Binary classification based on Logistic Regression with a quadratic regularization

import library

In [1]:

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

load data

In [2]:

```
fname_data_train    = 'assignment_10_data_train.csv'
fname_data_test     = 'assignment_10_data_test.csv'

data_train  = np.genfromtxt(fname_data_train, delimiter=',')
data_test   = np.genfromtxt(fname_data_test, delimiter=',')

num_data_train = data_train.shape[0]
num_data_test = data_test.shape[0]

x1 = np.zeros(num_data_train)
y1 = np.zeros(num_data_train)
label1   = np.zeros(num_data_train)

for i in range(num_data_train):
    x1[i]  = data_train[i,0]
    y1[i]  = data_train[i,1]
    label1[i]    = data_train[i,2]

###############################################################################

x2 = np.zeros(num_data_test)
y2 = np.zeros(num_data_test)
label2   = np.zeros(num_data_test)

for i in range(num_data_test):
    x2[i]  = data_test[i,0]
    y2[i]  = data_test[i,1]
    label2[i]    = data_test[i,2]
```

plot the data

In [ ]:

```

```

define feature function

In [3]:

```
def feature_function(x, y):
    feature = np.array([1, x, y, x*y, x*x, y*y, x*x*y, x*y*y, x*x*x, y*y*y], dtype = object)
    return feature
```

define regression function based on the feature function

In [4]:

```
def regression_function(theta, feature):
    value = np.matmul(np.transpose(theta), feature)
    return value
```

define regularization function on the model parameters

In [5]:

```
def regularization_function(theta):
    value = np.matmul(theta, theta)
    return value
```

define sigmoid function

In [6]:

```
def logistic_function(x):
    z = 1/(1+np.exp(-x))
    return z
```

define loss function where α is a weight for the quadratic regularization term (Note that you need to add a small number (np.finfo(float).eps) inside logarithm function in order to avoid $\log(0)$)

In [7]:

```
def compute_loss_feature(theta, feature, label, alpha):
    z = regression_function(theta, feature)
    h = logistic_function(z)
    loss = (-label * np.log(h + np.finfo(float).eps) - (1 - label) * np.log(1 - h + np.finfo(float)
    return loss
```

define gradient vector for the model parameters with the quadratic regularization term whose weight is α

In [8]:

```
def compute_gradient_feature(theta, feature, label, alpha):
    z = regression_function(theta, feature)
    h = logistic_function(z)
    num_data = feature[1].shape[0]
    one = np.ones(num_data)
    X = np.column_stack([one, feature[1], feature[2], feature[3], feature[4], feature[5], feature[6]
    gradient = np.dot(X.T, (h - label)) / num_data
    return gradient
```

compute the accuracy

In [9]:

```
def compute_accuracy(theta, feature, label):
    num_data = feature[1].shape[0]
    correctNum = 0
    f = regression_function(theta, feature)
    for i in range(0, num_data)  :
        if f[i]>=0 :
            ans = 1
        else :
            ans = 0
        if ans == label[i] :
            correctNum += 1
    accuracy = correctNum / num_data
    return round(accuracy, 5)
```

gradient descent for the model parameters θ

In [10]:

```
num_iteration   = 30000
learning_rate   = 0.3
alpha           = 0.001

theta           = np.array((0, 0, 0, 0, 0, 0, 0, 0, 0, 0))
theta_iteration = np.zeros((num_iteration, theta.size))
loss1_iteration  = np.zeros(num_iteration)
loss2_iteration  = np.zeros(num_iteration)
```

In [11]:

```
# theta_iteration             = np.zeros((num_iteration, dim_feature))
# loss_iteration_train        = np.zeros(num_iteration)
# loss_iteration_test         = np.zeros(num_iteration)
accuracy_iteration_train    = np.zeros(num_iteration)
accuracy_iteration_test     = np.zeros(num_iteration)
```

In [12]:

```
for i in range(num_iteration):
    theta = theta - learning_rate * compute_gradient_feature(theta, feature_function(x1, y1), label
    loss1 = compute_loss_feature(theta, feature_function(x1, y1), label1, alpha)
    theta_iteration[i] = theta
    loss1_iteration[i] = loss1

    loss2 = compute_loss_feature(theta, feature_function(x2, y2), label2, alpha)
    loss2_iteration[i] = loss2

    accuracy1 = compute_accuracy(theta, feature_function(x1, y1), label1)
    accuracy2 = compute_accuracy(theta, feature_function(x2, y2), label2)

    accuracy_iteration_train[i] = accuracy1
    accuracy_iteration_test[i] = accuracy2

    print("[%4d] loss(train) = %5.5f, loss(test) = %5.5f, acc(train) = %5.5f, acc(test) = %5.5f" % (

theta_optimal = theta
```

```
st) = 0.93400
[29991] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
[29992] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
[29993] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
[29994] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
[29995] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
[29996] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
[29997] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
[29998] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
[29999] loss(train) = 0.14333, loss(test) = 0.16530, acc(train) = 0.94400, acc(te
st) = 0.93400
```

compute accuracy of the classifiers

In [34]:

```
accuracy_train  = compute_accuracy(theta_optimal, feature_function(x1, y1), label1)
accuracy_test   = compute_accuracy(theta_optimal, feature_function(x2, y2), label2)
```

```
<class 'float'>
```

plot the results

In [14]:

```
def plot_loss_curve(loss_iteration_train, loss_iteration_test):

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

    plt.plot(loss_iteration_train, '-', color = 'red', label = 'train')
    plt.plot(loss_iteration_test, '-', color = 'blue', label = 'test')
    plt.xlabel('iteration')
    plt.ylabel('loss')

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

In [15]:

```
def plot_accuracy_curve(accuracy_iteration_train, accuracy_iteration_test):

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

    plt.plot(accuracy_iteration_train, '-', color = 'red', label = 'train')
    plt.plot(accuracy_iteration_test, '-', color = 'blue', label = 'test')
    plt.xlabel('iteration')
    plt.ylabel('accuracy')

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

In [16]:

```
def plot_data(data_train, data_test):

    f = plt.figure(figsize=(16,8))
    ax1 = f.add_subplot(1, 2, 1)
    ax2 = f.add_subplot(1, 2, 2)

    num_data_train = data_train.shape[0]
    num_data_test = data_test.shape[0]

    x1 = np.zeros(num_data_train)
    y1 = np.zeros(num_data_train)
    label1   = np.zeros(num_data_train)

    for i in range(num_data_train):
        x1[i]  = data_train[i,0]
        y1[i]  = data_train[i,1]
        label1[i]    = data_train[i,2]

    ########################################################################

    x2 = np.zeros(num_data_test)
    y2 = np.zeros(num_data_test)
    label2   = np.zeros(num_data_test)

    for i in range(num_data_test):
        x2[i]  = data_test[i,0]
        y2[i]  = data_test[i,1]
        label2[i]    = data_test[i,2]

    plt.title('training data')
    xx = []
    yy = []
    xxx = []
    yyy = []
    for i in range(0, num_data_train) :
        x = x1[i]
        y = y1[i]
        if label1[i] == 0 :
            xx.append(x)
            yy.append(y)
        elif label1[i] == 1 :
            xxx.append(x)
            yyy.append(y)
    ax1.scatter(xx,yy,c='blue', label = 'Class = 0')
    ax1.scatter(xxx,yyy,c='red', label = 'Class = 1')

    ############################################################

    plt.title('testing data')
    xx = []
    yy = []
    xxx = []
    yyy = []
    for i in range(0, num_data_test) :
        x = x2[i]
        y = y2[i]
        if label2[i] == 0 :
            xx.append(x)
            yy.append(y)
```

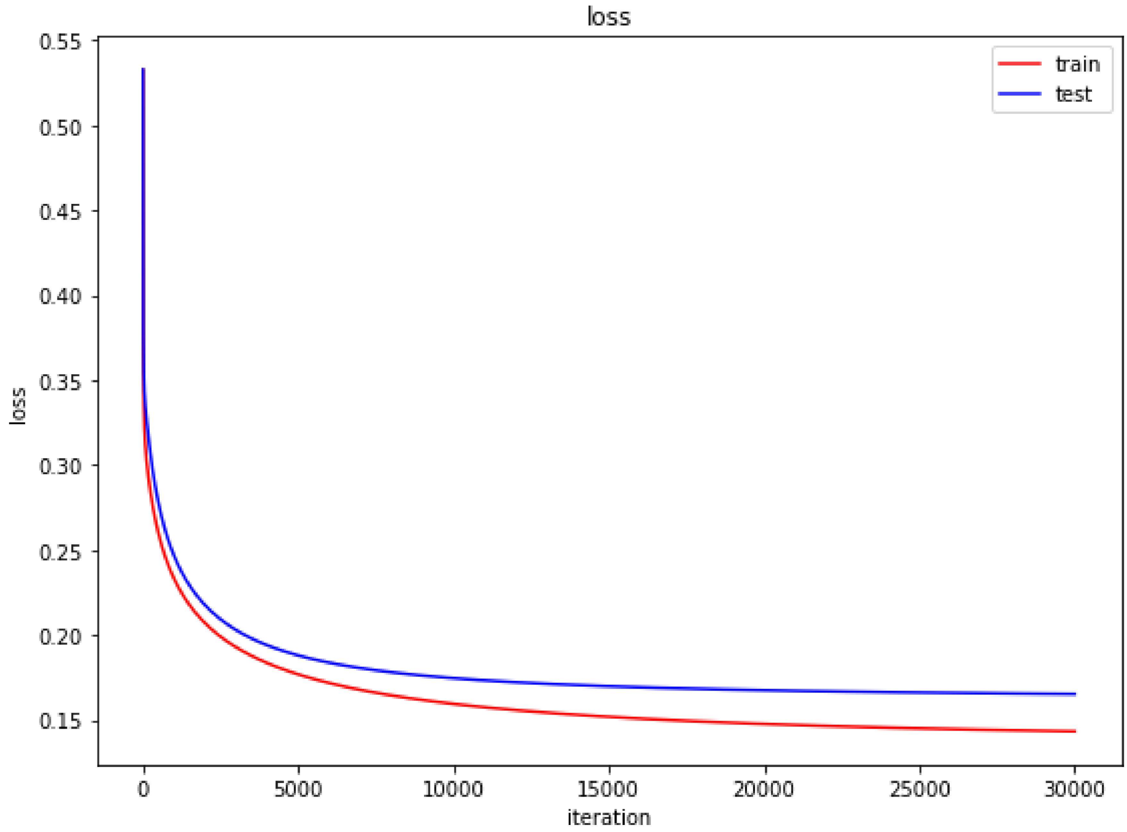
```
        elif label2[i] == 1 :
            xxx.append(x)
            yyy.append(y)
    ax2.scatter(xx,yy,c='blue', label = 'Class = 0')
    ax2.scatter(xxx,yyy,c='red', label = 'Class = 1')

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

In [17]:

```
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 = r'$₩theta_0$')
    plt.plot(thetaT[1], '-', color = 'green', label = r'$₩theta_1$')
    plt.plot(thetaT[2], '-', color = 'blue', label = r'$₩theta_2$')
    plt.plot(thetaT[3], '-', color = 'black', label = r'$₩theta_3$')
    plt.plot(thetaT[4], '-', color = 'orange', label = r'$₩theta_4$')
    plt.plot(thetaT[5], '-', color = 'skyblue', label = r'$₩theta_5$')
    plt.plot(thetaT[6], '-', color = 'pink', label = r'$₩theta_6$')
    plt.plot(thetaT[7], '-', color = 'brown', label = r'$₩theta_7$')
    plt.plot(thetaT[8], '-', color = 'purple', label = r'$₩theta_8$')
    plt.plot(thetaT[9], '-', color = 'darkblue', label = r'$₩theta_9$')

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

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

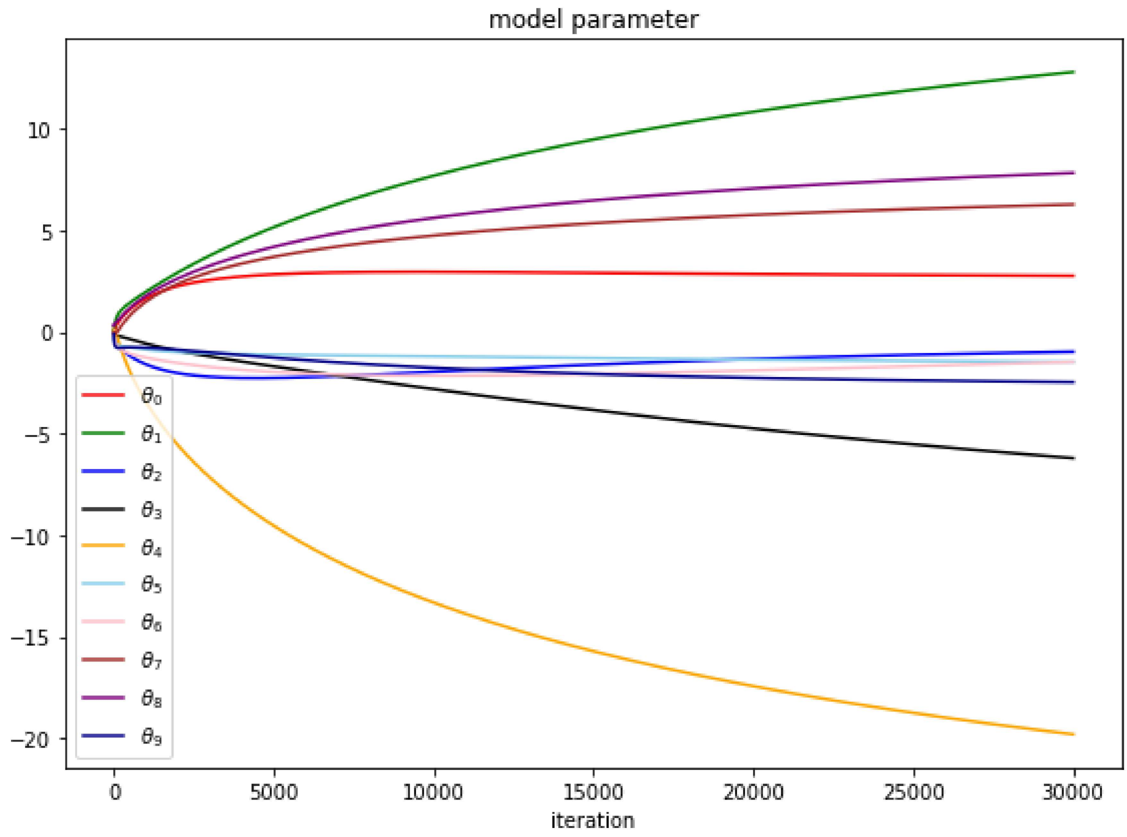
In [18]:

```
plot_loss_curve(loss1_iteration, loss2_iteration)
```



In [19]:

```
plot_model_parameter(theta_iteration)
```

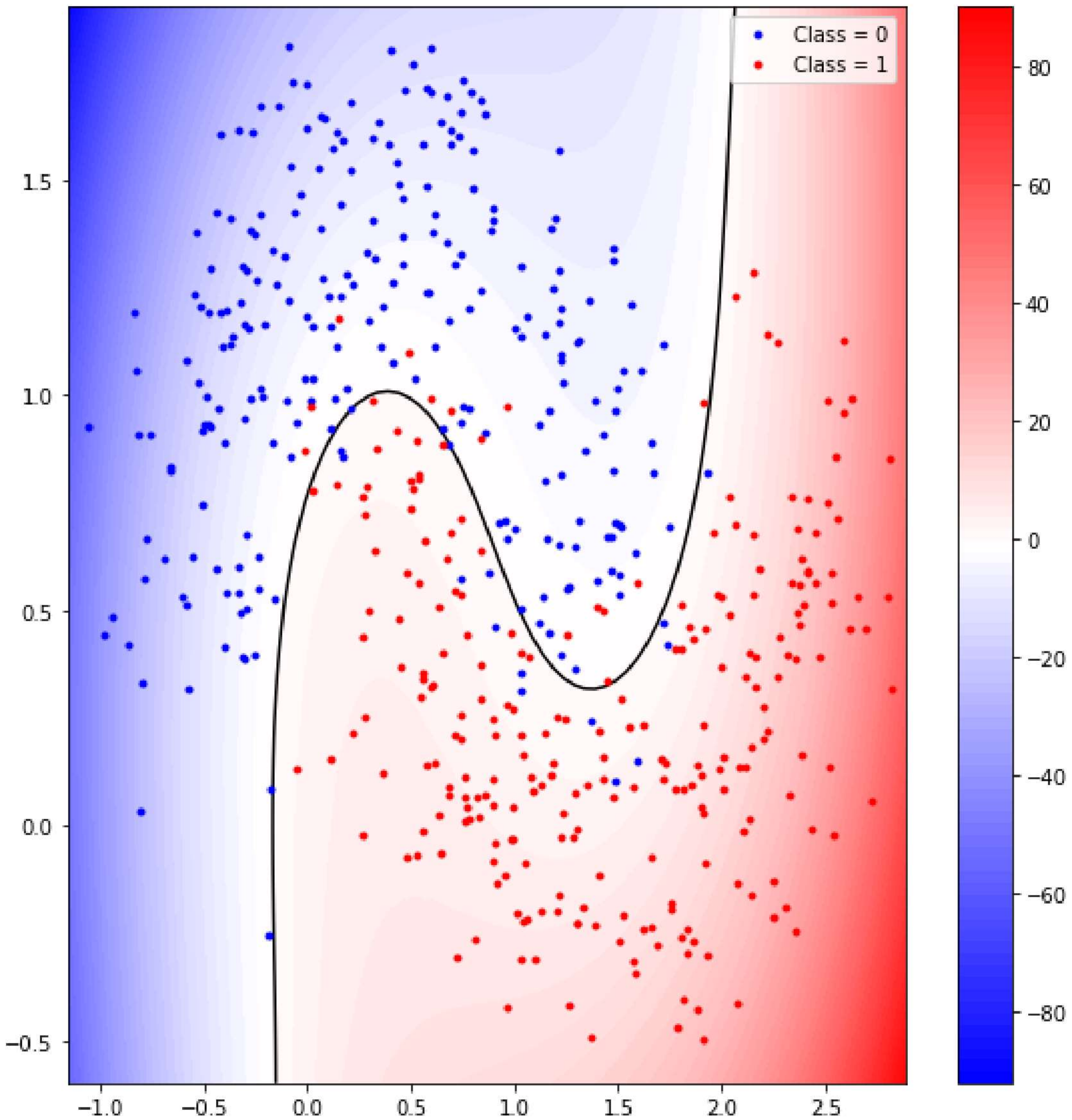


In [20]:

```
def plot_classifier(data, theta):

    plt.figure(figsize=(8,8))

    num_data = data.shape[0]

    x = np.zeros(num_data)
    y = np.zeros(num_data)
    label   = np.zeros(num_data)

    for i in range(num_data):
        x[i]  = data[i,0]
        y[i]  = data[i,1]
        label[i]    = data[i,2]

    xx = []
    yy = []
    xxx = []
    yyy = []
    X = np.arange(x.min()-0.1, x.max()+0.1, 0.05)
    Y = np.arange(y.min()-0.1, y.max()+0.1, 0.05)
    gX, gY = np.meshgrid(X, Y)
    Z = regression_function(theta, feature_function(gX, gY))
    for i in range(0, num_data) :
        a = x[i]
        b = y[i]
        if label[i] == 0 :
            xx.append(a)
            yy.append(b)
        elif label[i] == 1 :
            xxx.append(a)
            yyy.append(b)
    plt.plot(xx, yy, 'o',color='blue', label='Class = 0', markersize = 3)
    plt.plot(xxx, yyy, 'o',color='red', label='Class = 1', markersize = 3)
    plt.contourf(gX, gY, Z, levels=100, cmap = 'bwr')
    plt.colorbar()
    plt.contour(gX,gY, Z, levels = [0], colors = 'black')

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

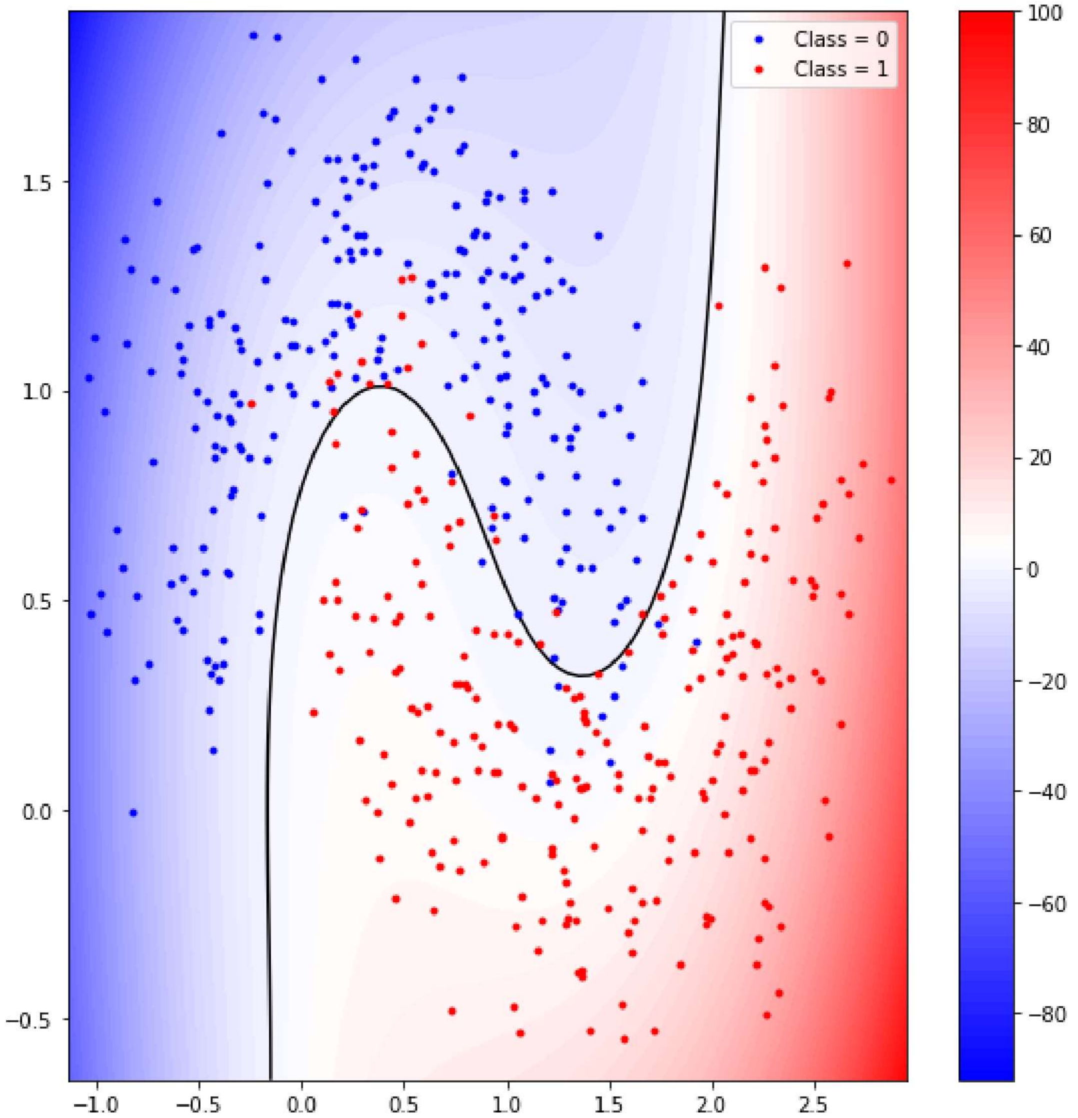
In [21]:

```
plot_classifier(data_train, theta_optimal)
```



In [22]:

```
plot_classifier(data_test, theta_optimal)
```

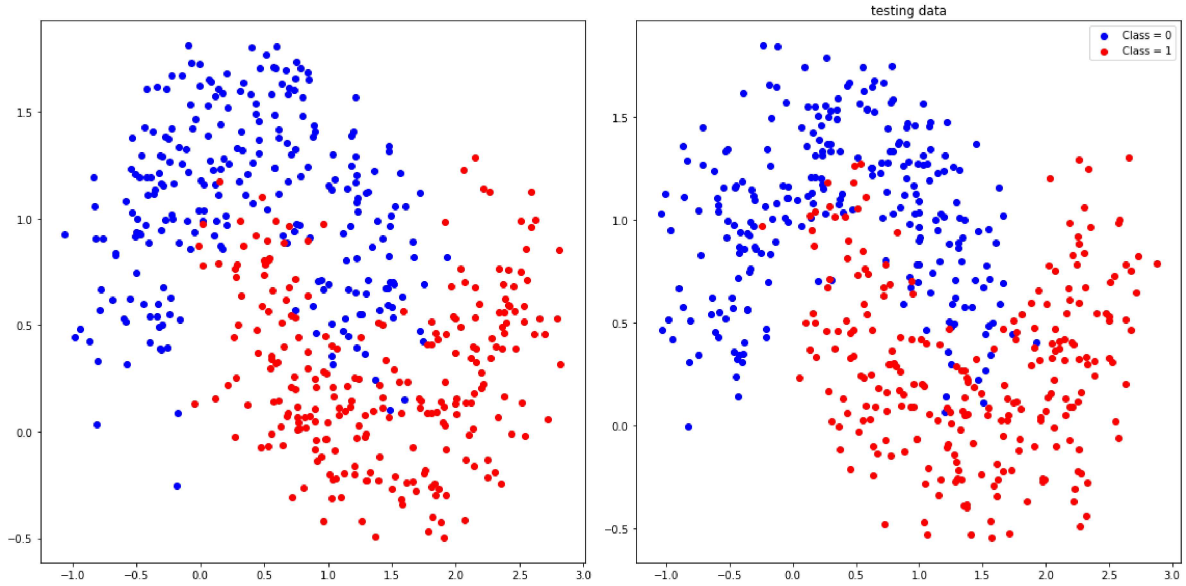


results

1. plot the input data (training on the left sub-figure and testing on the right sub-figure) in blue for class 0 and in red for class 1 from the file [assignment_10_data_train.csv] and [assignment_10_data_test.csv], respectively,

In [23]:

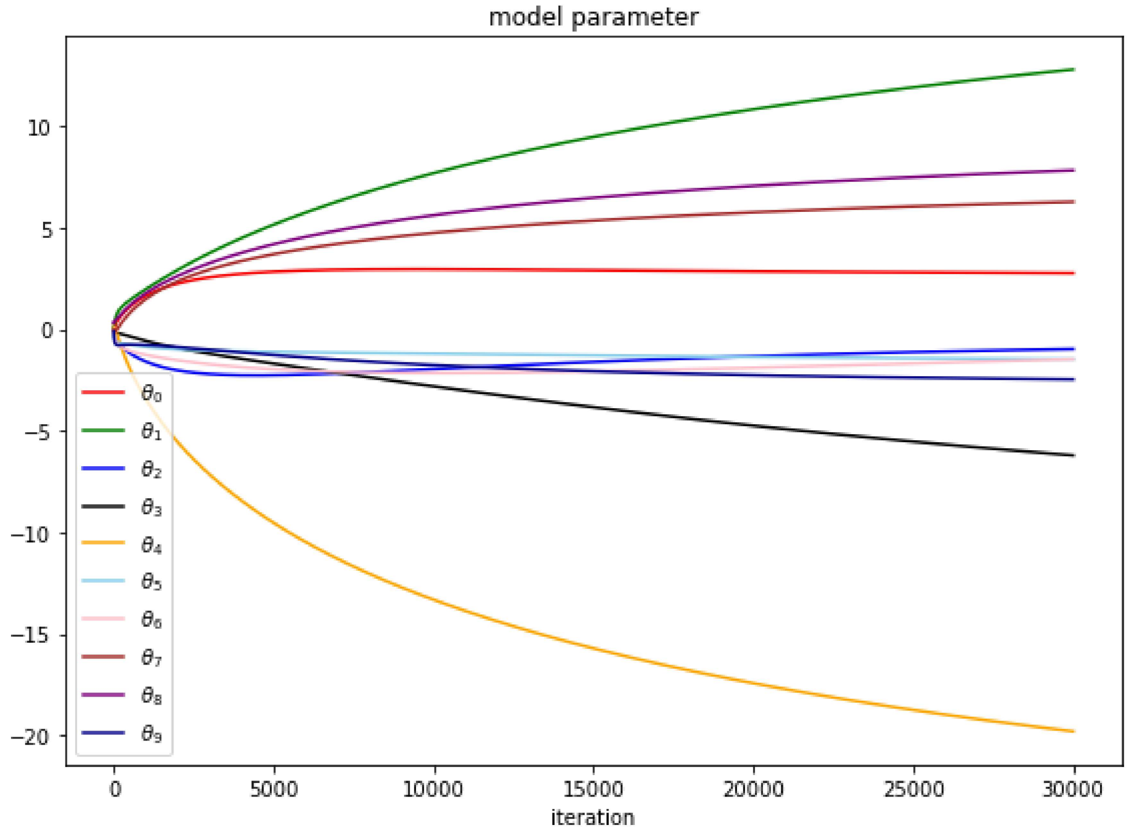
```
plot_data(data_train, data_test)
```



2. plot the values of the model parameters θ as curves over the gradient descent iterations using different colors

In [24]:

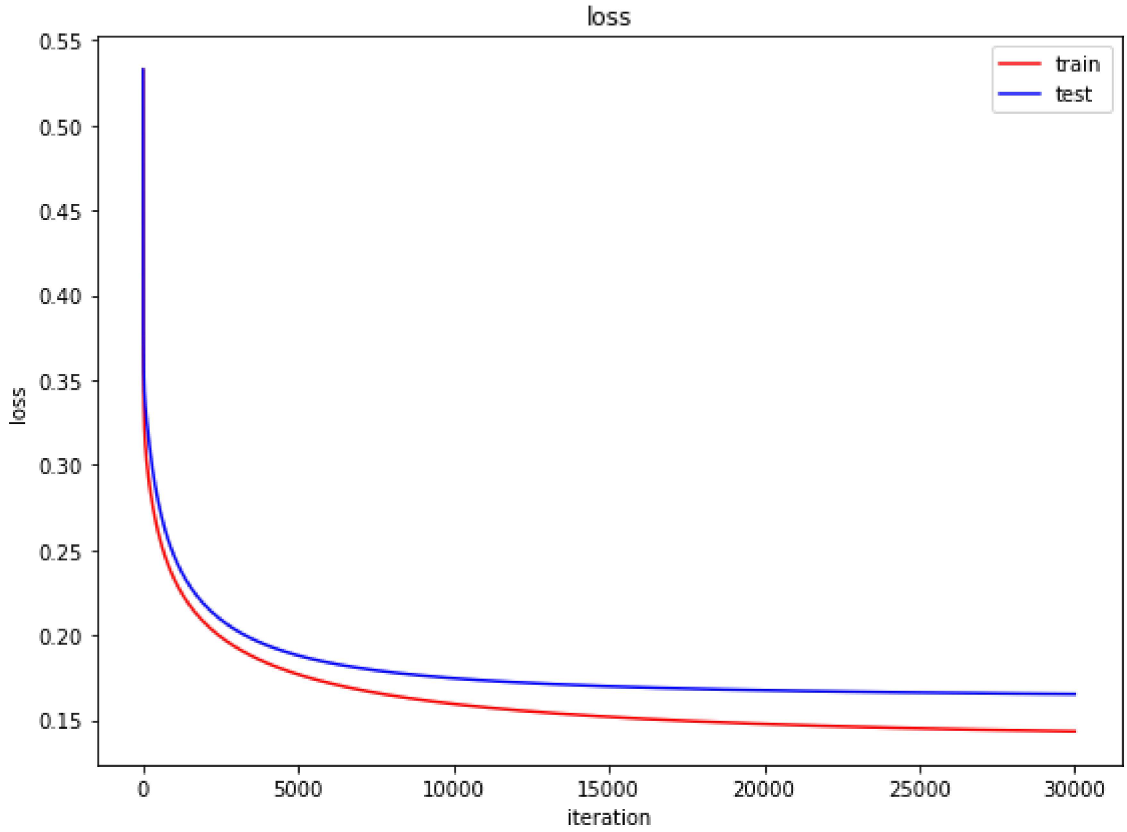
```
plot_model_parameter(theta_iteration)
```



3. plot the training loss in red curve and the testing loss in blue curve over the gradient descent iterations

In [25]:

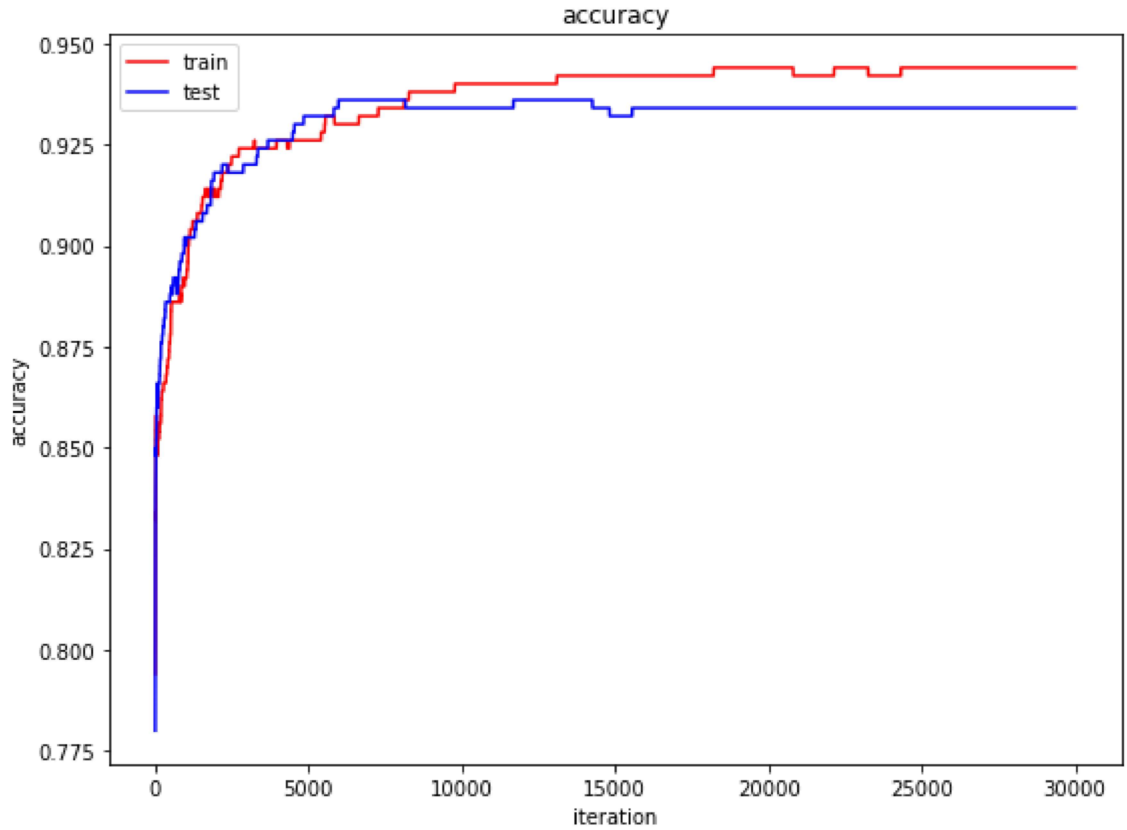
```
plot_loss_curve(loss1_iteration, loss2_iteration)
```



4. plot the training accuracy in red curve and the testing accuracy in blue curve over the gradient descent iterations

In [26]:

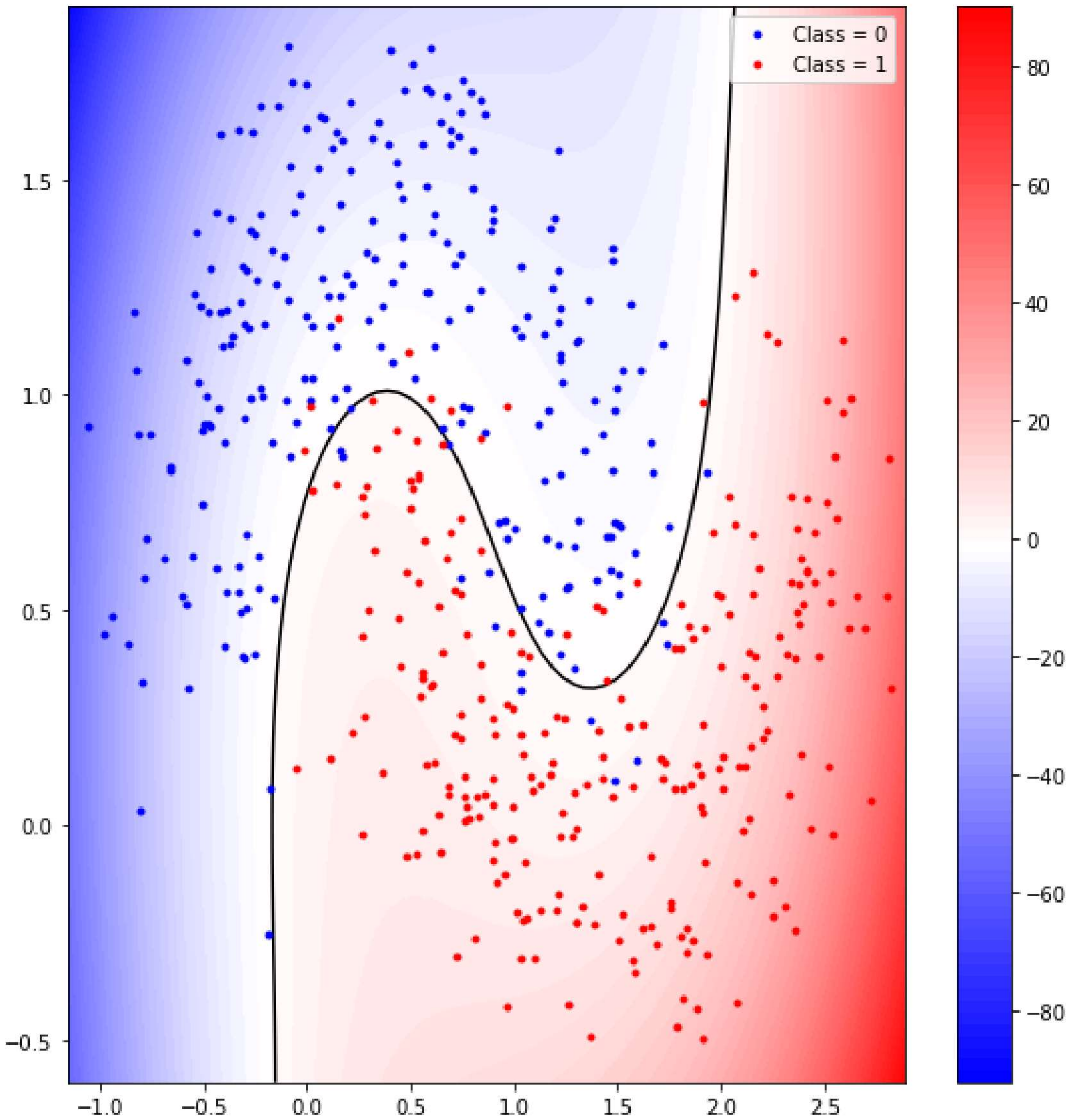
```
plot_accuracy_curve(accuracy_iteration_train, accuracy_iteration_test)
```



5. plot the classifier using the prediction values in the color coding scheme ranges from blue (class 0) to red (class 1) with the training data

In [27]:

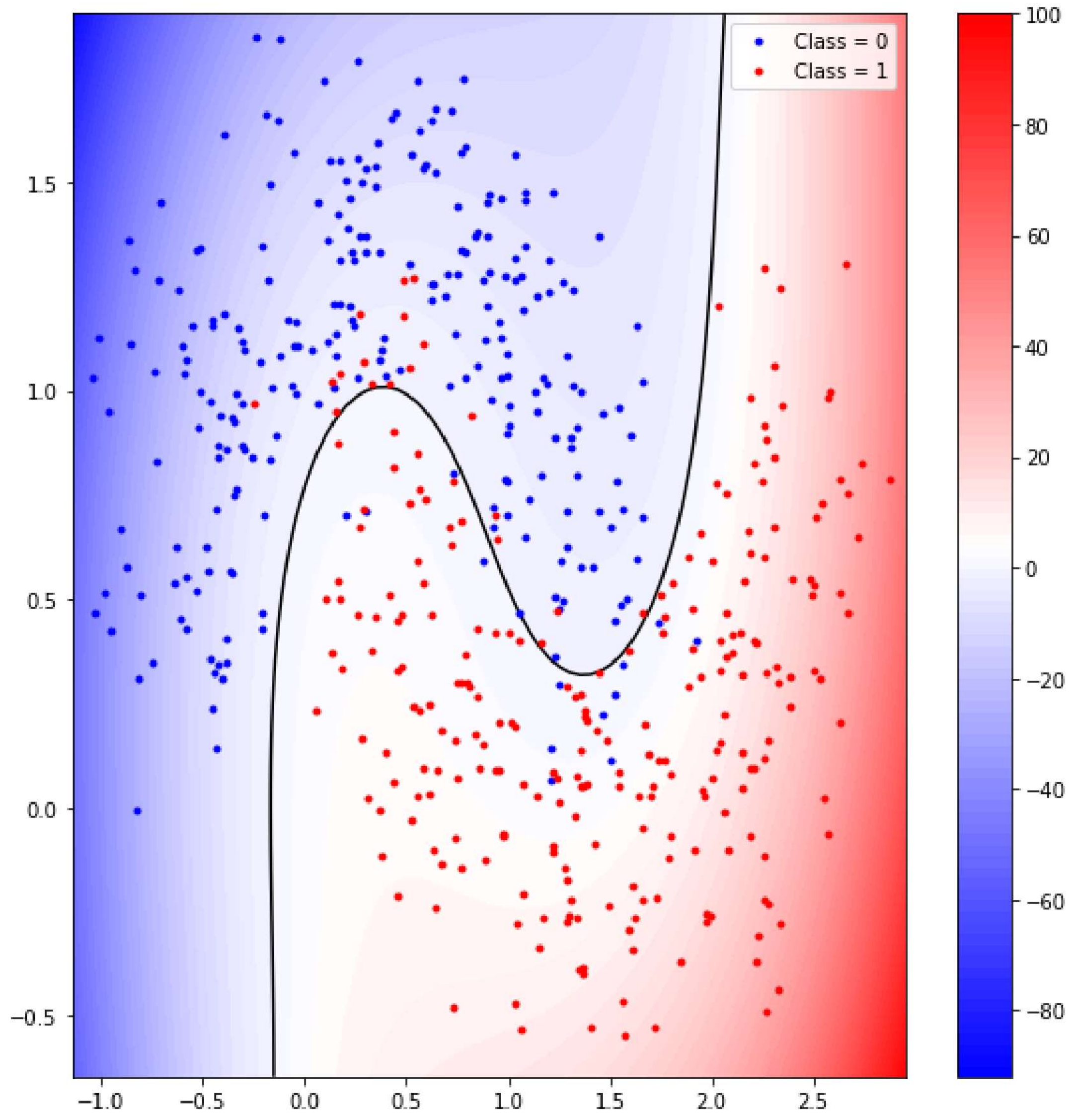
```
plot_classifier(data_train, theta_optimal)
```



6. plot the classifier using the prediction values in the color coding scheme ranges from blue (class 0) to red (class 1) with the testing data

In [28]:

```
plot_classifier(data_test, theta_optimal)
```



7. print out the final training accuracy and the final testing accuracy in number with 5 decimal places (e.g. 0.98765)

In [35]:

```
print('accuract(train): %5.5f' % (accuracy_train))
print('accuracy(test) : %5.5f' % (accuracy_test))
```

```
accuract(train): 0.94400
accuracy(test) : 0.93400
```

In [ ]: