Logistic Regression

import library

In [2]:
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib import ticker, cm
```

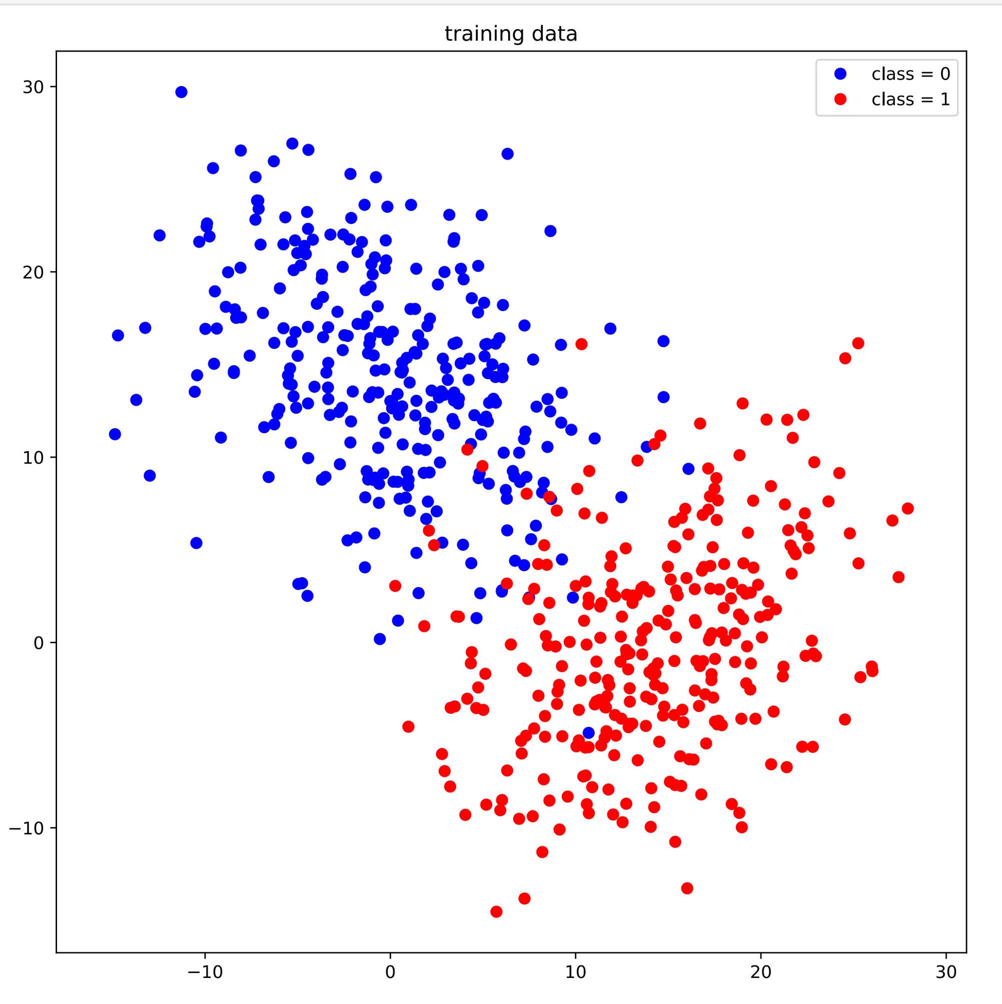
load training data

In [3]:
```python
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 [4]:
```python
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 [5]:
```python
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 [6]:
```python
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 [7]:
```python
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 [8]:
```python
def compute_residual(theta, point, label):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #-lilog(hi) -(1-li)log(1-hi)


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

    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 [34]:
```python
def compute_loss(theta, point, label):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    loss = np.sum(compute_residual(theta, point, label)) / len(point)
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    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 [50]:
```python
def compute_gradient(theta, point, label):

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


    gradient = np.sum(np.matmul(compute_logistic_regression(theta, point) - label ,point))/ len(point)
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return gradient
```

initialize the gradient descent algorithm

In [51]:
```python
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 [52]:
```python
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 [53]:
```python
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 [54]:
```python
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 [55]:
```python
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 [56]:
```python
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 [57]:
```python
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 [58]:
```python
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 [59]:
```python
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 [102…
```python
def function_result_08():

    X = np.arange(-20, 35, 0.1) # USE THIS VALUE for the range of x values in the construction of coordinate
    Y = np.arange(-20, 35, 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
    #
    print(XX.shape)
    print(YY.shape)
    plt.figure(figsize=(8,6))
    out = compute_linear_regression(theta_optimal,  np.array([np.ones(XX.shape), XX,YY]))


    plt.plot(out)
    plt.show()
```

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

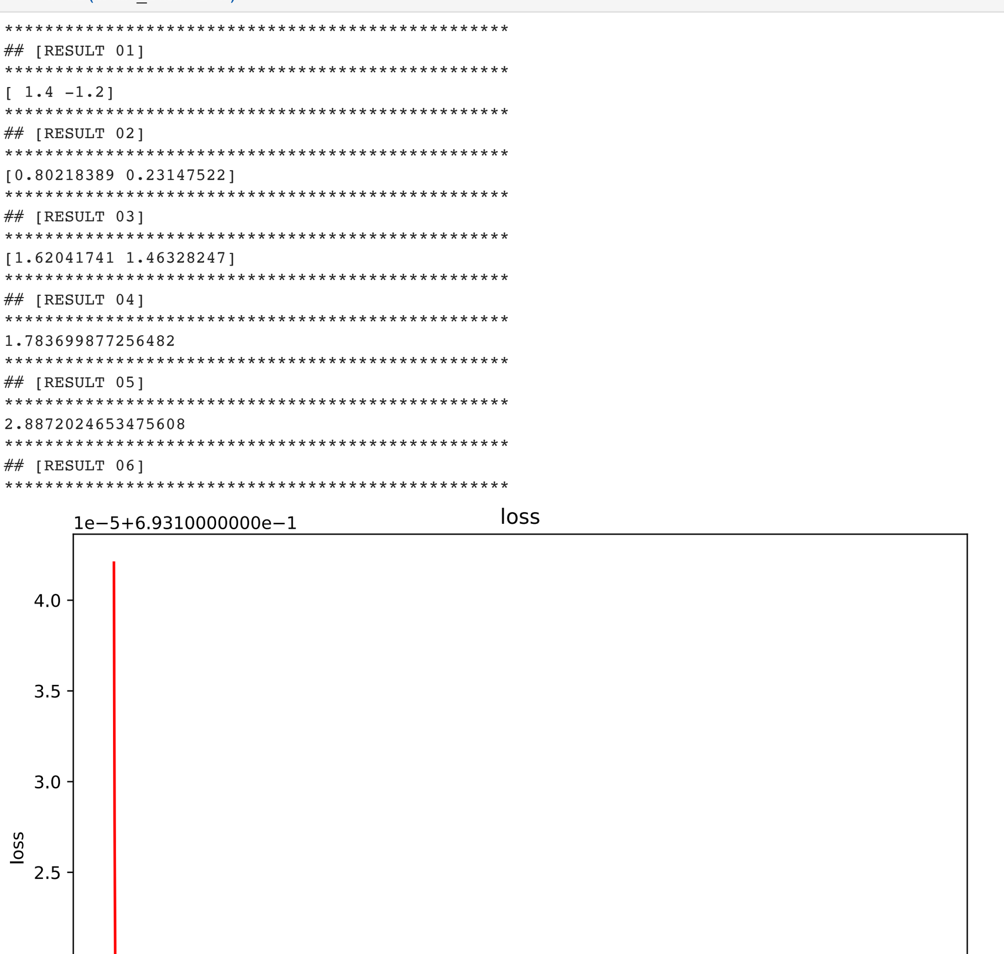
In [100…
```python
def function_result_09():

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


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

---

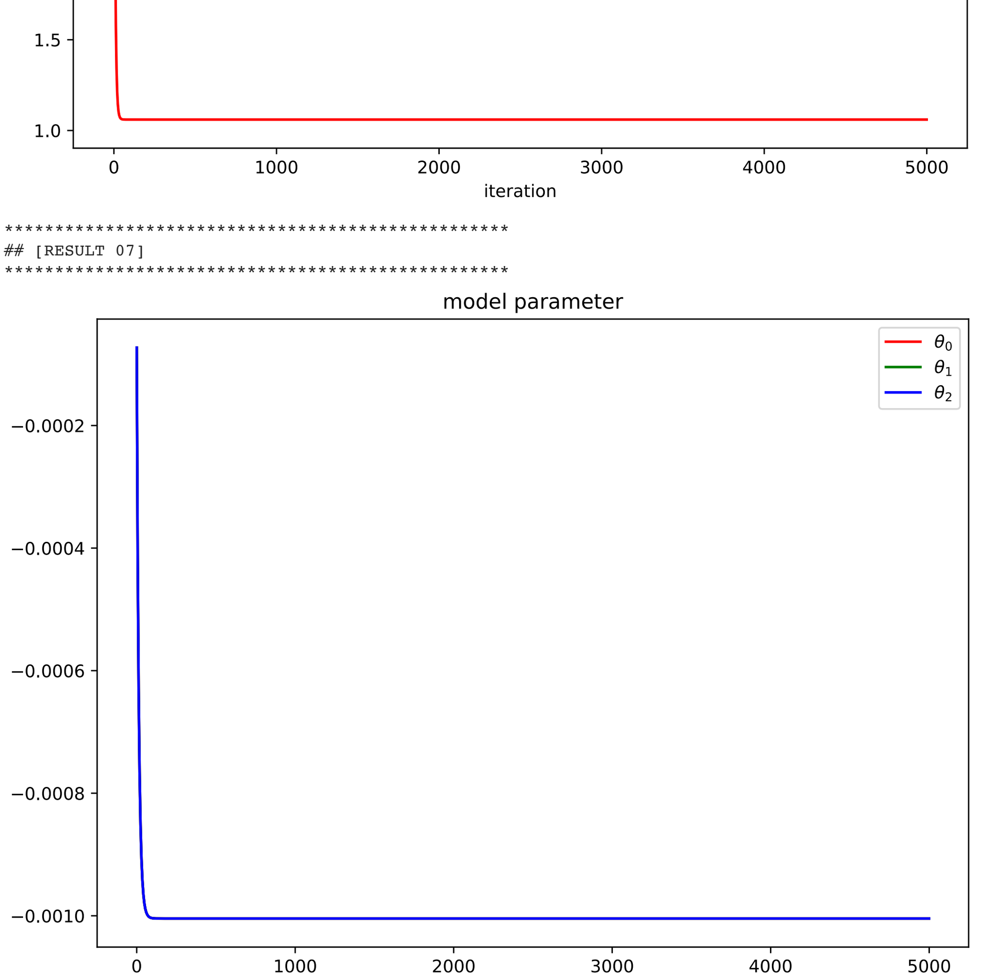
results

In [101…
```python
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]
**************************************************
2.8872024653475608
**************************************************
## [RESULT 06]
**************************************************
```




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




```
**************************************************
## [RESULT 08]
**************************************************
(550, 550)
(550, 550)
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [101], in <cell line: 3>()
      8 print(title)
      9 print('**************************************************')
---> 10 eval(name_function)

File <string>:1, in <module>

Input In [99], in function_result_08()
     12 print(YY.shape)
     13 plt.figure(figsize=(8,6))
---> 14 out = compute_linear_regression(theta_optimal,  np.array([1, XX,YY]))
     17 plt.plot(out)
     18 plt.show()

Input In [5], in compute_linear_regression(theta, point)
      1 def compute_linear_regression(theta, point):
      2
      3     # ++++++++++++++++++++++++++++++++++++++++++++++++++
      4     # complete the blanks
      5     #
----> 8     value = theta[0] * point[:, 0] + theta[1] * point[:, 1]  + theta[2] * point[:, 2]
      9     #
     10     # ++++++++++++++++++++++++++++++++++++++++++++++++++
     12     return value

IndexError: too many indices for array: array is 1-dimensional, but 2 were indexed
```

In [ ]:

In [ ]: