Logistic Regression

import library

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
```

load training data

In [2]:

```
fname_data = 'assignment_08_data.csv'

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

point_x = np.zeros(num_data)
point_y = np.zeros(num_data)
label   = np.zeros(num_data)

for i in range(num_data):

    point_x[i]  = data[i,0]
    point_y[i]  = data[i,1]
    label[i]    = data[i,2]
```

define linear regression function with inputs $\theta = (\theta_0, \theta_1, \theta_2)$ and $\text{point} = (1, x, y)$

In [3]:

```
def linear_regression(theta, x, y):
    num_data = x.shape[0]
    first = np.ones(num_data)

    X = np.column_stack([first, x, y])
    value = np.dot(X, theta)
    return value
```

define sigmoid function with input x

In [4]:

```
def sigmoid(x):
    for _ in range(len(x)) :
        z = 1/(1+np.exp(-x))
    return z
```

define loss function for the logistic regression

In [5]:

```
def compute_loss(theta, x, y, label):
    z = linear_regression(theta, x, y)
    h = sigmoid(z)
    loss = (-label * np.log(h) - (1 - label) * np.log(1 - h)).mean()
    return loss
```

define gradient vector for the model parameters $\theta = (\theta_0, \theta_1, \theta_2)$

In [6]:

```
def compute_gradient(theta, x, y, label):
    z = linear_regression(theta, x, y)
    h = sigmoid(z)
    num_data = x.shape[0]
    first = np.ones(num_data)
    X = np.column_stack([first, x, y])
    gradient = np.dot(X.T, (h - label)) / label.shape[0]
    return gradient
```

gradient descent for the model parameters $\theta = (\theta_0, \theta_1, \theta_2)$

In [7]:

```
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))   # USE THIS VALUE for the initial condition of the model para

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

iterations for the gradient descent

In [8]:

```
for i in range(num_iteration):
    theta = theta - learning_rate * compute_gradient(theta, point_x, point_y, label)
    loss = compute_loss(theta, point_x, point_y, label)
    theta_iteration[i] = theta
    loss_iteration[i] = loss
    print("iteration = %4d, loss = %5.5f" % (i, loss))
```

```
iteration =    0, loss = 0.66813
iteration =    1, loss = 0.64476
iteration =    2, loss = 0.62295
iteration =    3, loss = 0.60256
iteration =    4, loss = 0.58350
iteration =    5, loss = 0.56567
iteration =    6, loss = 0.54897
iteration =    7, loss = 0.53332
iteration =    8, loss = 0.51864
iteration =    9, loss = 0.50485
iteration =   10, loss = 0.49188
iteration =   11, loss = 0.47967
iteration =   12, loss = 0.46817
iteration =   13, loss = 0.45731
iteration =   14, loss = 0.44706
iteration =   15, loss = 0.43736
iteration =   16, loss = 0.42817
iteration =   17, loss = 0.41947
iteration =   18, loss = 0.41121
```

plot the results

In [9]:

```
def plot_loss_curve(loss_iteration):

    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 [10]:

```
def plot_data(point_x, point_y, label):
    plt.figure(figsize=(8,8))

    plt.title('training data')
    xx = []
    yy = []
    xxx = []
    yyy = []
    for i in range(0, num_data) :
        x = point_x[i]
        y = point_y[i]
        if label[i] == 0 :
            xx.append(x)
            yy.append(y)
        elif label[i] == 1 :
            xxx.append(x)
            yyy.append(y)
    plt.scatter(xx,yy,c='blue', label = 'Class = 0')
    plt.scatter(xxx,yyy,c='red', label = 'Class = 1')

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

In [11]:

```
def plot_model_parameter(theta_iteration):

    plt.figure(figsize=(8,6))
    plt.title('model parameter')
    thetaT = theta_iteration.T
    plt.plot(thetaT[0], '-', color = 'red', label = 'theta0')
    plt.plot(thetaT[1], '-', color = 'green', label = 'theta1')
    plt.plot(thetaT[2], '-', color = 'blue', label = 'theta2')

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

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

In [45]:

```
def plot_classifier(point_x, point_y, label):
    plt.figure(figsize=(8,8))
    xx = []
    yy = []
    xxx = []
    yyy = []
    X = np.arange(point_x.min(), point_x.max(), 0.5)
    Y = np.arange(point_y.min(), point_y.max(), 0.5)
    XX, YY = np.meshgrid(X,Y)
    Z = linear_regression(theta,XX,YY)
    CS = plt.contourf(X,Y,Z,levels=np.arrange(-1,1,0.05), alpha=0.5, cmap='seismic')
    for i in range(0, num_data) :
        x = point_x[i]
        y = point_y[i]
        if label[i] == 0 :
            xx.append(x)
            yy.append(y)
        elif label[i] == 1 :
            xxx.append(x)
            yyy.append(y)


    plt.scatter(xx, yy, color='b', label='Class = 0')
    plt.scatter(xxx, yyy, color='r', label='Class = 1')
    z = linear_regression(theta, point_x, point_y)



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

*** ***

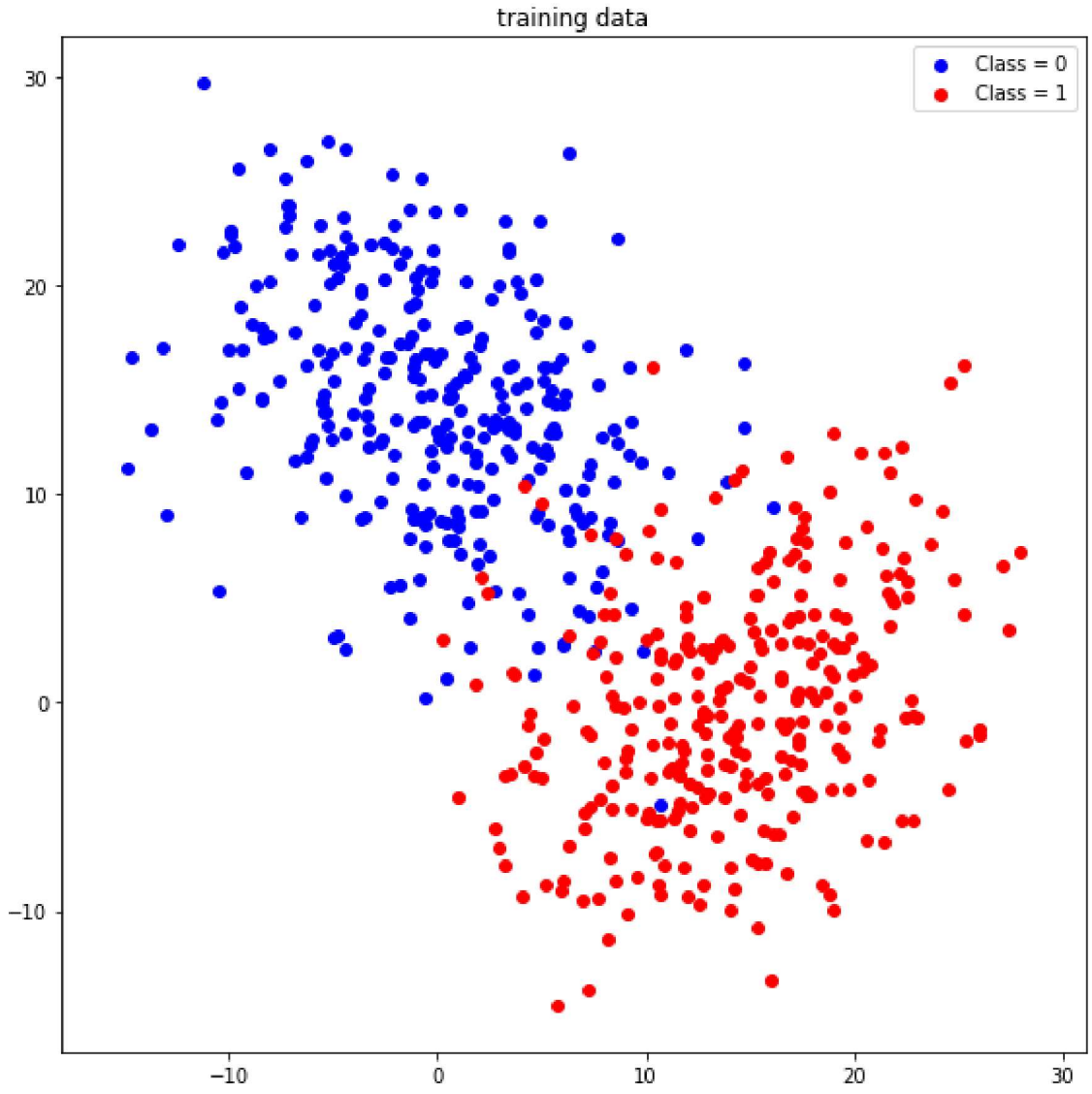
* results

*** ***

01. plot the input data point in blue for class 0 and in red for class 1

In [13]:

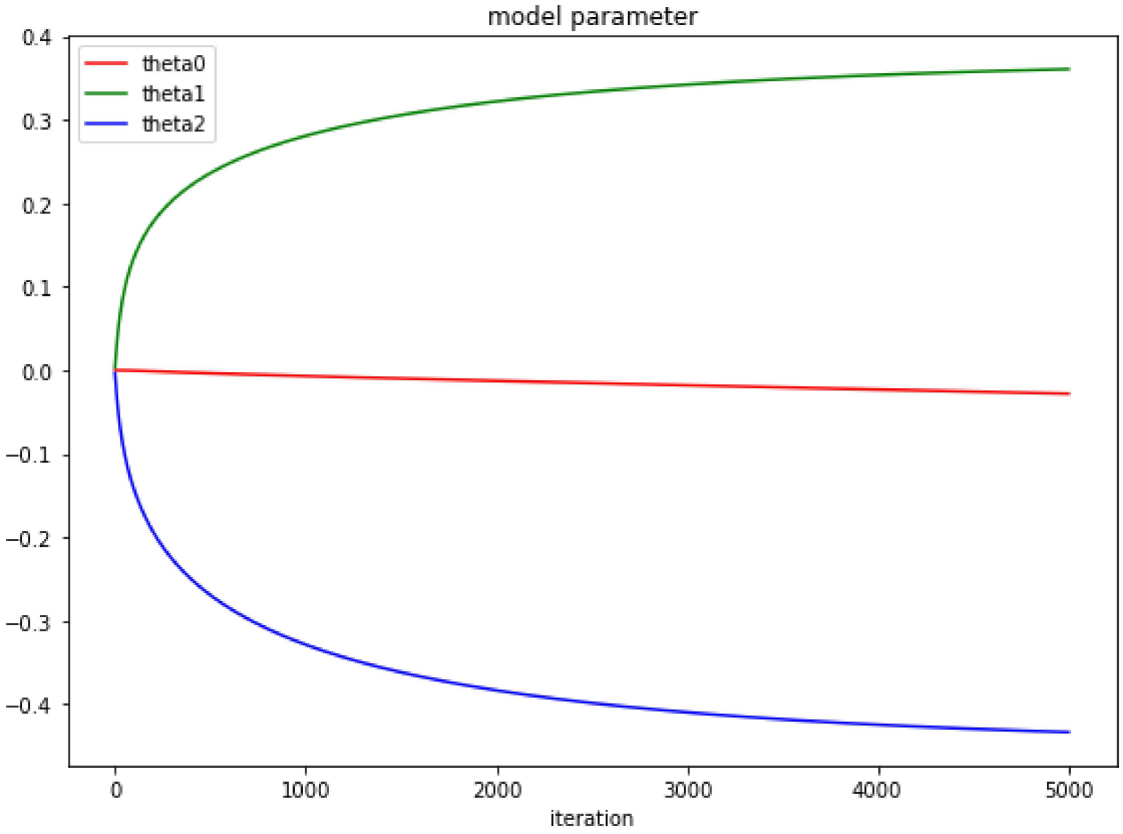
```
plot_data(point_x, point_y, label)
```



 # 02. plot the values of the model parameters θ_0 in red curve, θ_1 in green curve, and θ_2 in blue curve over the gradient descent iterations

In [14]:

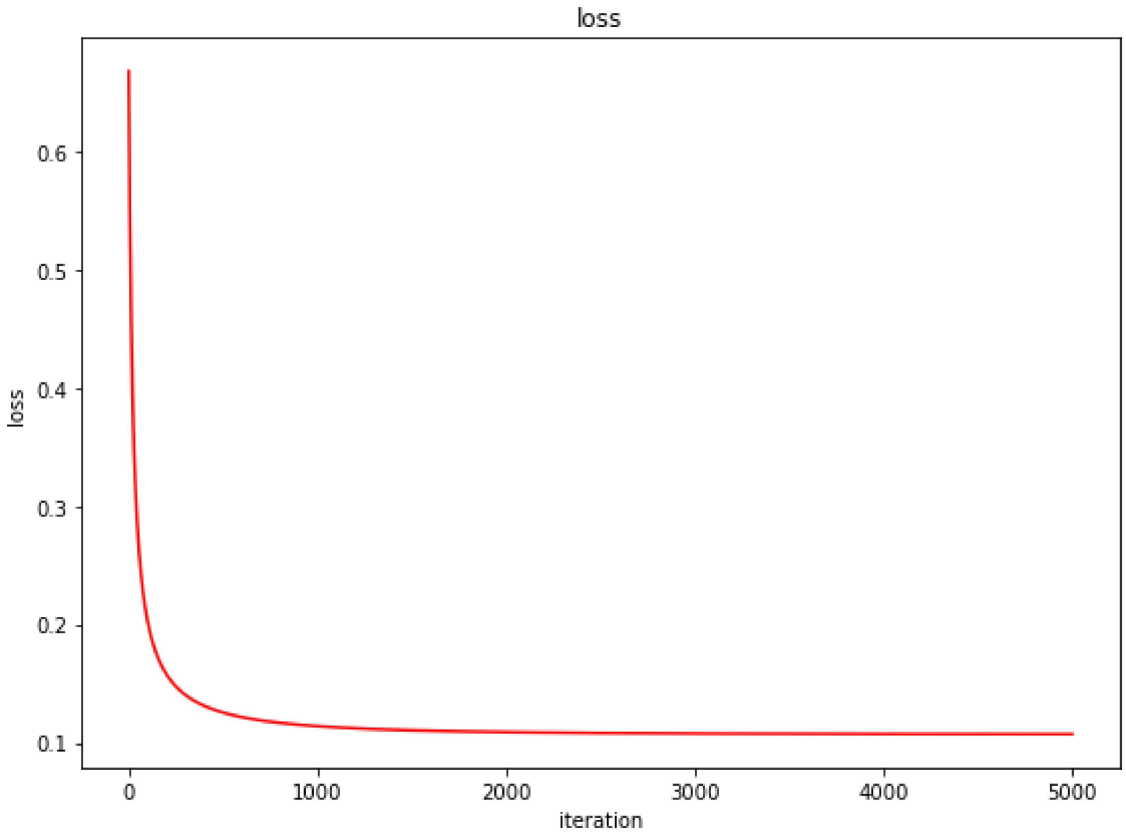
```
plot_model_parameter(theta_iteration)
```



03. plot the loss values $\mathcal{L}(\theta)$ in red curve over the gradient descent iterations

In [15]:

```
plot_loss_curve(loss_iteration)
```



04. plot the classifier with the given data points superimposed

In [46]:

```
plot_classifier(point_x, point_y, label)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-46-b379f8b38353> in <module>
----> 1 plot_classifier(point_x, point_y, label)

<ipython-input-45-6af590e51ee2> in plot_classifier(point_x, point_y, label)
      8     Y = np.arange(point_y.min(), point_y.max(), 0.5)
      9     XX, YY = np.meshgrid(X,Y)
---> 10       Z = linear_regression(theta,XX,YY)
     11     CS = plt.contourf(X,Y,Z,levels=np.arrange(-1,1,0.05), alpha=0.5, cmap
='seismic')
     12     for i in range(0, num_data) :

<ipython-input-3-0f41b277279a> in linear_regression(theta, x, y)
      4
      5     X = np.column_stack([first, x, y])
----> 6       value = np.dot(X, theta)
      7     return value

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (89,173) and (3,) not aligned: 173 (dim 1) != 3 (dim 0)
```

<Figure size 576x576 with 0 Axes>

In [ ]: