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_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=',')

number_data_train   = data_train.shape[0]
number_data_test    = data_test.shape[0]

data_train_point    = data_train[:, 0:2]
data_train_point_x  = data_train_point[:, 0]
data_train_point_y  = data_train_point[:, 1]
data_train_label    = data_train[:, 2]

data_test_point     = data_test[:, 0:2]
data_test_point_x   = data_test_point[:, 0]
data_test_point_y   = data_test_point[:, 1]
data_test_label     = data_test[:, 2]

data_train_label_class_0    = (data_train_label == 0)
data_train_label_class_1    = (data_train_label == 1)

data_test_label_class_0     = (data_test_label == 0)
data_test_label_class_1     = (data_test_label == 1)

data_train_point_x_class_0  = data_train_point_x[data_train_label_class_0]
data_train_point_y_class_0  = data_train_point_y[data_train_label_class_0]

data_train_point_x_class_1  = data_train_point_x[data_train_label_class_1]
data_train_point_y_class_1  = data_train_point_y[data_train_label_class_1]

data_test_point_x_class_0   = data_test_point_x[data_test_label_class_0]
data_test_point_y_class_0   = data_test_point_y[data_test_label_class_0]

data_test_point_x_class_1   = data_test_point_x[data_test_label_class_1]
data_test_point_y_class_1   = data_test_point_y[data_test_label_class_1]

print('shape of point in train data = ', data_train_point.shape)
print('shape of point in test data = ', data_train_point.shape)

print('shape of label in train data = ', data_test_label.shape)
print('shape of label in test data = ', data_test_label.shape)

print('data type of point x in train data = ', data_train_point_x.dtype)
print('data type of point y in train data = ', data_train_point_y.dtype)
```

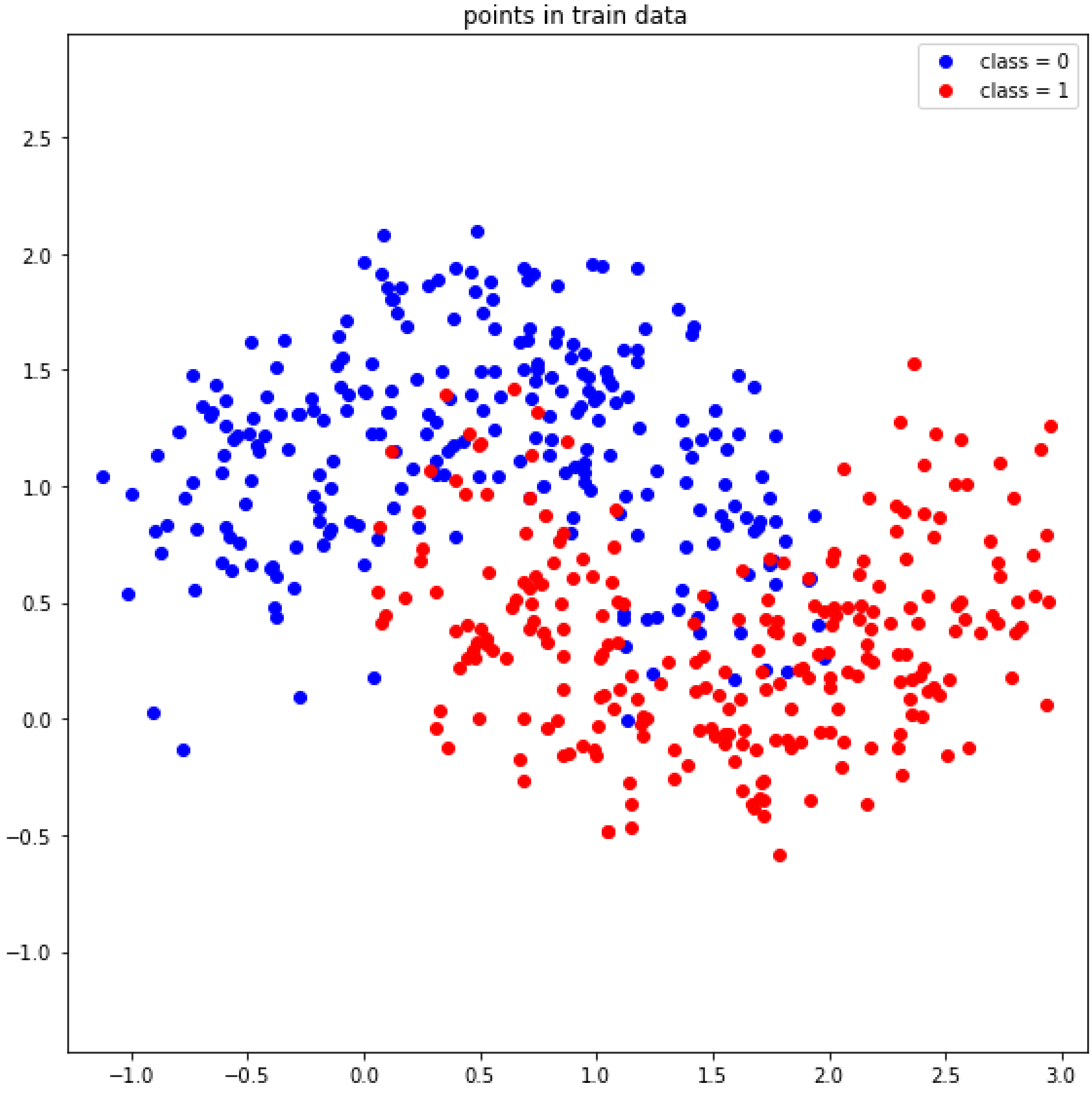
```
print('data type of point x in test data = ', data_test_point_x.dtype)
print('data type of point y in test data = ', data_test_point_y.dtype)
```

```
shape of point in train data =  (500, 2)
shape of point in test data =  (500, 2)
shape of label in train data =  (500,)
shape of label in test data =  (500,)
data type of point x in train data =  float64
data type of point y in train data =  float64
data type of point x in test data =  float64
data type of point y in test data =  float64
```

plot the data

In [ ]:

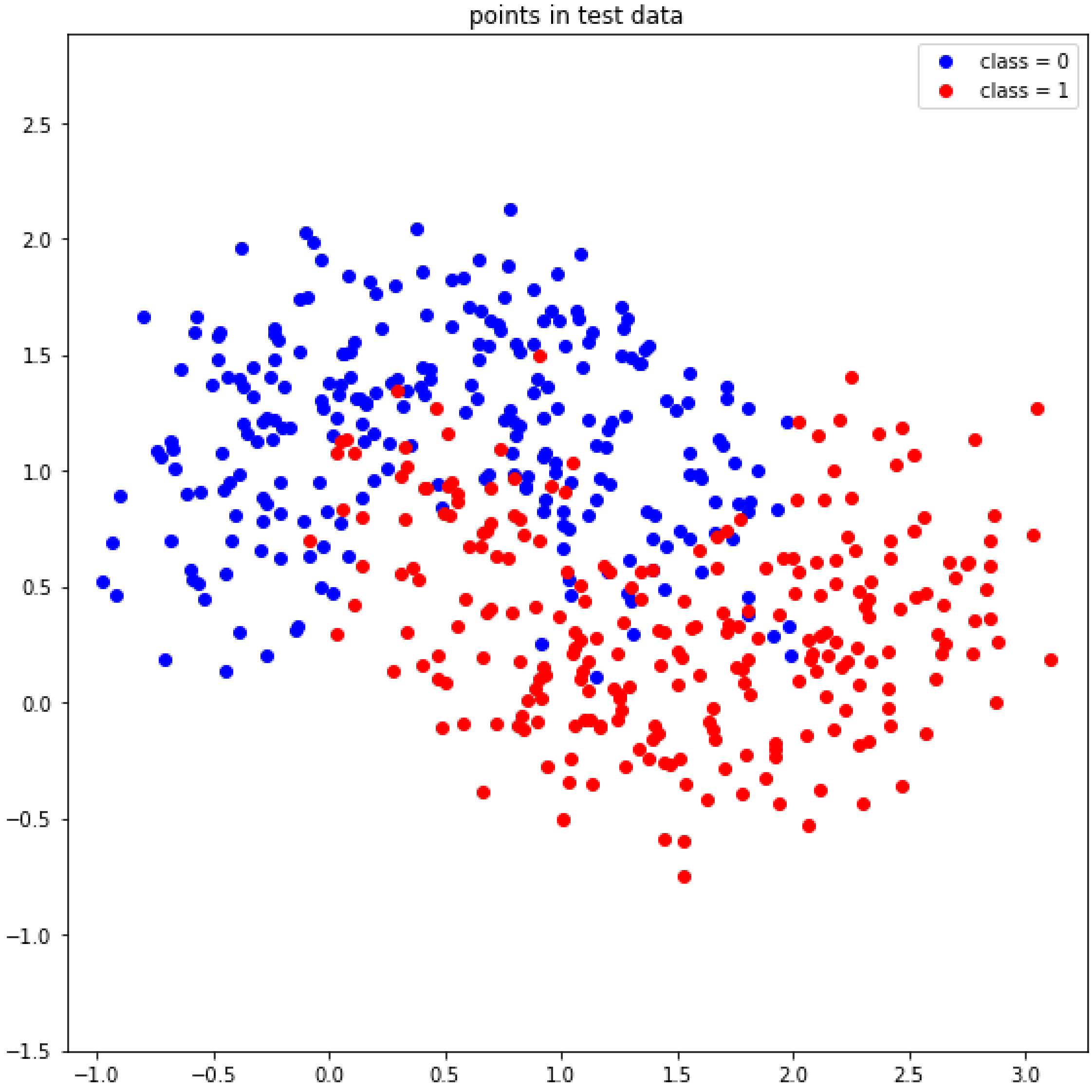
```
f = plt.figure(figsize=(8,8))

plt.title('points in train data')
plt.plot(data_train_point_x_class_0, data_train_point_y_class_0, 'o', color='blue', la
plt.plot(data_train_point_x_class_1, data_train_point_y_class_1, 'o', color='red', lab
plt.axis('equal')
plt.legend()
plt.tight_layout()
plt.show()
```



In [ ]:

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

```
plt.title('points in test data')
plt.plot(data_test_point_x_class_0, data_test_point_y_class_0, 'o', color='blue', labe
plt.plot(data_test_point_x_class_1, data_test_point_y_class_1, 'o', color='red', labe
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_feature(point):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    feature = np.zeros((point[:,0].size,8))
    feature[:,7] = point[:,0]**6
    feature[:,6] = point[:,0]**5
    feature[:,5] = point[:,0]**4
    feature[:,4] = point[:,0]**3
    feature[:,3] = point[:,0]**2
    feature[:,2] = point[:,0]
```

```
    feature[:,1] = point[:,1]
    feature[:,0] = np.ones(point[:,0].size)

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

    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 = feature.dot(theta.T)
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    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

- $\theta = (\theta_0, \theta_1, \cdots, \theta_{k-1}) \in \mathbb{R}^k$
- 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
    #
    f = compute_linear_regression(theta, feature)
    value = sigmoid(f)
```

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

    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 = (-1)*label*np.log1p(h) - (1-label)*np.log1p(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, alpha):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    temp = np.square(theta)
    regularization = np.sum(temp)*alpha
    regularization /= 2
    loss = np.sum(compute_residual(theta, feature, label))/len(label) + regularizatio

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

    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, alpha):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    # gradient = np.zeros(theta.size)

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

    gradient = (h.dot(feature))/len(label) + alpha*theta

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

    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
    #
    h = np.array(compute_logistic_regression(theta, feature))
    accuracy = 0
    for i in range(len(label)):
        if h[i]>=0.5:
            h[i] =1
        else:
            h[i] =0
    for i in range(len(label)):
        if h[i] == label[i]:
            accuracy = accuracy + 1
    accuracy = accuracy/len(label)



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

    return accuracy
```

initialize the gradient descent algorithm

In [ ]:
```
number_iteration    = 750000 # you can change this value as you want
learning_rate       = 0.035 # you can change this value as you want
number_feature      = 8 # you can change this value as you want
alpha               = 0.00001 # you can change this value as you want

theta                       = np.zeros(number_feature)
loss_iteration_train        = np.zeros(number_iteration)
loss_iteration_test         = np.zeros(number_iteration)
accuracy_iteration_train    = np.zeros(number_iteration)
accuracy_iteration_test     = np.zeros(number_iteration)
```

run the gradient descent algorithm to optimize the

loss function with respect to the model parameter

In [ ]:

```
feature_train = compute_feature(data_train_point)
feature_test = compute_feature(data_test_point)
for i in range(number_iteration):

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

    theta           = theta - learning_rate * compute_gradient(theta, feature_train, 
    loss_train      = compute_loss(theta, feature_train, data_train_label, alpha)
    loss_test       = compute_loss(theta, feature_test, data_test_label, alpha)
    accuracy_train  = compute_accuracy(theta, feature_train, data_train_label)
    accuracy_test   = compute_accuracy(theta, feature_test, data_test_label)

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

    loss_iteration_train[i]     = loss_train
    loss_iteration_test[i]      = loss_test
    accuracy_iteration_train[i] = accuracy_train
    accuracy_iteration_test[i]  = accuracy_test

theta_optimal = theta
```

---

functions for presenting the results

In [ ]:

```
def function_result_01():

    print("final training accuracy = {:13.10f}".format(accuracy_iteration_train[-1]))
```

In [ ]:

```
def function_result_02():

    print("final testing accuracy = {:13.10f}".format(accuracy_iteration_test[-1]))
```

In [ ]:

```
def function_result_03():

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

    plt.plot(loss_iteration_train, '-', color='red')
    plt.xlabel('iteration')
    plt.ylabel('loss')

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

In [ ]:

```
def function_result_04():

    plt.figure(figsize=(8,6))
```

```
    plt.title('testing loss')

    plt.plot(loss_iteration_test, '-', color='red')
    plt.xlabel('iteration')
    plt.ylabel('loss')

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

In [ ]:

```
def function_result_05():

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

    plt.plot(accuracy_iteration_train, '-', color='red')
    plt.xlabel('iteration')
    plt.ylabel('accuracy')

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

In [ ]:

```
def function_result_06():

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

    plt.plot(accuracy_iteration_test, '-', 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_07():

    plt.figure(figsize=(8,8))
    plt.title('linear regression values on the training data')

    min_x   = np.min(data_train_point_x)
    max_x   = np.max(data_train_point_x)
    min_y   = np.min(data_train_point_y)
    max_y   = np.max(data_train_point_y)

    X = np.arange(min_x - 0.5, max_x + 0.5, 0.1)
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1)

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

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

    X_Flatten = np.matrix.flatten(XX)
    Y_Flatten = np.matrix.flatten(YY)
```

```
    point_data = np.stack([X_Flatten, Y_Flatten], 1)

    feature_train_data = compute_feature(point_data)

    linear_regression_train = compute_linear_regression(theta, feature_train_data)
    reg_train = np.reshape(linear_regression_train, XX.shape)

    plt.contourf(XX, YY, reg_train, levels = 100, cmap='RdBu_r')
    plt.colorbar()

    plt.contour(XX, YY, reg_train, levels=0, colors='black')

    # plt.plot(data_train_point_x_class_0, data_train_point_y_class_0, '.', color='blu
    # plt.plot(data_train_point_x_class_1, data_train_point_y_class_1, '.', color='rec
    #
    # +++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.plot(data_train_point_x_class_0, data_train_point_y_class_0, '.', color='blue
    plt.plot(data_train_point_x_class_1, data_train_point_y_class_1, '.', color='red'

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

In [ ]:

```
def function_result_08():

    plt.figure(figsize=(8,8))
    plt.title('linear regression values on the testing data')

    min_x   = np.min(data_test_point_x)
    max_x   = np.max(data_test_point_x)
    min_y   = np.min(data_test_point_y)
    max_y   = np.max(data_test_point_y)

    X = np.arange(min_x - 0.5, max_x + 0.5, 0.1)
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1)

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

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


    X_Flatten = np.matrix.flatten(XX)
    Y_Flatten = np.matrix.flatten(YY)
    point_data = np.stack([X_Flatten, Y_Flatten], 1)

    feature_test_data = compute_feature(point_data)

    linear_regression_test = compute_linear_regression(theta, feature_test_data)
    reg_test = np.reshape(linear_regression_test, XX.shape)

    plt.contourf(XX, YY, reg_test, levels = 100, cmap='RdBu_r')
    plt.colorbar()

    plt.contour(XX, YY, reg_test, levels=0, colors='black')

    # plt.plot(data_test_point_x_class_0, data_test_point_y_class_0, '.', color='blue'
    # plt.plot(data_test_point_x_class_1, data_test_point_y_class_1, '.', color='red',
    #
    # +++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
    plt.plot(data_test_point_x_class_0, data_test_point_y_class_0, '.', color='blue',
    plt.plot(data_test_point_x_class_1, data_test_point_y_class_1, '.', color='red',

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

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

In [ ]:

```
def function_result_09():

    plt.figure(figsize=(8,8))
    plt.title('logistic regression values on the training data')

    min_x   = np.min(data_train_point_x)
    max_x   = np.max(data_train_point_x)
    min_y   = np.min(data_train_point_y)
    max_y   = np.max(data_train_point_y)

    X = np.arange(min_x - 0.5, max_x + 0.5, 0.1)
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1)

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

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    X_Flatten = np.matrix.flatten(XX)
    Y_Flatten = np.matrix.flatten(YY)
    point_data = np.stack([X_Flatten, Y_Flatten], 1)

    feature_train_data = compute_feature(point_data)

    logistic_regression_train = compute_logistic_regression(theta, feature_train_data)
    log_train = np.reshape(logistic_regression_train, XX.shape)

    plt.contourf(XX, YY, log_train, levels = 100, cmap='RdBu_r')
    plt.colorbar()

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


    plt.plot(data_train_point_x_class_0, data_train_point_y_class_0, '.', color='blue
    plt.plot(data_train_point_x_class_1, data_train_point_y_class_1, '.', color='red'

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

In [ ]:

```
def function_result_10():

    plt.figure(figsize=(8,8))
    plt.title('logistic regression values on the testing data')

    min_x   = np.min(data_test_point_x)
    max_x   = np.max(data_test_point_x)
    min_y   = np.min(data_test_point_y)
    max_y   = np.max(data_test_point_y)
```

```
    X = np.arange(min_x - 0.5, max_x + 0.5, 0.1)
    Y = np.arange(min_y - 0.5, max_y + 0.5, 0.1)

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

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    X_Flatten = np.matrix.flatten(XX)
    Y_Flatten = np.matrix.flatten(YY)
    point_data = np.stack([X_Flatten, Y_Flatten], 1)

    feature_test_data = compute_feature(point_data)

    logistic_regression_test = compute_logistic_regression(theta, feature_test_data)
    log_test = np.reshape(logistic_regression_test, XX.shape)

    plt.contourf(XX, YY, log_test, levels = 100, cmap='RdBu_r')
    plt.colorbar()



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


    plt.plot(data_test_point_x_class_0, data_test_point_y_class_0, '.', color='blue',
    plt.plot(data_test_point_x_class_1, data_test_point_y_class_1, '.', color='red',

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

---

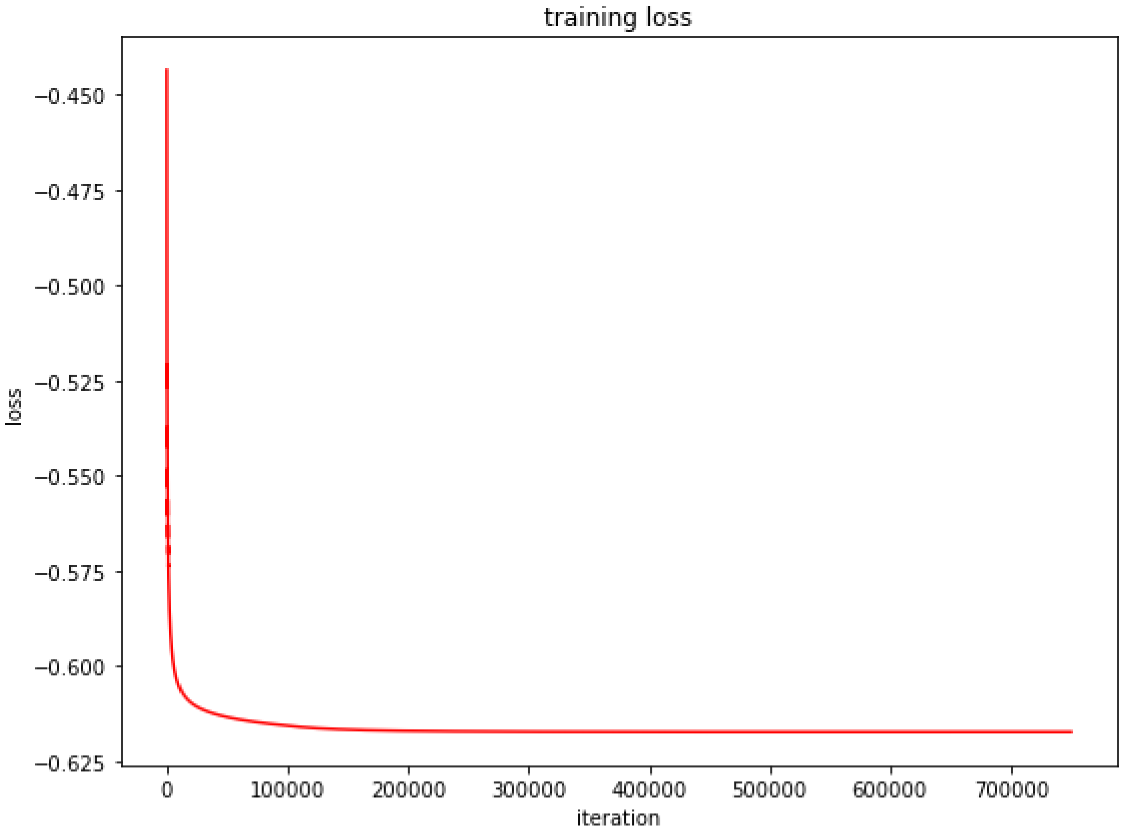
results

In [ ]:

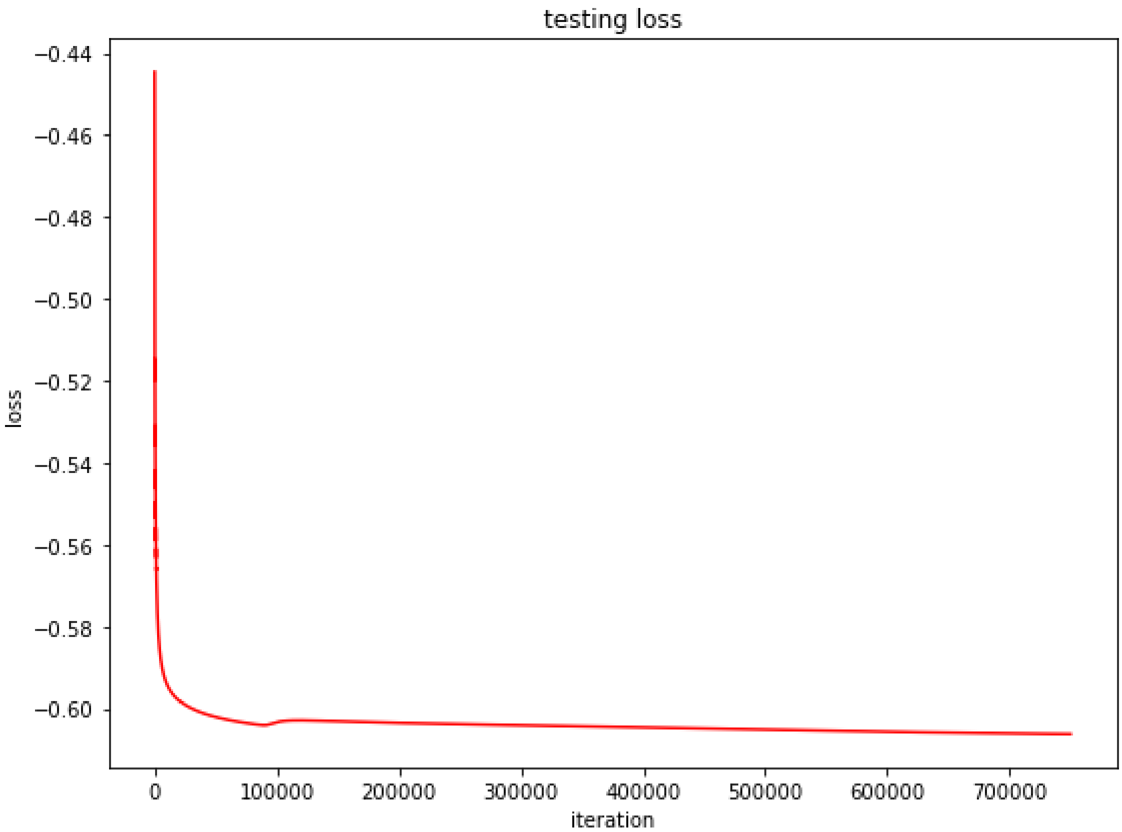
```
number_result = 10

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 training accuracy =  0.9180000000
**************************************************
## [RESULT 02]
**************************************************
final testing accuracy =  0.9000000000
**************************************************
```

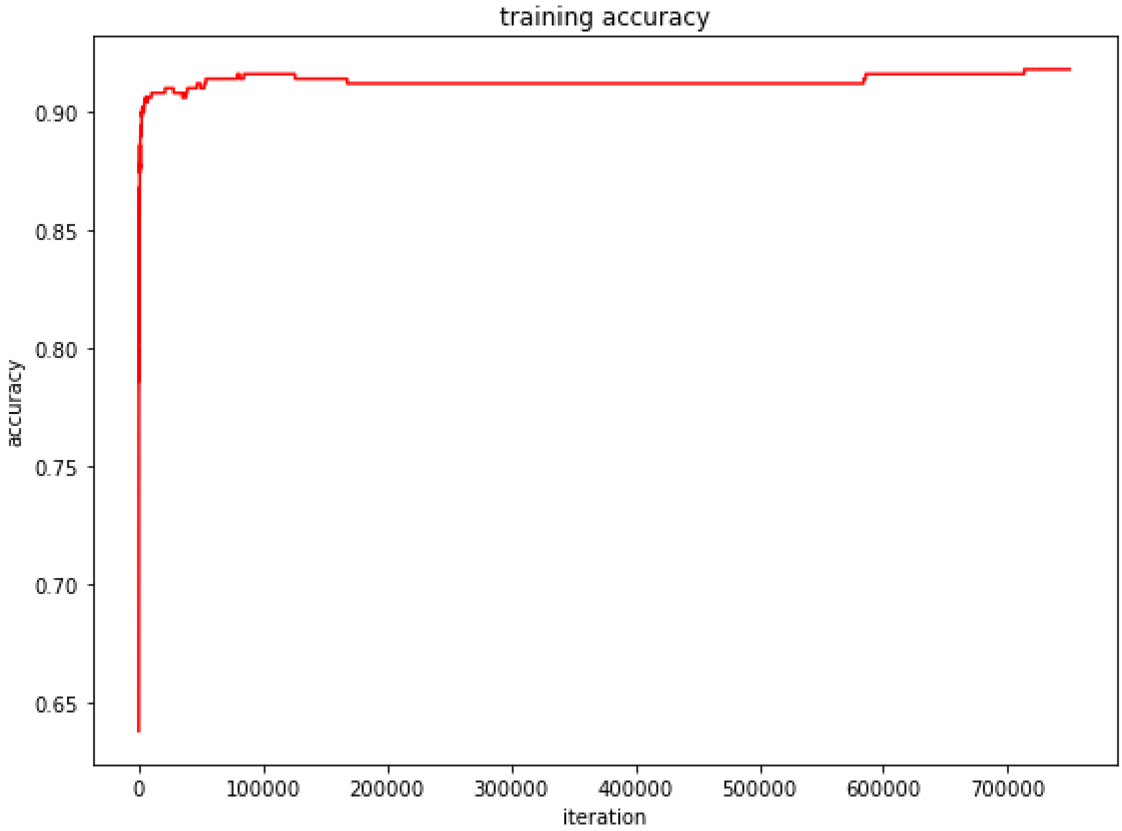
```
## [RESULT 03]
**************************************************
```



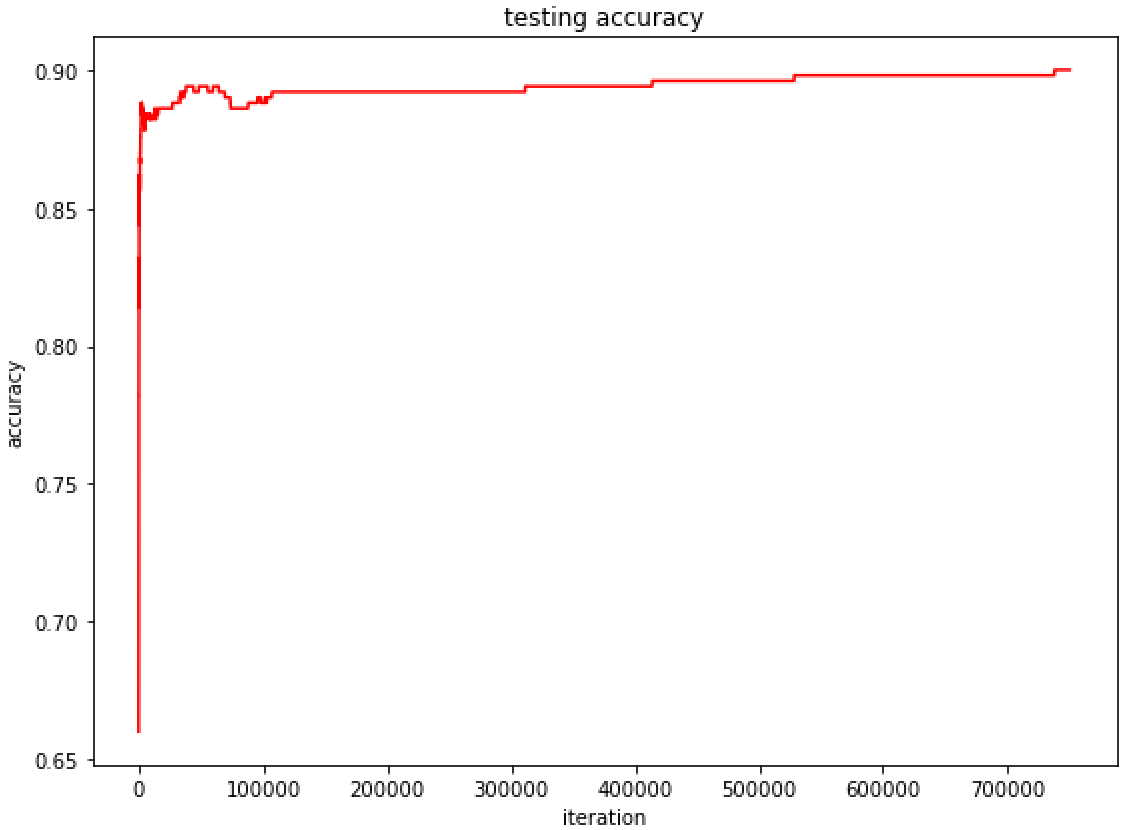

```
**************************************************
## [RESULT 04]
**************************************************
```



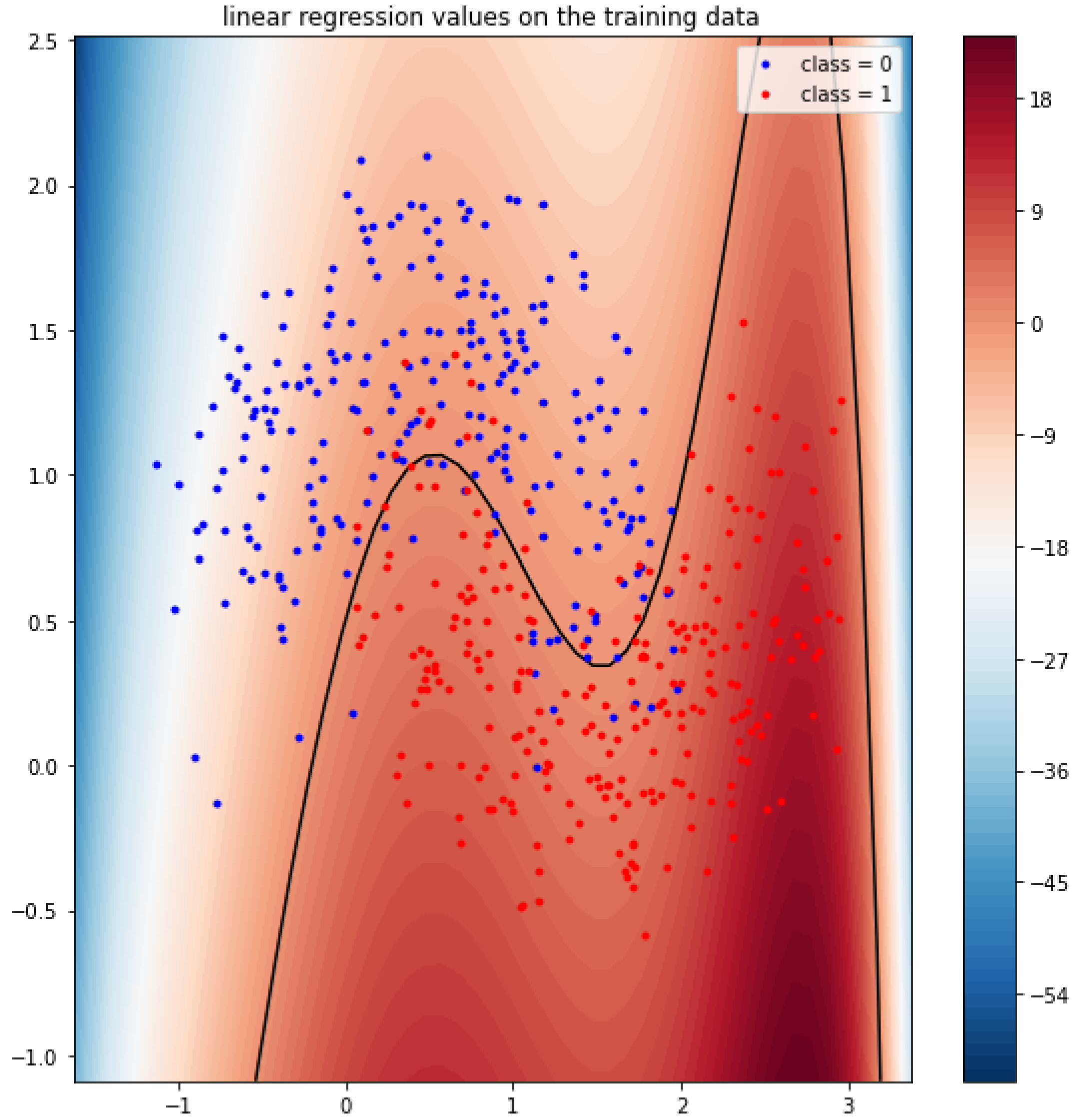

```
**************************************************
## [RESULT 05]
**************************************************
```



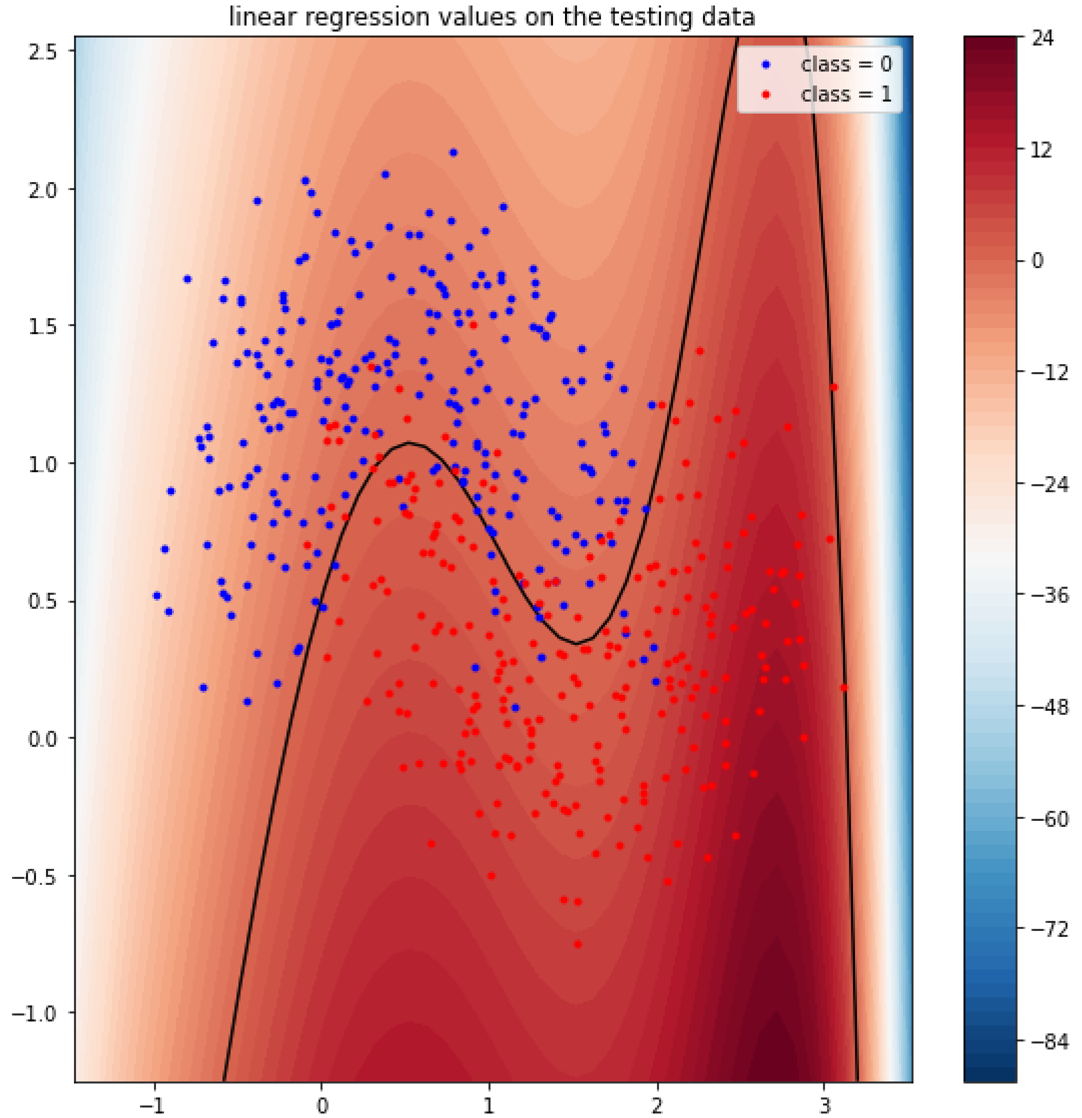

```
**************************************************
## [RESULT 06]
**************************************************
```



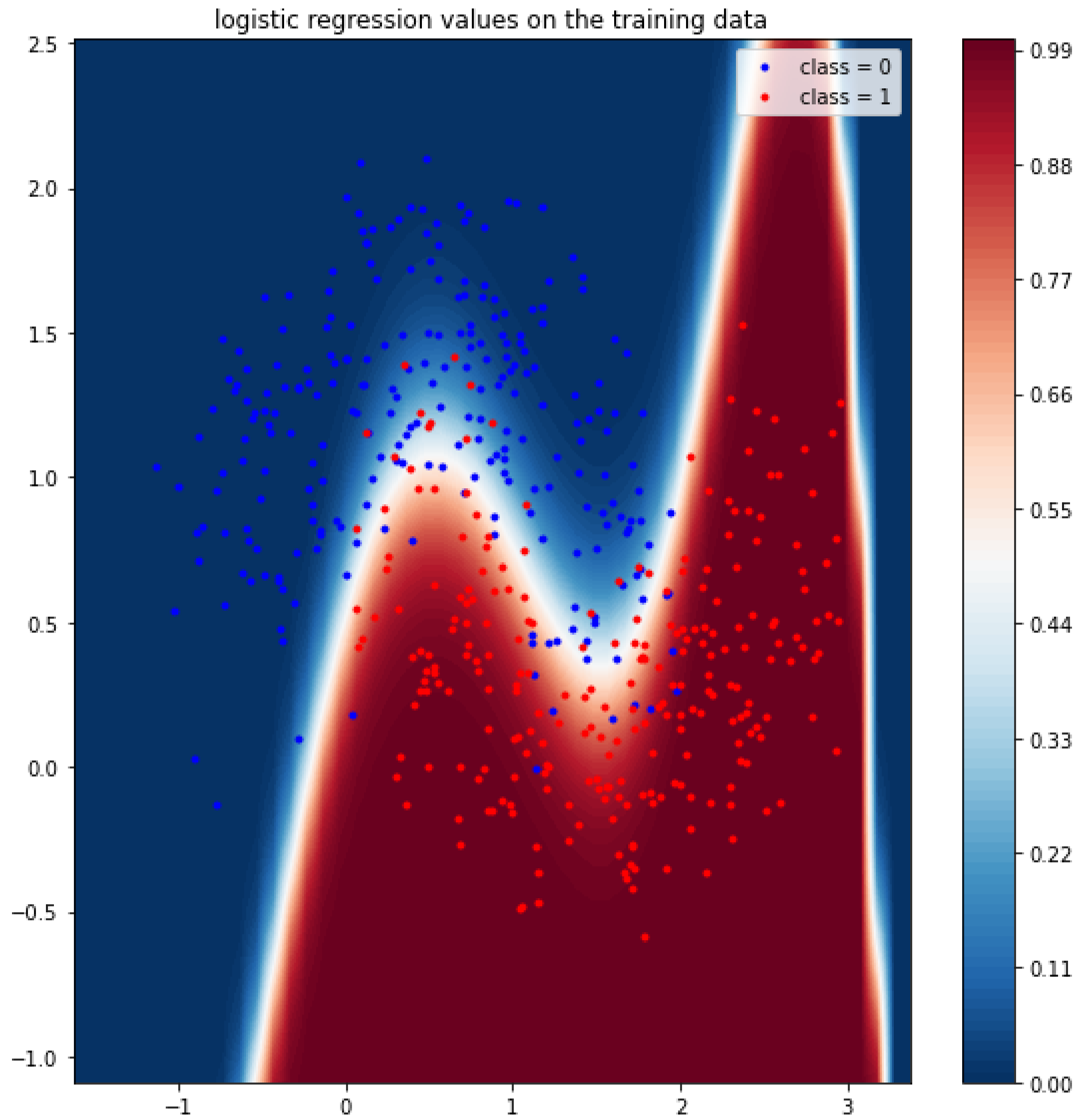

```
**************************************************
## [RESULT 07]
**************************************************
```


linear regression values on the training data
class = 0
class = 1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2
3
18
9
0
−9
−18
−27
−36
−45
−54

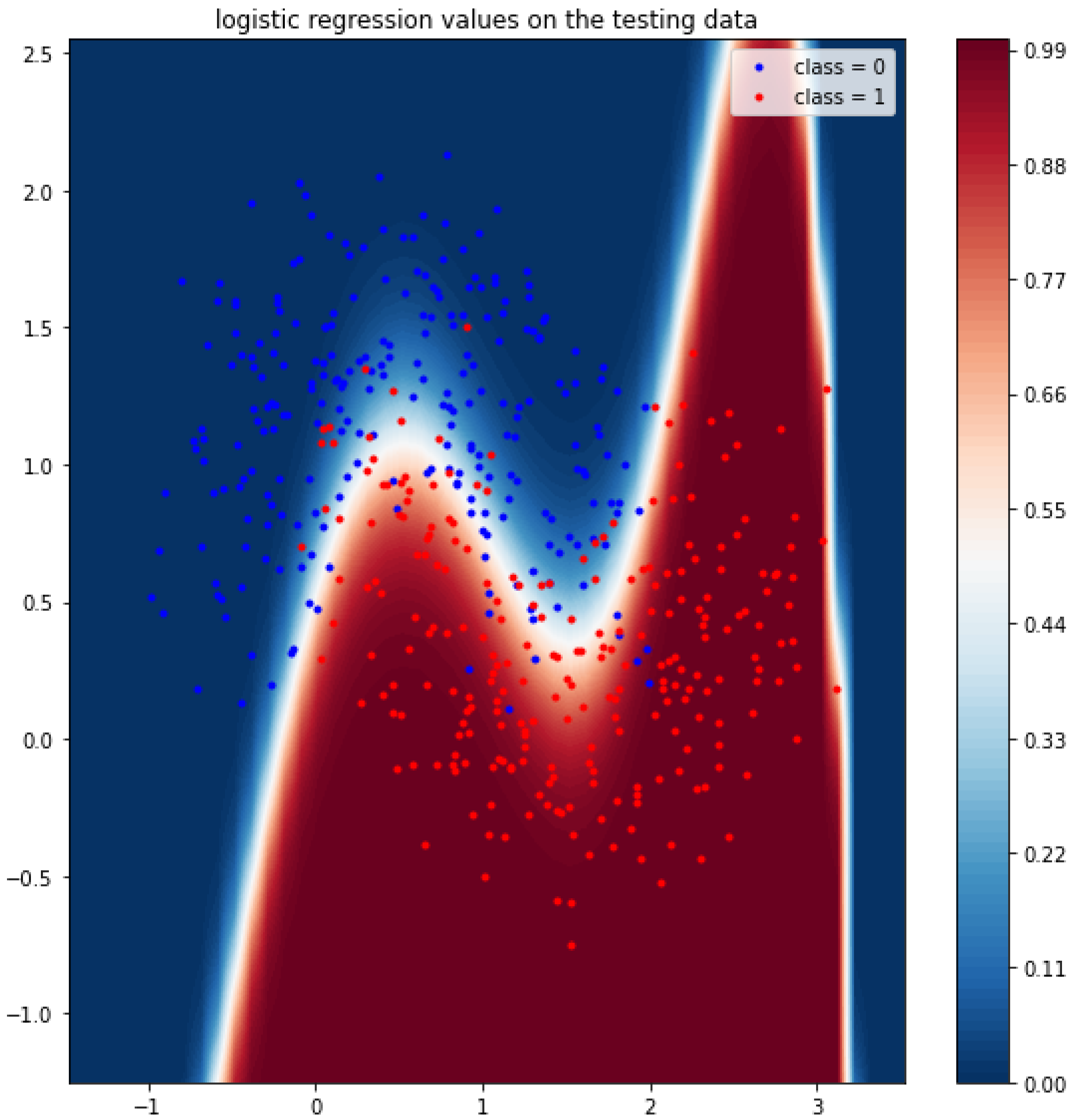
```
**************************************************
## [RESULT 08]
**************************************************
```


linear regression values on the testing data
class = 0
class = 1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2
3
24
12
0
−12
−24
−36
−48
−60
−72
−84

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


logistic regression values on the training data
class = 0
class = 1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2
3
0.99
0.88
0.77
0.66
0.55
0.44
0.33
0.22
0.11
0.00

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


logistic regression values on the testing data
class = 0
class = 1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2
3
0.99
0.88
0.77
0.66
0.55
0.44
0.33
0.22
0.11
0.00

```
In [ ]:
```