Logistic Regression

import library

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

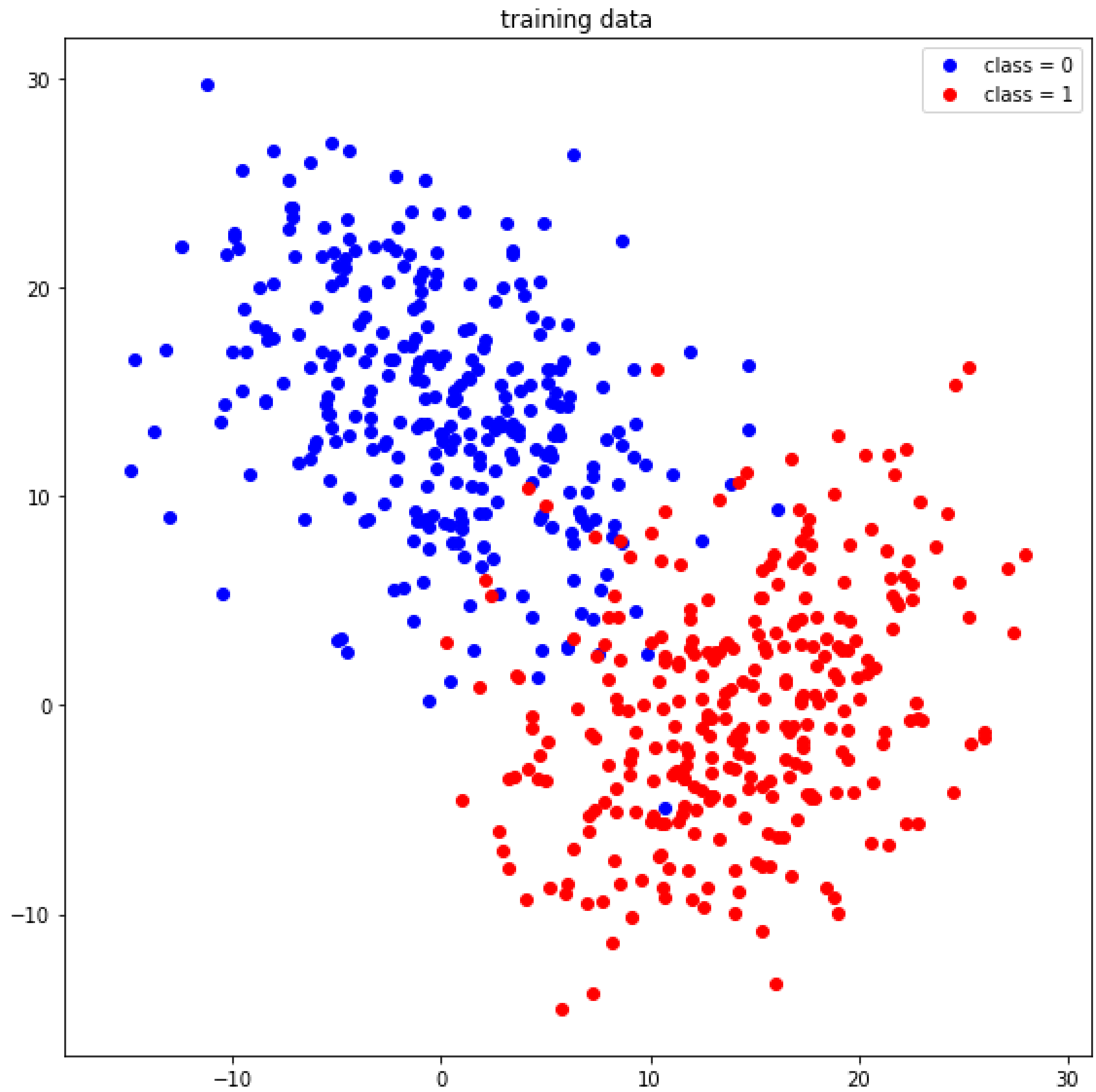
load training data

In [ ]:
```
fname_data = 'assignment_08_data.csv'

data        = np.genfromtxt(fname_data, delimiter=',')
number_data = data.shape[0]

point_x = data[:, 0]
point_y = data[:, 1]
label   = data[:, 2]

print('number of data = ', number_data)
print('data type of point x = ', point_x.dtype)
print('data type of point y = ', point_y.dtype)

point_x_class_0 = point_x[label == 0]
point_y_class_0 = point_y[label == 0]

point_x_class_1 = point_x[label == 1]
point_y_class_1 = point_y[label == 1]
```

```
number of data =  600
data type of point x =  float64
data type of point y =  float64
```

plot the data

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

plt.title('training data')
plt.plot(point_x_class_0, point_y_class_0, 'o', color='blue', label='class = 0')
plt.plot(point_x_class_1, point_y_class_1, 'o', color='red', label='class = 1')
plt.axis('equal')
plt.legend()
plt.tight_layout()
plt.show()
```



define the linear regression function

- $\theta = (\theta_0, \theta_1, \theta_2)$
- point = $(1, x, y) \in \mathbb{R}^3$

In [ ]:
```
def compute_linear_regression(theta, point):

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

    value = theta[0]*point[:,0] + theta[1]*point[:,1] + theta[2]*point[:,2]
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return value
```

define sigmoid function with input

- $z \in \mathbb{R}$

In [ ]:
```
def sigmoid(z):
```

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

    value = 1/(1+np.exp((-1)*z))
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return value
```

define the logistic regression function

- $\theta = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$
- point = $(1, x, y) \in \mathbb{R}^3$

In [ ]:

```
def compute_logistic_regression(theta, point):

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

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

    return value
```

define the residual function

- $\theta = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$
- point $= (x, y) \in \mathbb{R}^2$
- label $= l \in \{0, 1\}$

In [ ]:

```
def compute_residual(theta, point, label):

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

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

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

    return residual
```

define the loss function for the logistic regression

- $\theta = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$
- point $= (1, x, y) \in \mathbb{R}^3$
- label $= l \in \{0, 1\}$

In [ ]:
```
def compute_loss(theta, point, label):

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

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

    return loss
```

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

- $\theta = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$
- point $= (1, x, y) \in \mathbb{R}^3$
- label $= l \in \{0, 1\}$

In [ ]:
```
def compute_gradient(theta, point, label):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    x = np.ones(point.shape[0])
    x = np.append(x, point[:, 1], axis=0)
    x = np.append(x, point[:, 2], axis=0)
    x = np.reshape(x, (3, point.shape[0]))
    x = x.T

    h = (compute_logistic_regression(theta, point) - label)

    gradient = (h.dot(x))/point.shape[0]
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return gradient
```

initialize the gradient descent algorithm

In [ ]:
```
num_iteration    = 5000  # USE THIS VALUE for the number of gradient descent iterations
learning_rate    = 0.001 # USE THIS VALUE for the learning rate

theta            = np.array((0, 0, 0))
theta_iteration = np.zeros((num_iteration, theta.size))
loss_iteration   = np.zeros(num_iteration)

number_point_class_0     = len(point_x_class_0)
number_point_class_1     = len(point_x_class_1)

point_class_0    = np.ones((number_point_class_0, 3))
point_class_1    = np.ones((number_point_class_1, 3))

point_class_0[:, 1] = point_x_class_0
point_class_0[:, 2] = point_y_class_0

point_class_1[:, 1] = point_x_class_1
```

```
point_class_1[:, 2] = point_y_class_1

label_0     = np.zeros(number_point_class_0)
label_1     = np.ones(number_point_class_1)

point = np.concatenate((point_class_0, point_class_1), axis=0)
label = np.concatenate((label_0, label_1), axis=0)

print('shape of point_class_0 : ', point_class_0.shape)
print('shape of point_class_1 : ', point_class_1.shape)
print('shape of label_0 : ', label_0.shape)
print('shape of label_1 : ', label_1.shape)

print('shape of point : ', point.shape)
print('shape of label : ', label.shape)
```

```
shape of point_class_0 :  (300, 3)
shape of point_class_1 :  (300, 3)
shape of label_0 :  (300,)
shape of label_1 :  (300,)
shape of point :  (600, 3)
shape of label :  (600,)
```

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

In [ ]:

```
for i in range(num_iteration):

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

    theta   = theta - learning_rate*compute_gradient(theta, point, label)
    loss    = compute_loss(theta, point, label)
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    theta_iteration[i, :]   = theta
    loss_iteration[i]       = loss

theta_optimal = theta
```

---

functions for presenting the results

In [ ]:

```
def function_result_01():

    input1  = np.array([0.1, 0.2, 0.3])
    input2  = np.array([[1, 2, 3], [1, -2, -3]])

    value = compute_linear_regression(input1, input2)

    print(value)
```

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

    input1  = np.array([0.1, 0.2, 0.3])
    input2  = np.array([[1, 2, 3], [1, -2, -3]])

    value = compute_logistic_regression(input1, input2)

    print(value)
```

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

    input1  = np.array([0.1, 0.2, 0.3])
    input2  = np.array([[1, 2, 3], [1, -2, -3]])
    input3  = np.array([0, 1])

    value = compute_residual(input1, input2, input3)

    print(value)
```

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

    input1  = np.array([0.1, 0.2, 0.3])
    input2  = np.array([[1, 2, 3], [1, -2, -3]])
    input3  = np.array([[0], [1]])

    value = compute_loss(input1, input2, input3)

    print(value)
```

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

    input1  = np.array([0.1, 0.2, 0.3])
    input2  = np.array([[1, 2, 3], [1, -2, -3]])
    input3  = np.array([[0], [1]])

    value = compute_gradient(input1, input2, input3)

    print(value)
```

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

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

    plt.plot(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))   # USE THIS VALUE for the size of the figure
    plt.title('model parameter')

    plt.plot(theta_iteration[:, 0], '-', color='red', label=r'$₩theta_0$')
```

```
    plt.plot(theta_iteration[:, 1], '-', color='green', label=r'$\theta_1$')
    plt.plot(theta_iteration[:, 2], '-', color='blue', label=r'$\theta_2$')

    plt.xlabel('iteration')
    plt.legend()

    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_08():

    X = np.arange(-20, 35, 0.1) # USE THIS VALUE for the range of x values in the con
    Y = np.arange(-20, 35, 0.1) # USE THIS VALUE for the range of y values in the con

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

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

    xravel = XX.ravel(); yravel = YY.ravel()
    F = compute_linear_regression(theta_optimal, (np.array([np.ones(xravel.size), xra

    FF = F.reshape(XX.shape)

    plt.figure(figsize=(8,8))
    plt.title('linear regression values')

    cntrf = plt.contourf(XX, YY, FF, cmap=plt.get_cmap('RdBu_r'), levels = 90)
    plt.colorbar(cntrf)


    plt.plot(point_x_class_0, point_y_class_0, '.', color='blue', label='class 0')
    plt.plot(point_x_class_1, point_y_class_1, '.', color='red', label='class 1')
    plt.contour(XX, YY, FF, levels=[0], colors = "black")

    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_09():

    X = np.arange(-20, 35, 0.1) # USE THIS VALUE for the range of x values in the con
    Y = np.arange(-20, 35, 0.1) # USE THIS VALUE for the range of y values in the con

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

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

    xravel = XX.ravel(); yravel = YY.ravel()
    F = compute_logistic_regression(theta_optimal, (np.array([np.ones(xravel.size), x

    FF = F.reshape(XX.shape)

    plt.figure(figsize=(8,8))
    plt.title('logistic regression values')

    cntrf = plt.contourf(XX, YY, FF, cmap=plt.get_cmap('RdBu_r'), levels = 100)
    plt.colorbar(cntrf)


    plt.plot(point_x_class_0, point_y_class_0, '.', color='blue', label='class 0')
    plt.plot(point_x_class_1, point_y_class_1, '.', color='red', label='class 1')
    # plt.contour(XX, YY, FF, levels=[0], colors = "black")

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

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++
```

---

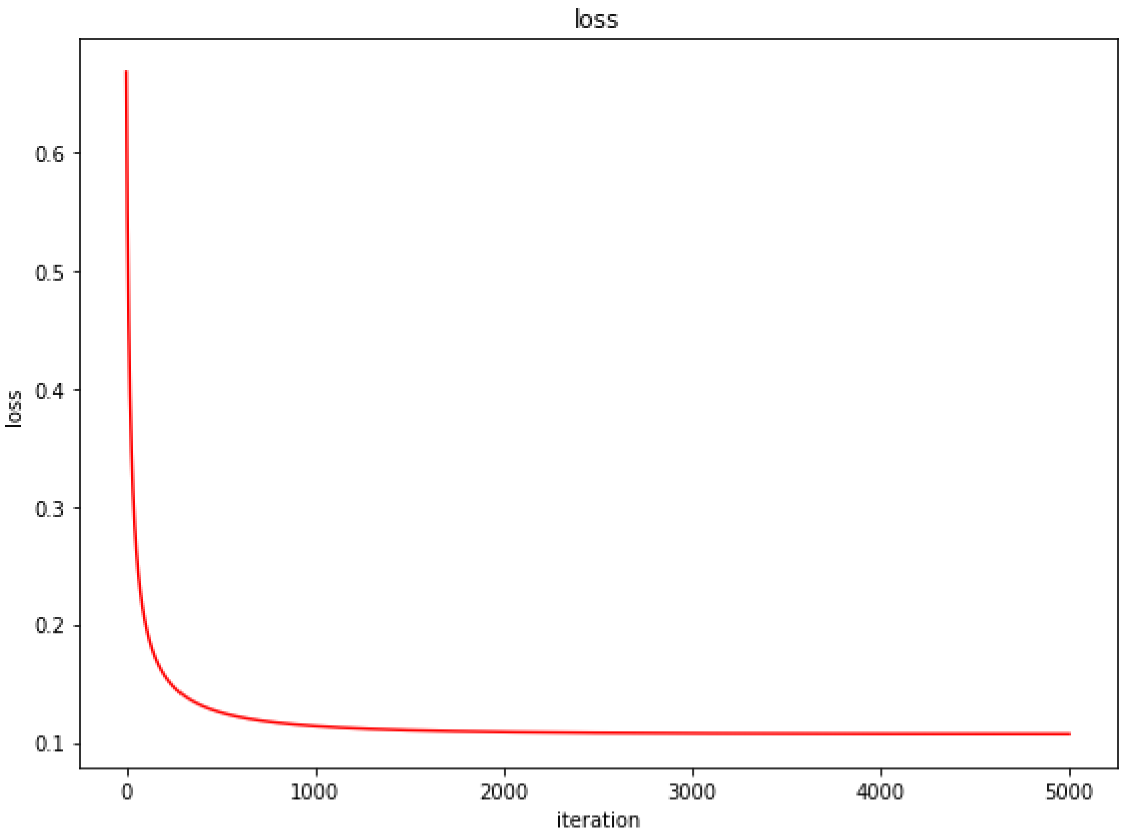
results

In [ ]:

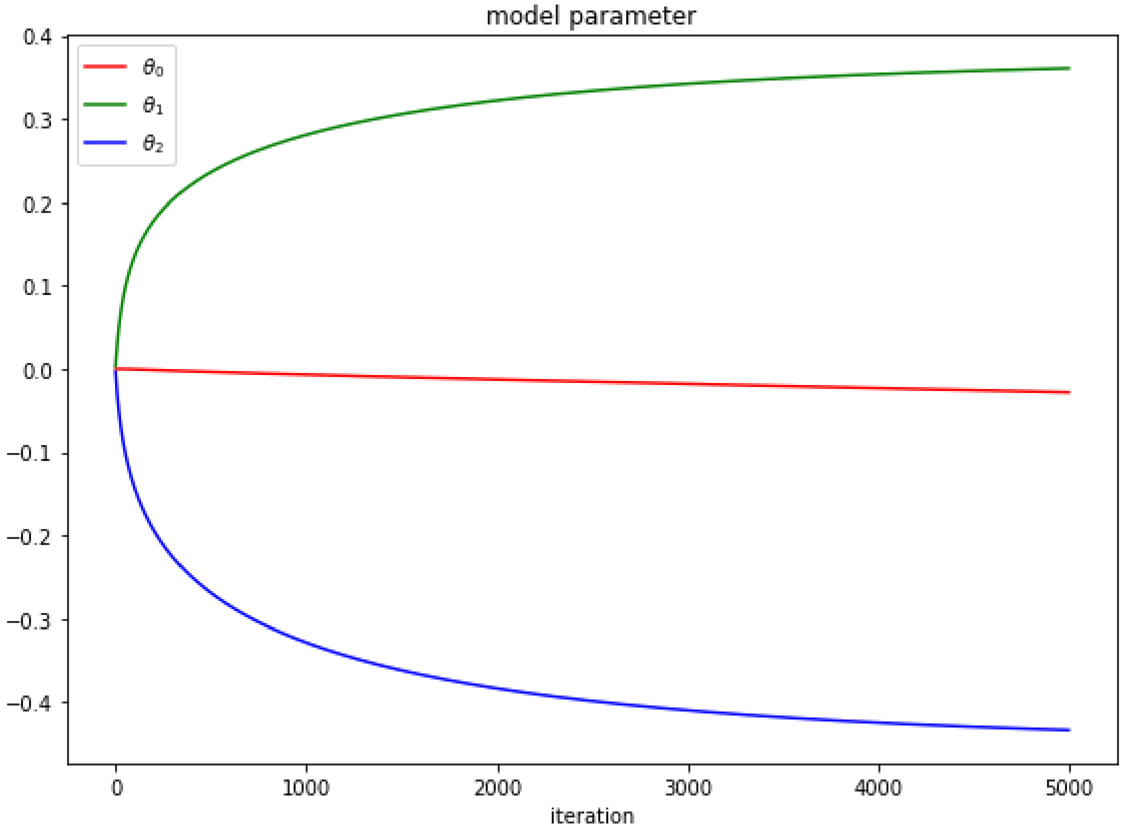
```
number_result = 9

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]
**************************************************
[ 1.4 -1.2]
**************************************************
## [RESULT 02]
**************************************************
[0.80218389 0.23147522]
**************************************************
## [RESULT 03]
**************************************************
[1.62041741 1.46328247]
**************************************************
## [RESULT 04]
```

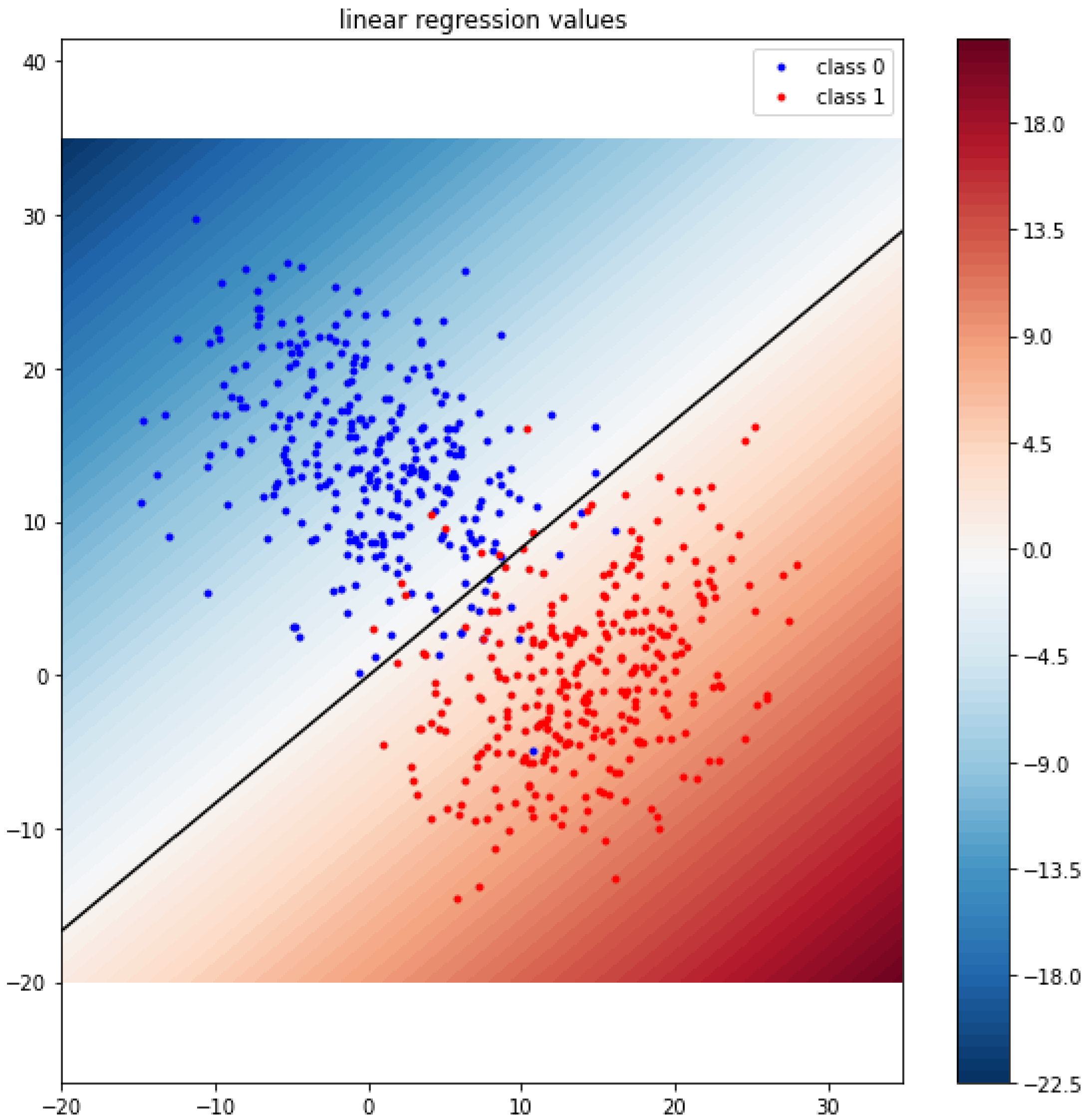
```
**************************************************
1.783699877256482
**************************************************
## [RESULT 05]
**************************************************
[[ 0.51682955  0.57070867  0.85606301]
 [-0.48317045  0.57070867  0.85606301]]
**************************************************
## [RESULT 06]
**************************************************
```



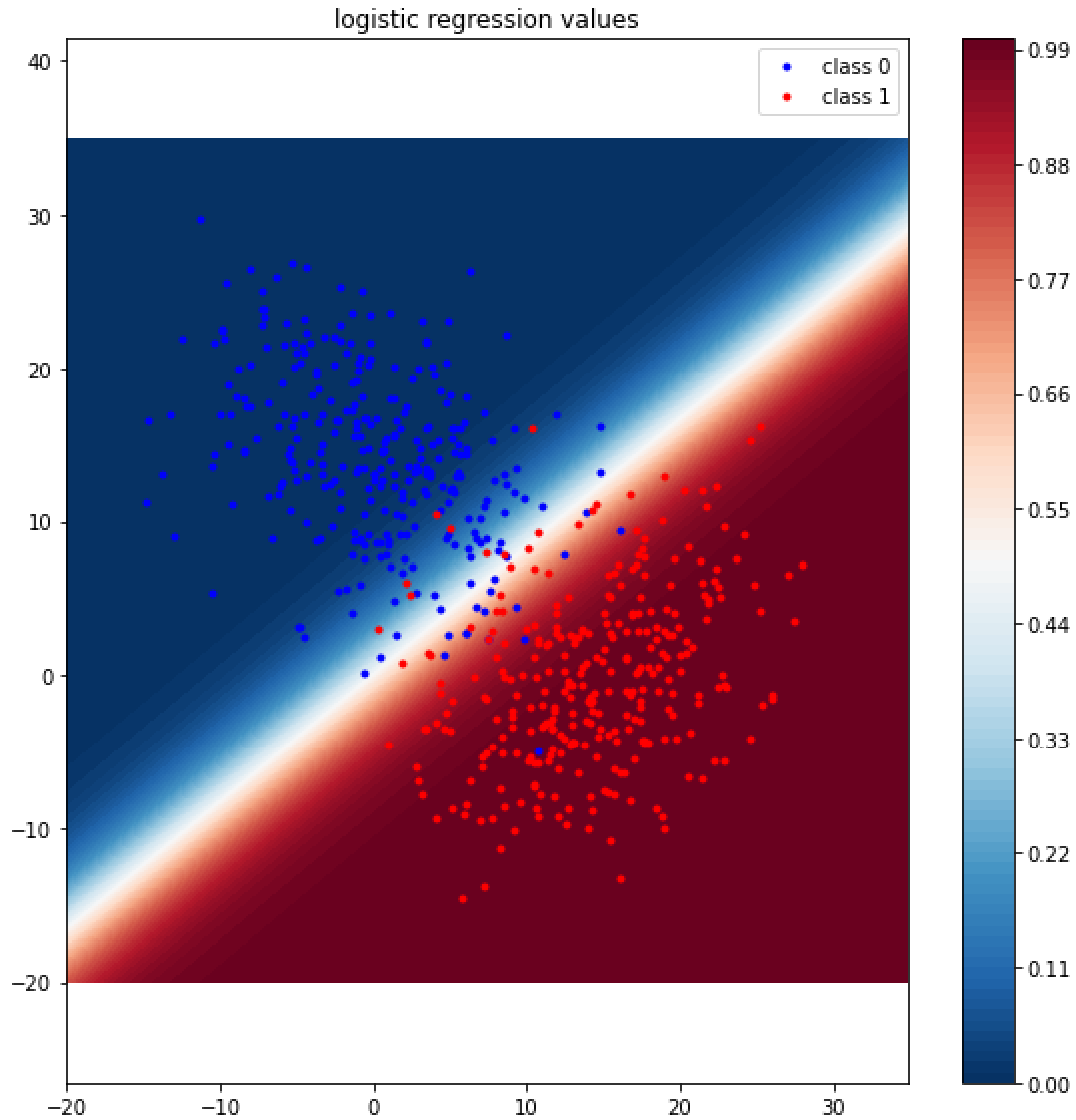

```
**************************************************
## [RESULT 07]
**************************************************
```


model parameter
θ_0
θ_1
θ_2
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
1000
2000
3000
4000
5000
iteration

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


linear regression values
class 0
class 1
40
30
20
10
0
−10
−20
−20
−10
0
10
20
30
18.0
13.5
9.0
4.5
0.0
−4.5
−9.0
−13.5
−18.0
−22.5

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



logistic regression values
class 0
class 1
40
30
20
10
0
−10
−20
−20
−10
0
10
20
30
0.99
0.88
0.77
0.66
0.55
0.44
0.33
0.22
0.11
0.00

In [ ]: