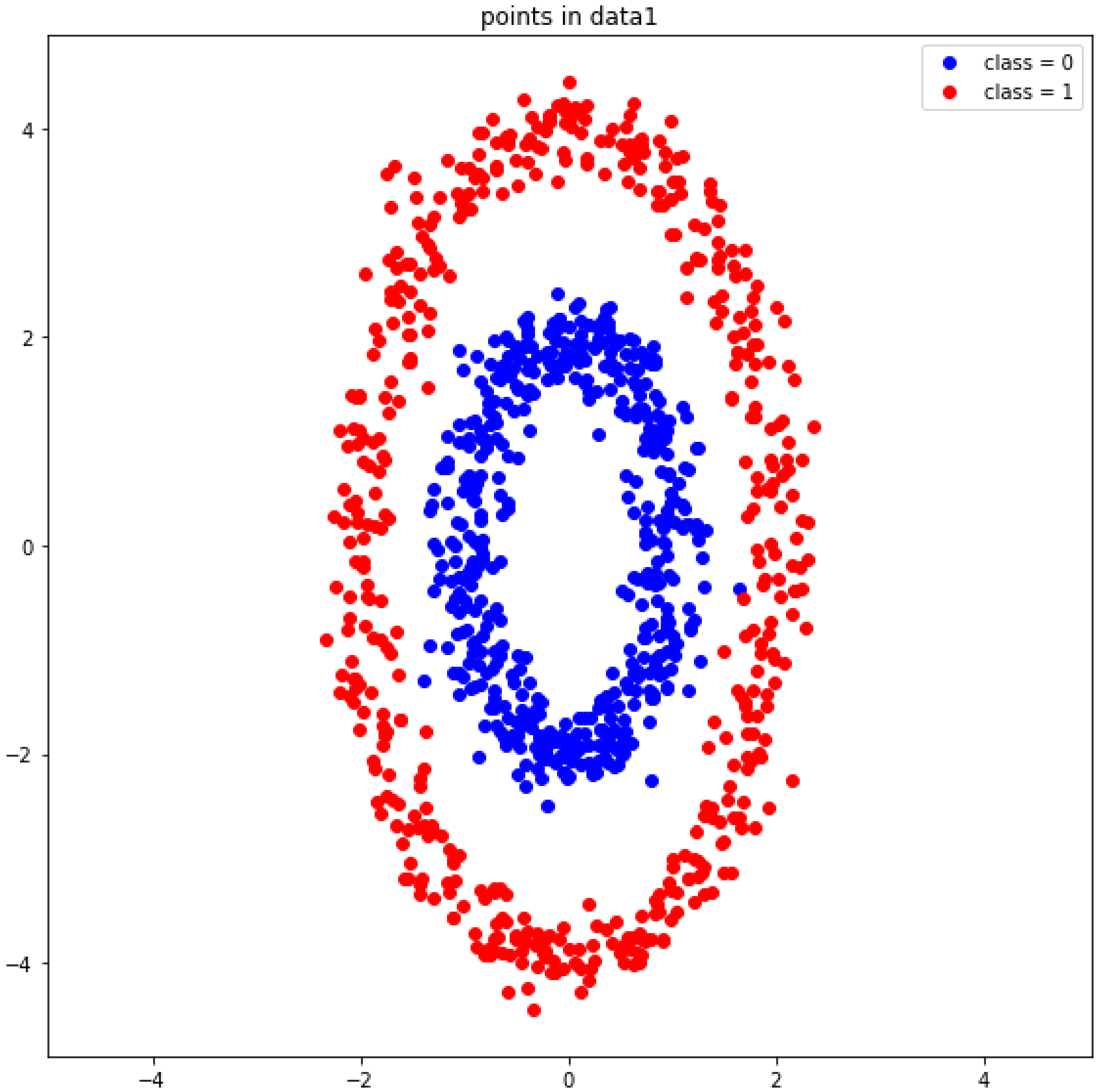
```
fname_data1 = 'assignment_09_data1.txt'
fname_data2 = 'assignment_09_data2.txt'

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

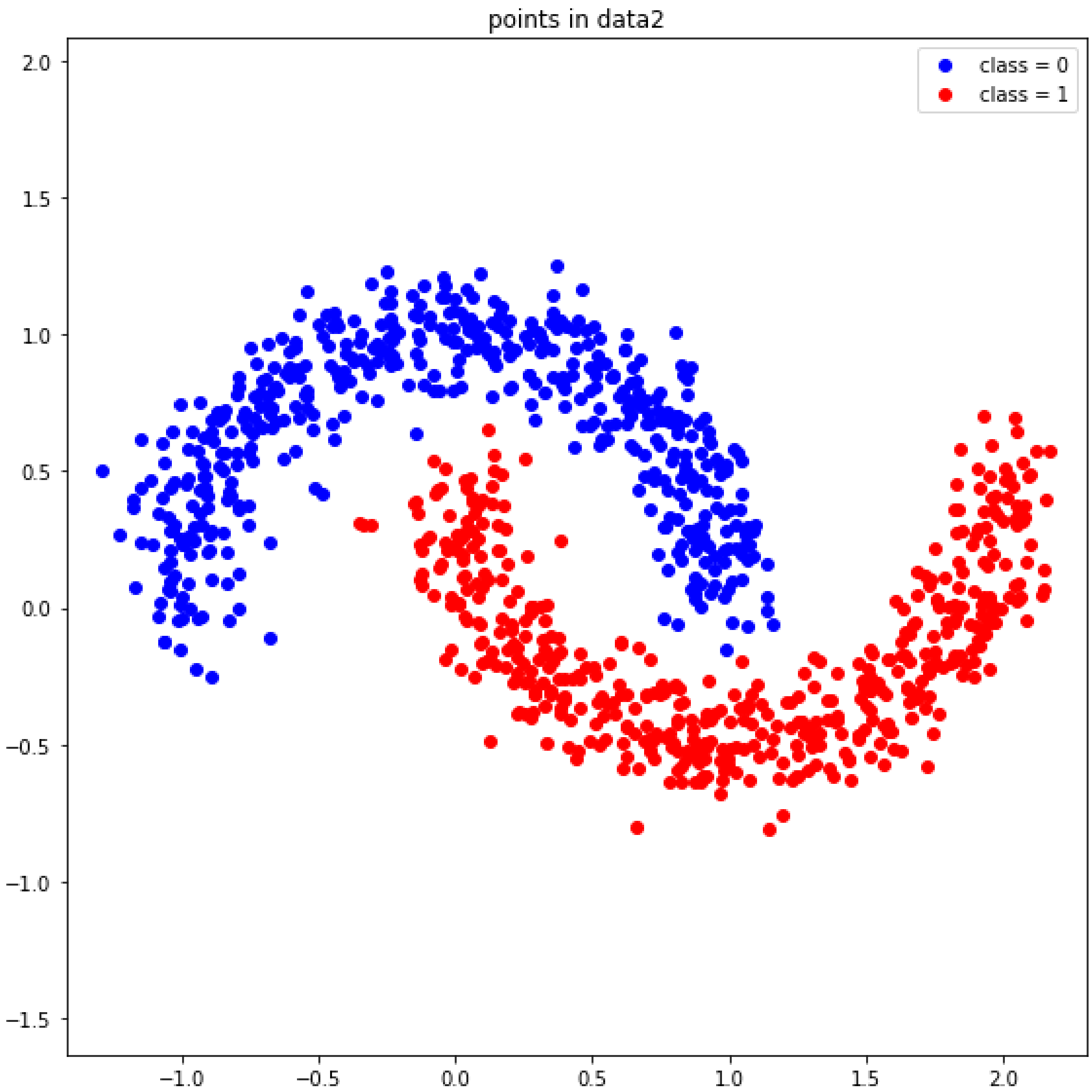
In [ ]:

```
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 $(1, f_1(x, y), f_2(x, y), \cdots, f_{k-1}(x, y)) \in \mathbb{R}^k$

In [ ]:

```
def compute_feature1(point):

    # +++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    n = np.shape(data1_point)[0]
    x = data1_point[:,0]
    y = data1_point[:,1]

    x_square = np.power(x,2)
    y_square = np.power(y,2)
    xy = x*y

    feature = np.column_stack([np.ones(n), x, y, x_square, y_square, xy])

    #
    # +++++++++++++++++++++++++++++++++++++++++++++++++++

    return feature
```

In [ ]:

```
def compute_feature2(point):

    # +++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    n = np.shape(data2_point)[0]
    x = data2_point[:,0]
    y = data2_point[:,1]

    x_square = np.power(x,2)
    x_cubic = np.power(x,3)


    feature = np.column_stack([np.ones(n), x, y, x_square, x_cubic])

    #
    # +++++++++++++++++++++++++++++++++++++++++++++++++++

    return feature
```

define the linear regression function

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

In [ ]:

```
def compute_linear_regression(theta, feature):

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

    value = np.matmul(theta, feature.T)
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    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

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

In [ ]:

```
def compute_logistic_regression(theta, feature):

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

    value = sigmoid(compute_linear_regression(theta, feature))
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return value
```

define the residual function

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

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

In [ ]:

```
def compute_loss(theta, feature, label):

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

    residual = compute_residual(theta, feature, label)
    loss = np.sum(residual) / np.shape(residual)[0]
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return loss
```

define the gradient of the loss with respect to the model parameter θ

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

In [ ]:

```
def compute_gradient(theta, feature, label):

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

    h = compute_logistic_regression(theta, feature)
    gradient = np.matmul(feature.T, (h-label)) / np.shape(label)[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 = np.shape(feature)[0]
    pred = np.zeros(n)
    h = compute_logistic_regression(theta, feature)
    sum = 0
    for i in range(n):
        if h[i] >= 0.5:
            pred[i] = 1
        if pred[i] == label[i]:
                sum += 1

    accuracy = sum / n


    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return accuracy
```

initialize the gradient descent algorithm

In [ ]:

```
data1_number_iteration   = 30000
data2_number_iteration   = 30000

data1_learning_rate   = 0.5
data2_learning_rate   = 0.3

data1_number_feature    = 6
data2_number_feature    = 5

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_feature1(data1
_point), data1_label)
    loss1       = compute_loss(theta1, compute_feature1(data1_point), data1_label)
    accuracy1   = compute_accuracy(theta1, compute_feature1(data1_point), data1_label)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    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_feature2(data2_point), data2_label)
    loss2       = compute_loss(theta2, compute_feature2(data2_point), data2_label)
    accuracy2   = compute_accuracy(theta2, compute_feature2(data2_point), data2_label)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    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 values in
the construction of coordinate
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1) # USE THIS VALUE for the range of y values in
the construction of coordinate

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

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

    ZZ = (data1_theta_optimal[0] + data1_theta_optimal[1]*XX + data1_theta_optimal[2]*YY + dat
a1_theta_optimal[3]*np.power(XX,2) +
    data1_theta_optimal[4]*np.power(YY,2) + data1_theta_optimal[5]*(XX*YY))
    #y_hat = -(theta_optimal[0]/theta_optimal[2] + theta_optimal[1]/theta_optimal[2]*X)

    plt.plot(data1_point_x_class_0, data1_point_y_class_0, 'o', markersize=3, color='blue', la
bel='class = 0')
    plt.plot(data1_point_x_class_1, data1_point_y_class_1, 'o', markersize=3, color='red', lab
el='class = 1')
    #plt.plot(X, y_hat,linestyle='-',color='black')



    im = plt.imshow(ZZ, aspect='auto', origin = 'lower', cmap = 'RdBu_r', alpha=0.98, extent=(
min_x-0.5,max_x+0.5,min_y-0.5,max_y+0.5), vmin=-20, vmax=20)
    plt.colorbar(im)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    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 values in the construction of coordinate
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1) # USE THIS VALUE for the range of y values in the construction of coordinate

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

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

    ZZ = (data2_theta_optimal[0] + data2_theta_optimal[1]*XX + data2_theta_optimal[2]*YY + data2_theta_optimal[3]*np.power(XX,2) +
    data2_theta_optimal[4]*np.power(XX,3))
    #y_hat = -(theta_optimal[0]/theta_optimal[2] + theta_optimal[1]/theta_optimal[2]*X)


    plt.plot(data2_point_x_class_0, data2_point_y_class_0, 'o', markersize=3, color='blue', label='class = 0')
    plt.plot(data2_point_x_class_1, data2_point_y_class_1, 'o', markersize=3, color='red', label='class = 1')
    #plt.plot(X, y_hat,linestyle='-',color='black')



    im = plt.imshow(ZZ, aspect='auto', origin = 'lower', cmap = 'RdBu_r', alpha=0.98, extent=(min_x-0.5,max_x+0.5,min_y-0.5,max_y+0.5), vmin=-20, vmax=20)
    plt.colorbar(im)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    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 values in
the construction of coordinate
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1) # USE THIS VALUE for the range of y values in
the construction of coordinate

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

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

    logit_ZZ = sigmoid(data1_theta_optimal[0] + data1_theta_optimal[1]*XX + data1_theta_optima
l[2]*YY + data1_theta_optimal[3]*np.power(XX,2) +
    data1_theta_optimal[4]*np.power(YY,2) + data1_theta_optimal[5]*(XX*YY))

    plt.plot(data1_point_x_class_0, data1_point_y_class_0, 'o', markersize=3, color='blue', la
bel='class = 0')
    plt.plot(data1_point_x_class_1, data1_point_y_class_1, 'o', markersize=3, color='red', lab
el='class = 1')


    im = plt.imshow(logit_ZZ, aspect='auto', origin = 'lower', cmap = 'RdBu_r', alpha=0.98, ex
tent=(min_x-0.5,max_x+0.5,min_y-0.5,max_y+0.5))
    plt.colorbar(im)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

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

In [ ]:

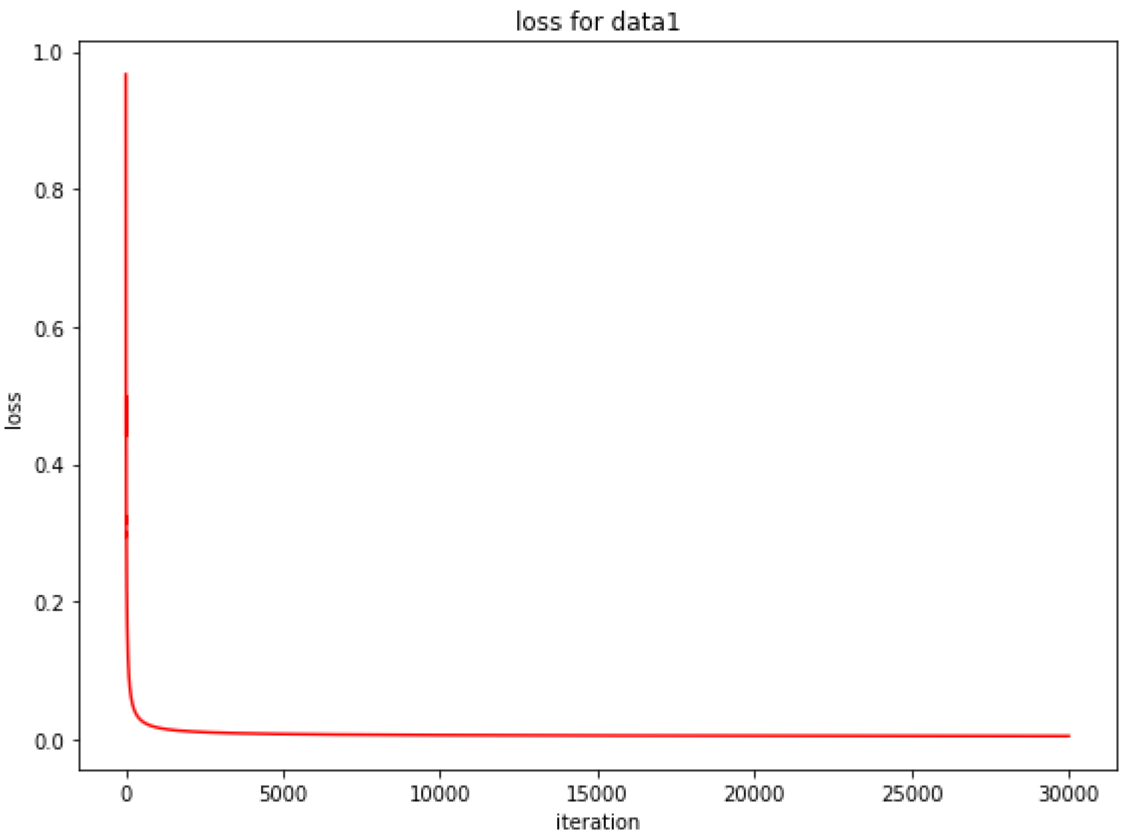
```
def function_result_12():

    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 values in the construction of coordinate
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1) # USE THIS VALUE for the range of y values in the construction of coordinate

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

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

    logit_ZZ = sigmoid(data2_theta_optimal[0] + data2_theta_optimal[1]*XX + data2_theta_optimal[2]*YY + data2_theta_optimal[3]*np.power(XX,2) +
    data2_theta_optimal[4]*np.power(XX,3))


    plt.plot(data2_point_x_class_0, data2_point_y_class_0, 'o', markersize=3, color='blue', label='class = 0')
    plt.plot(data2_point_x_class_1, data2_point_y_class_1, 'o', markersize=3, color='red', label='class = 1')

    im = plt.imshow(logit_ZZ, aspect='auto', origin = 'lower', cmap = 'RdBu_r', alpha=0.98, extent=(min_x-0.5,max_x+0.5,min_y-0.5,max_y+0.5))
    plt.colorbar(im)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

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

results

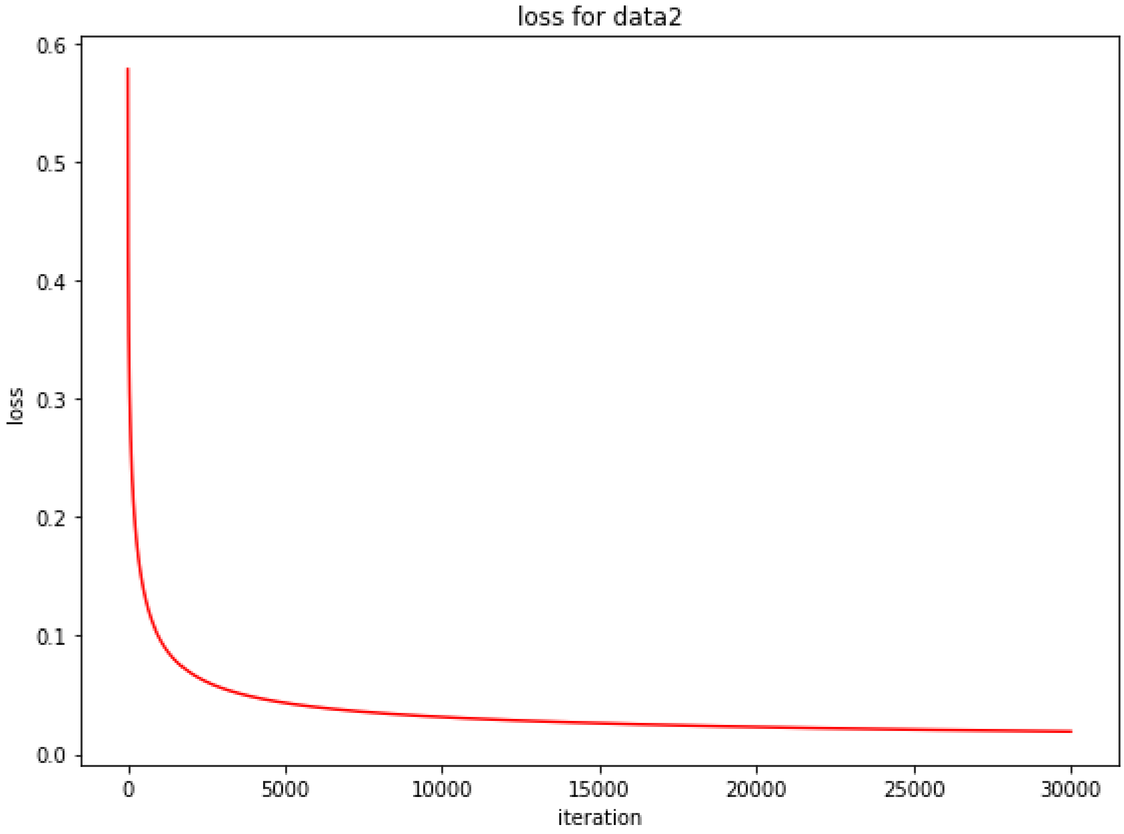
In [ ]:

```
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)
```

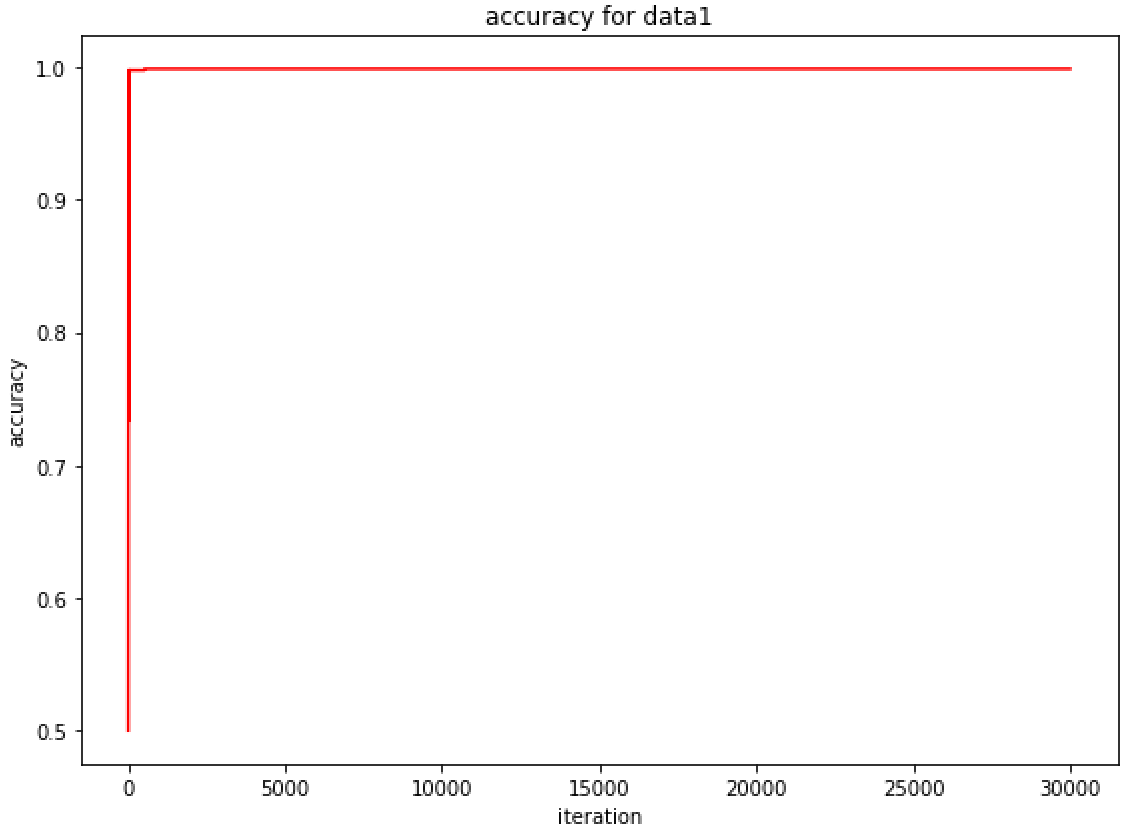
```
**************************************************
## [RESULT 01]
**************************************************
final loss for data1 =  0.0044944631
**************************************************
## [RESULT 02]
**************************************************
final loss for data2 =  0.0190214069
**************************************************
## [RESULT 03]
**************************************************
final accuracy for data1 =  0.9990000000
**************************************************
## [RESULT 04]
**************************************************
final accuracy for data2 =  0.9940000000
**************************************************
## [RESULT 05]
**************************************************
```



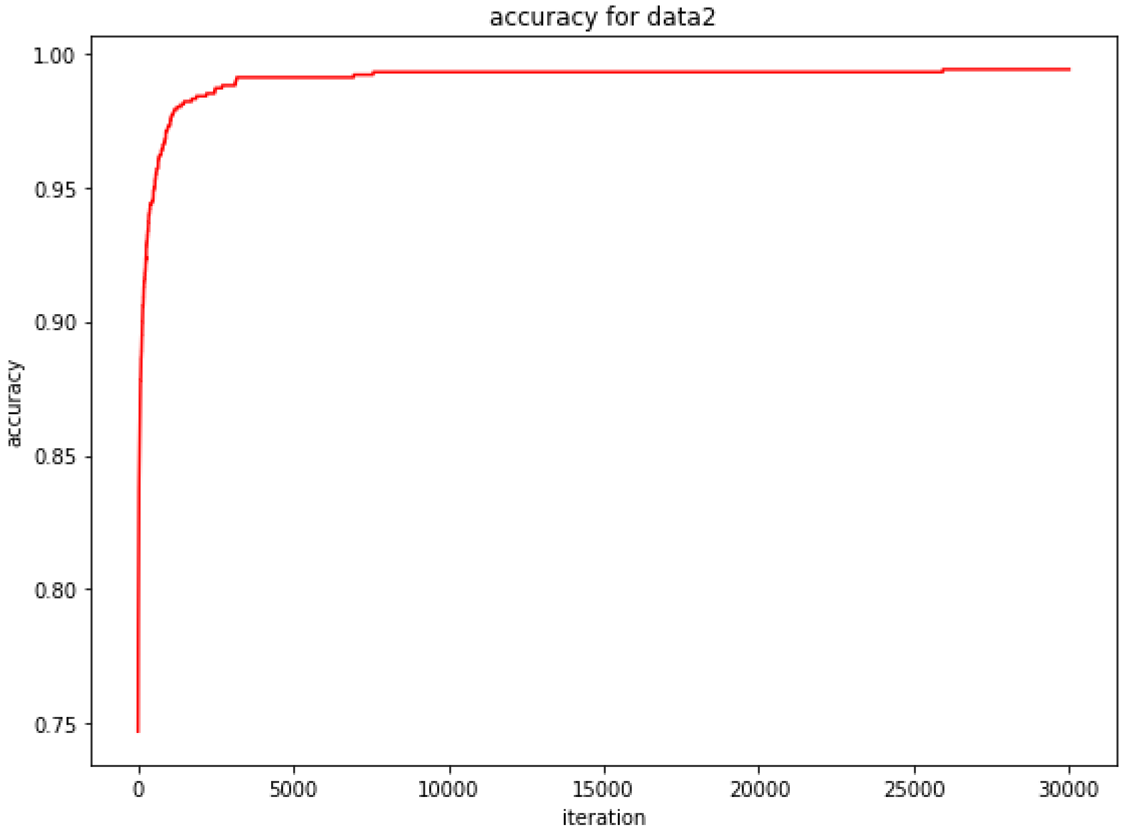

```
**************************************************
## [RESULT 06]
**************************************************
```



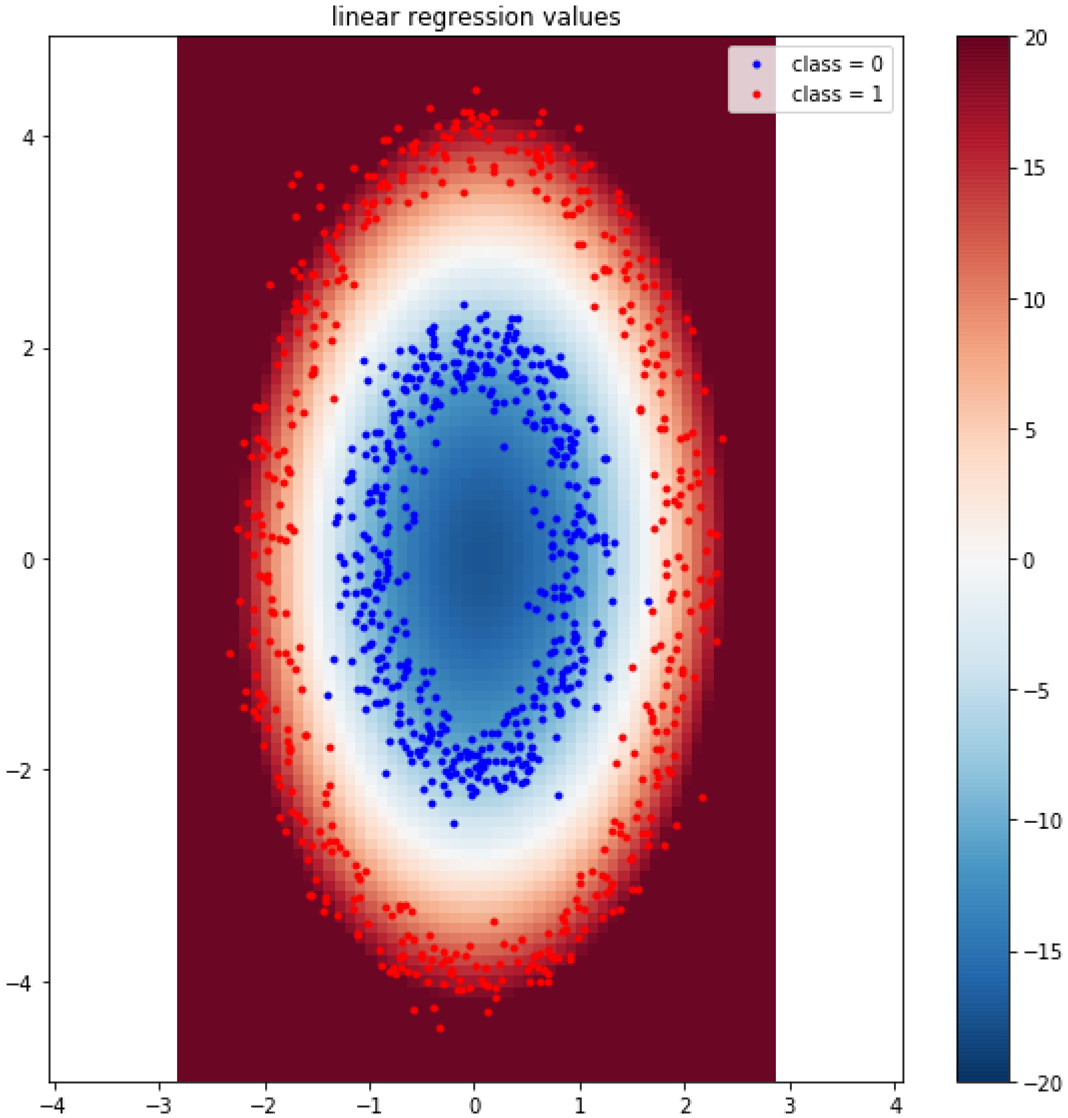

```
**************************************************
## [RESULT 07]
**************************************************
```



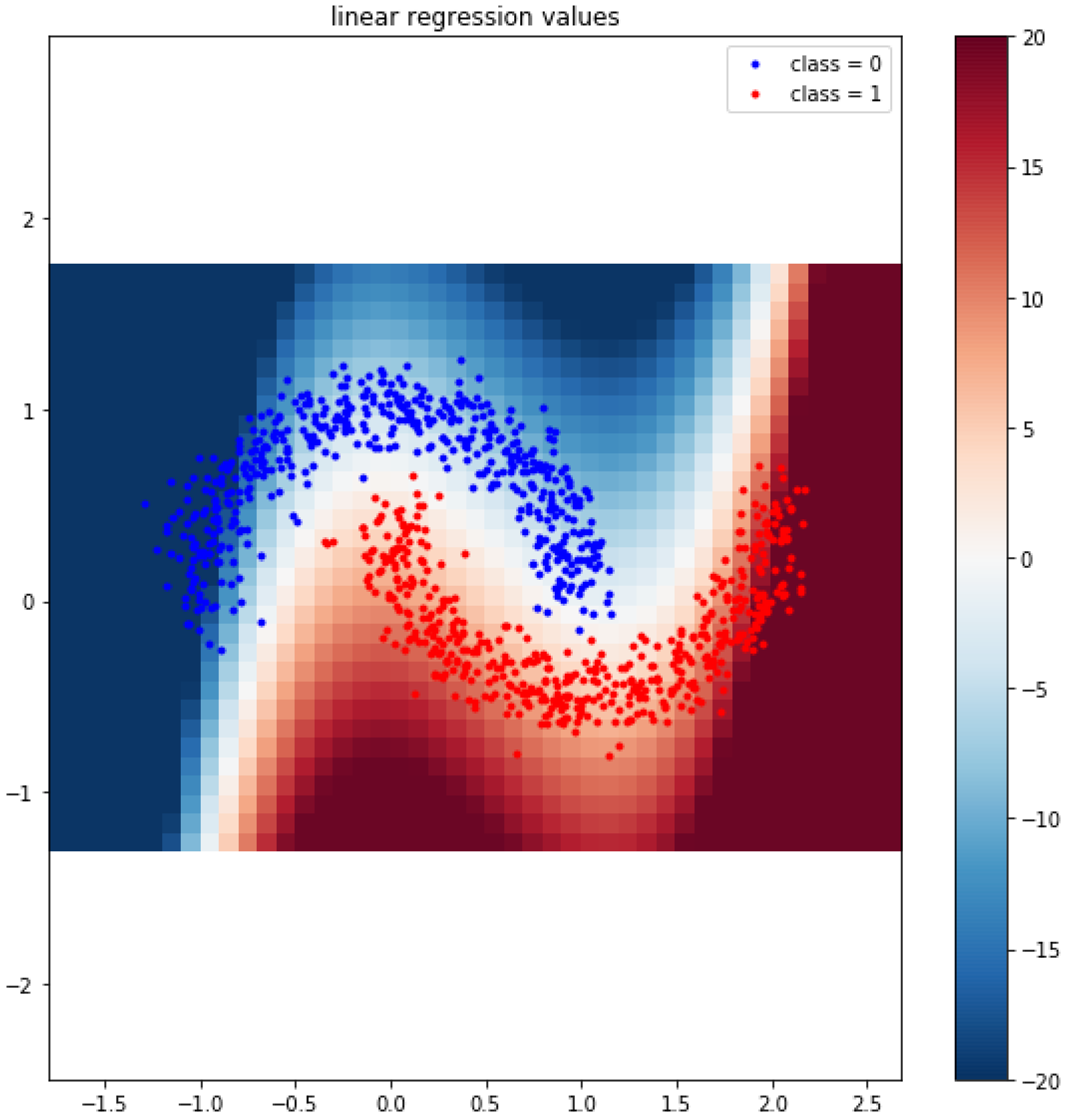

```
**************************************************
## [RESULT 08]
**************************************************
```


accuracy for data2
accuracy
iteration
1.00
0.95
0.90
0.85
0.80
0.75
0
5000
10000
15000
20000
25000
30000

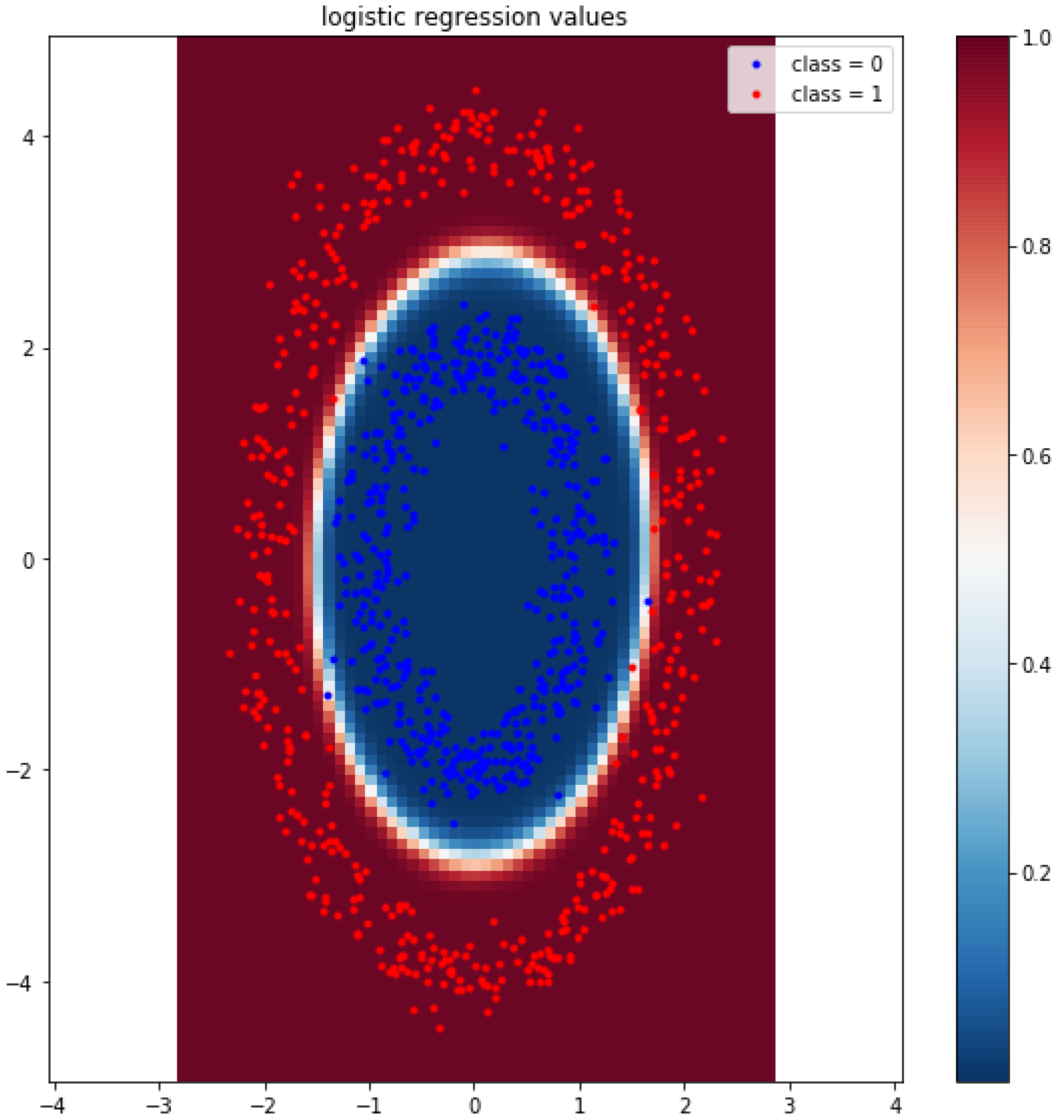
```
**************************************************
## [RESULT 09]
**************************************************
```


linear regression values
class = 0
class = 1
4
2
0
−2
−4
−4
−3
−2
−1
0
1
2
3
4
20
15
10
5
0
−5
−10
−15
−20

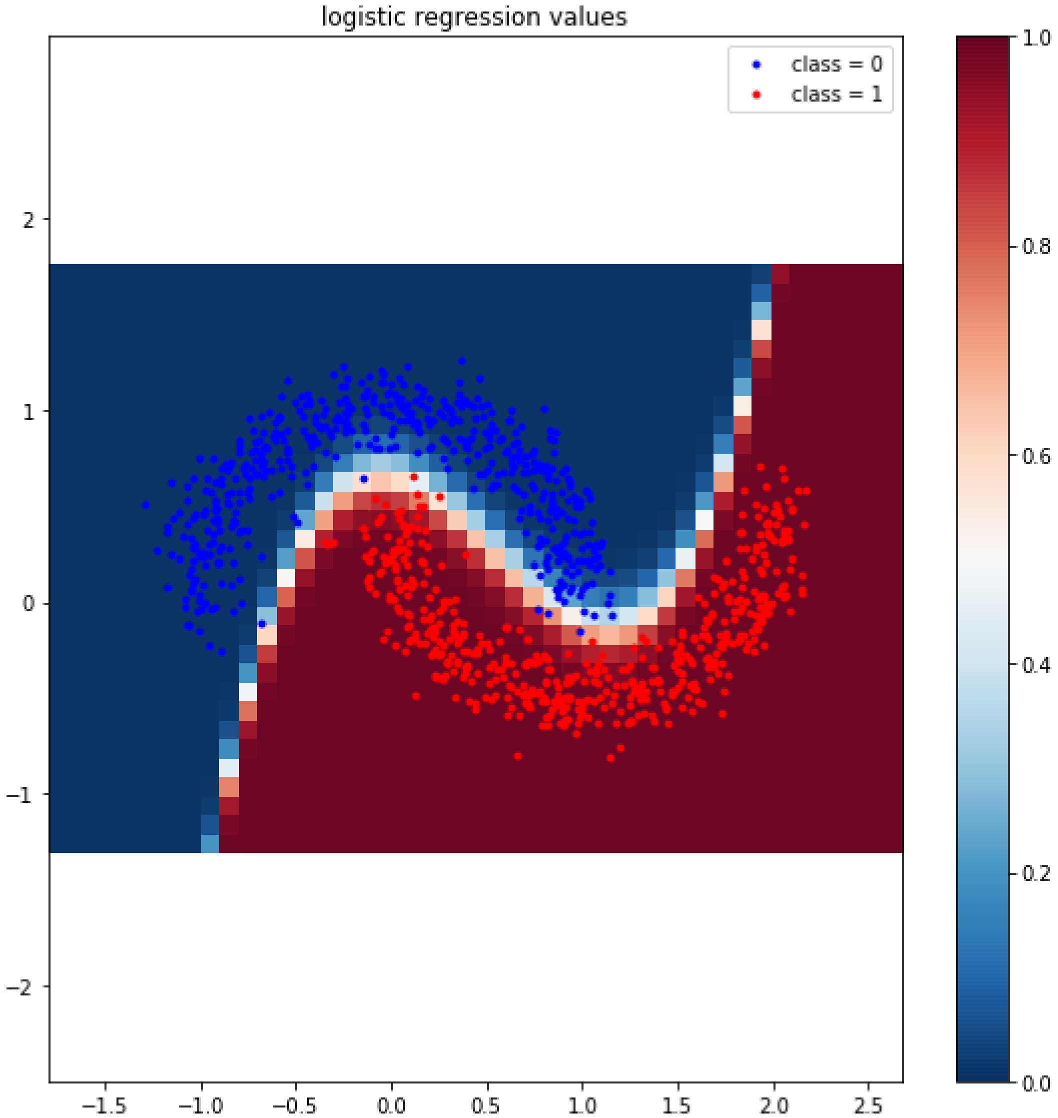
```
*************************************************
## [RESULT 10]
*************************************************
```


linear regression values
class = 0
class = 1
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
20
15
10
5
0
−5
−10
−15
−20

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


logistic regression values
class = 0
class = 1
4
2
0
−2
−4
−4
−3
−2
−1
0
1
2
3
4
1.0
0.8
0.6
0.4
0.2

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


logistic regression values
class = 0
class = 1
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
1.0
0.8
0.6
0.4
0.2
0.0

In [ ]:

```

```