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

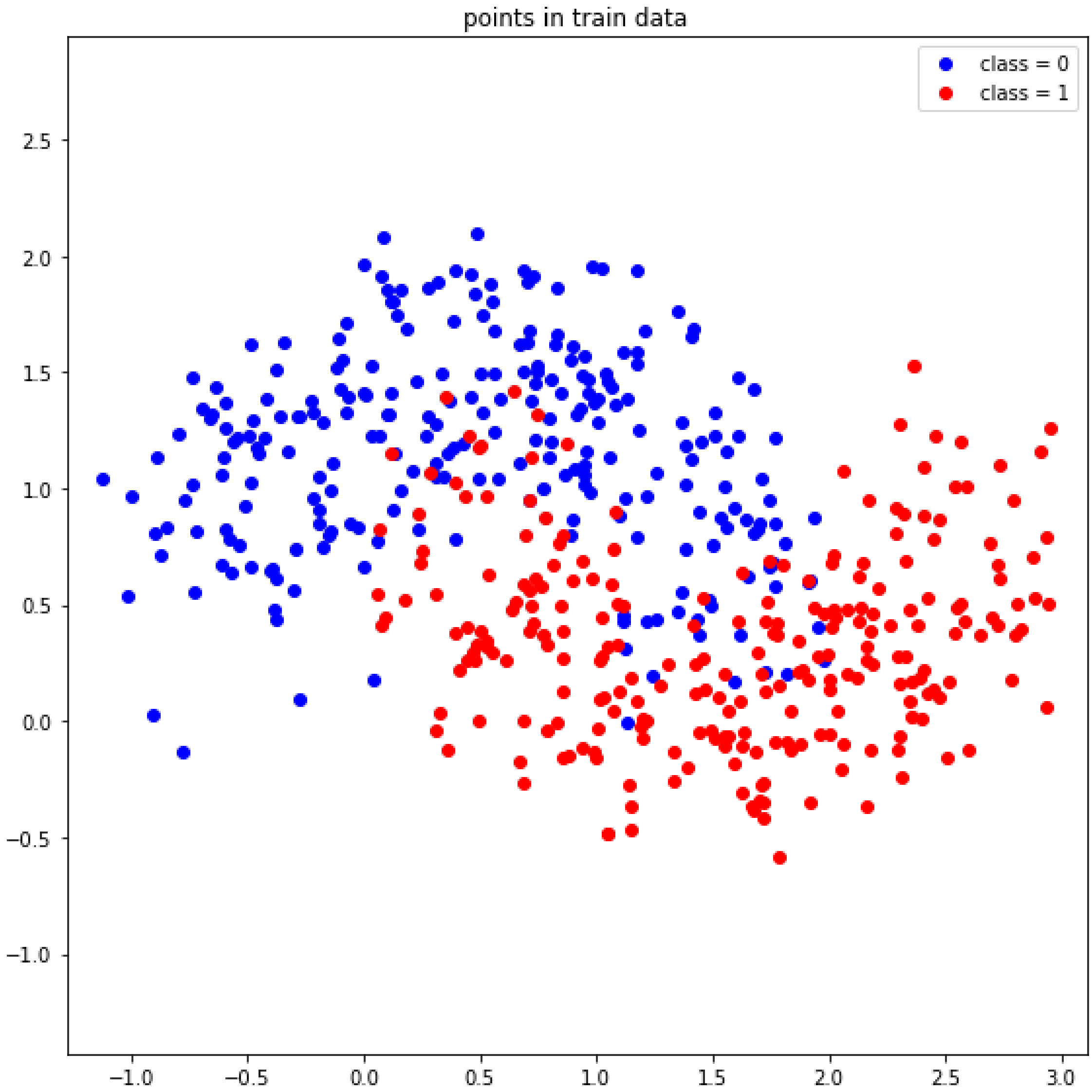
In [ ]:
```
fname_data_train    = 'assignment_10_data_train.csv'
fname_data_test     = 'assignment_10_data_test.csv'

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 [ ]:

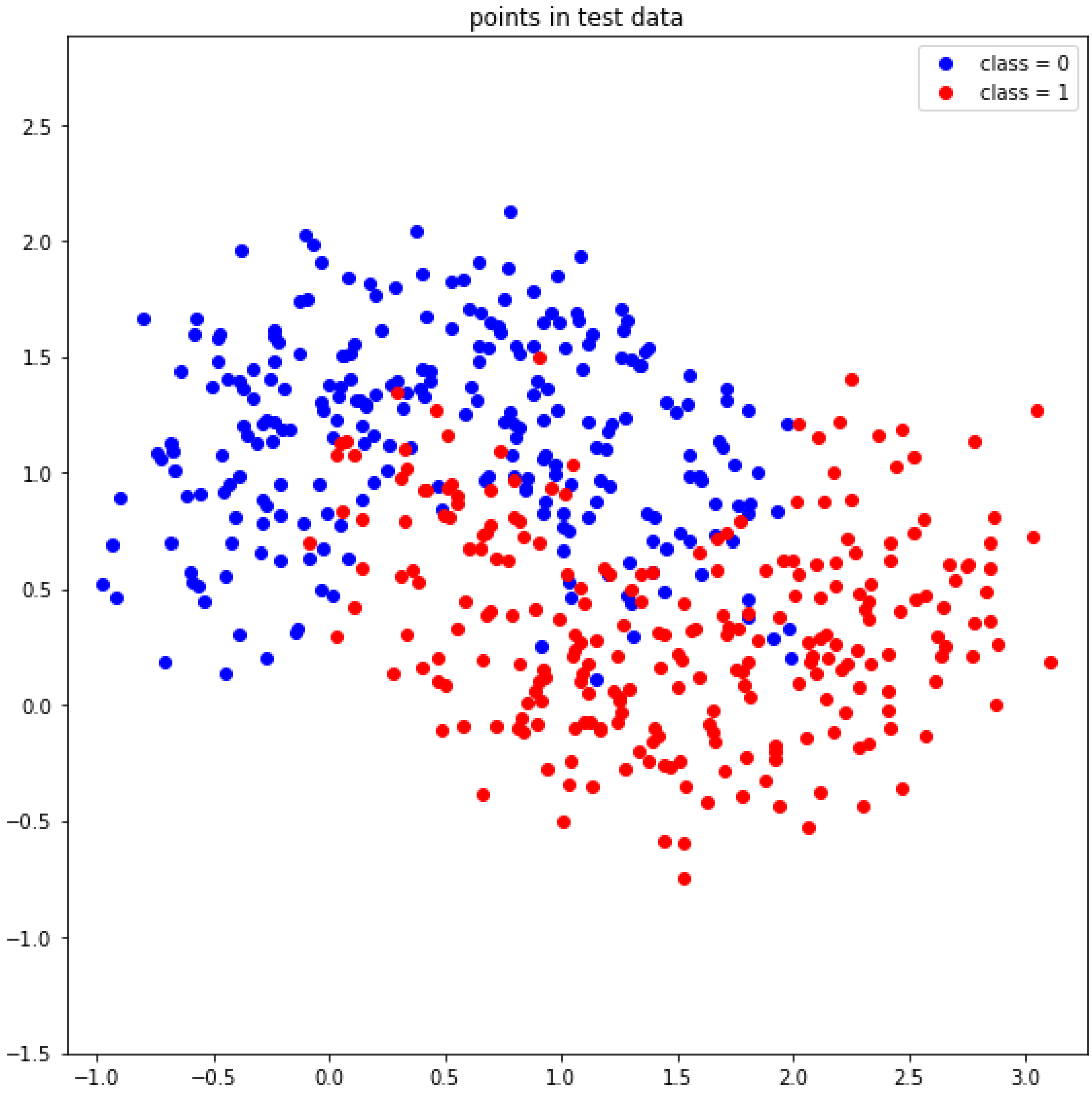
```
f = plt.figure(figsize=(8,8))

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
    #

    n = point.shape[0]
    x = point[:,0].reshape(n,1)
    y = point[:,1].reshape(n,1)



    k1 = x**3
    k2 = x**2
    k3 = x
    k4 = y
```

```python
    feature = np.concatenate((np.ones((n,1)),k1,k2,k3,k4),axis=1)

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

    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 [ ]:
```python
def compute_linear_regression(theta, feature):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    value = np.matmul(theta,feature.T)
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return value
```

define sigmoid function with input

- $z \in \mathbb{R}$

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

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

    value = np.divide(1,1+np.exp(-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 [ ]:
```python
def compute_logistic_regression(theta, feature):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    value = sigmoid(compute_linear_regression(theta, feature))

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

    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 = -label*np.log(h) - (1-label)*np.log(1-h)
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return residual
```

define the loss function for the logistic regression

- $\theta = (\theta_0, \theta_1, \cdots, \theta_{k-1}) \in \mathbb{R}^k$
- 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
    #
    n = len(label)
    residual = compute_residual(theta, feature, label)
    loss = np.divide(np.sum(residual),n) + (alpha/2)*np.inner(theta,theta)

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

    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
    #
    n= len(label)
    h = compute_logistic_regression(theta, feature)
    gradient = np.divide(np.matmul(feature.T,(h-label)),n)  + 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
    #
    n = feature.shape[0]
    threshold = 0.5

    h = compute_logistic_regression(theta, feature)

    prediction = np.where( h >= threshold, 1, 0)

    accuracy = np.divide(np.sum(label == prediction),n)



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

    return accuracy
```

initialize the gradient descent algorithm

In [ ]:
```
number_iteration    = 7000 # you can change this value as you want
learning_rate       = 0.5 # you can change this value as you want
number_feature      = 5 # you can change this value as you want
alpha               = 0 # 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 [ ]:
```
for i in range(number_iteration):

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

    theta           = theta - learning_rate*compute_gradient(theta, compute_feature(d
    loss_train      = compute_loss(theta, compute_feature(data_train_point), data_tra
    loss_test       = compute_loss(theta, compute_feature(data_test_point), data_test_
    accuracy_train  = compute_accuracy(theta, compute_feature(data_train_point), data_
    accuracy_test   = compute_accuracy(theta, compute_feature(data_test_point), data_t
```

```python
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    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
```

In [ ]:
```python
accuracy_iteration_train
```

Out[ ]:
```
array([0.71 , 0.726, 0.742, ..., 0.916, 0.916, 0.916])
```

In [ ]:
```python
accuracy_iteration_test
```

Out[ ]:
```
array([0.71 , 0.724, 0.762, ..., 0.9  , 0.9  , 0.9  ])
```

In [ ]:
```python
loss_iteration_train
```

Out[ ]:
```
array([0.80150965, 0.58330903, 0.49512433, ..., 0.21286384, 0.21286209,
       0.21286034])
```

In [ ]:
```python
loss_iteration_test
```

Out[ ]:
```
array([0.79376543, 0.58589701, 0.50452797, ..., 0.24151418, 0.2415129 ,
       0.24151163])
```

In [ ]:
```python
theta_optimal
```

Out[ ]:
```
array([  3.16392019,   4.292573  , -12.59913637,   8.58804276,
        -4.91267772])
```

---

functions for presenting the results

In [ ]:
```python
def function_result_01():

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

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

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

In [ ]:
```python
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
    #
    XY = np.c_[XX.ravel(), YY.ravel()]

    y_pred = compute_linear_regression(theta_optimal, compute_feature(XY))
```

```
    ZZ = y_pred.reshape(XX.shape)
    y = (-theta_optimal[0] - theta_optimal[1]*X**3 - theta_optimal[2]*X**2 - theta_o

    plt.contourf(XX, YY, ZZ,cmap="RdBu_r",levels=np.linspace(ZZ.min(),ZZ.max(),100))
    plt.colorbar()

    plt.plot(X,y,color='black')
    plt.ylim(-1,2.5)



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

    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
    #
    XY = np.c_[XX.ravel(), YY.ravel()]

    y_pred = compute_linear_regression(theta_optimal, compute_feature(XY))

    ZZ = y_pred.reshape(XX.shape)
    x = np.arange(-2,3,0.1)
    y = (-theta_optimal[0] - theta_optimal[1]*X**3 - theta_optimal[2]*X**2 - theta_o

    plt.contourf(XX, YY, ZZ,cmap="RdBu_r",levels=np.linspace(ZZ.min(),ZZ.max(),100))
    plt.colorbar()

    plt.plot(X,y,color='black')
    plt.ylim(-1,2.5)



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

    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
    #

    ZZ = np.divide(1,1+np.exp(-theta[0]-theta[1]*XX -theta[2]*XX -theta[3]*XX-theta[
    plt.contourf(XX,YY,ZZ,cmap="RdBu_r",levels = np.linspace(ZZ.min(),ZZ.max(),100))
    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
    #
```

```
    ZZ = np.divide(1,1+np.exp(-theta[0]-theta[1]*XX -theta[2]*XX -theta[3]*XX-theta[
    plt.contourf(XX,YY,ZZ,cmap="RdBu_r",levels = np.linspace(ZZ.min(),ZZ.max(),100))
    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()
```

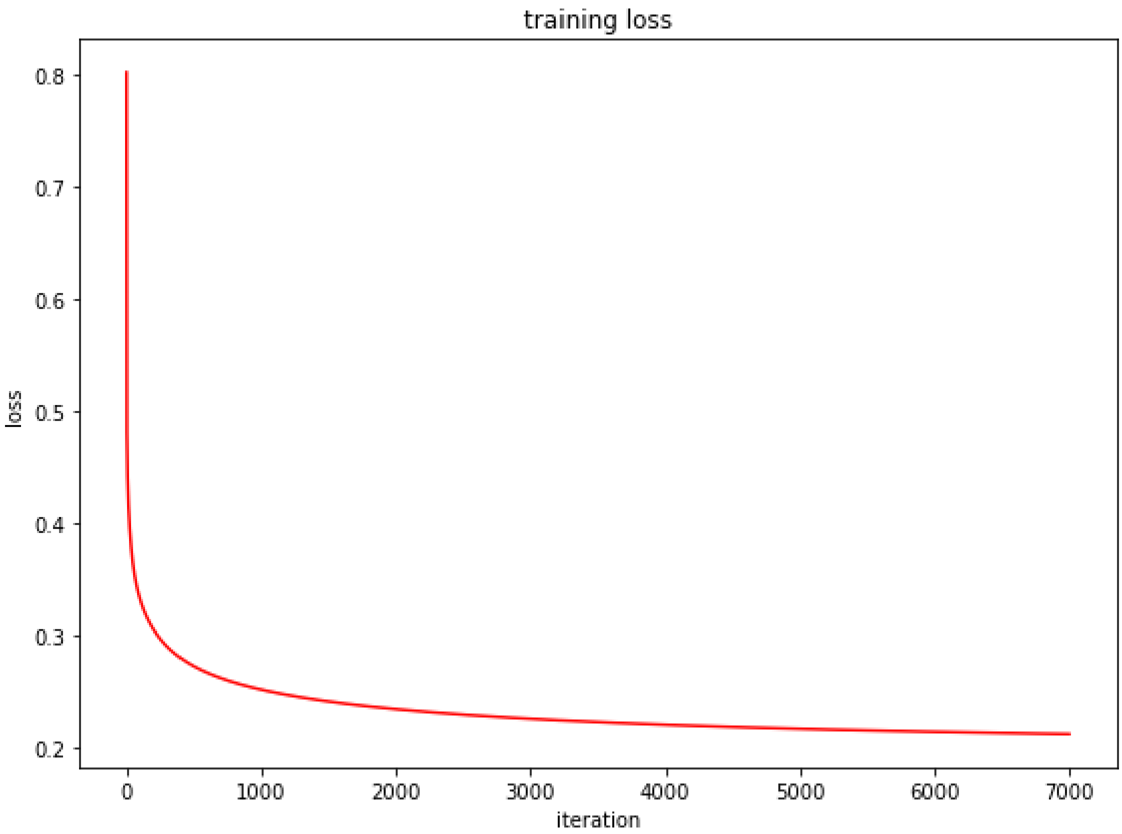
---

results

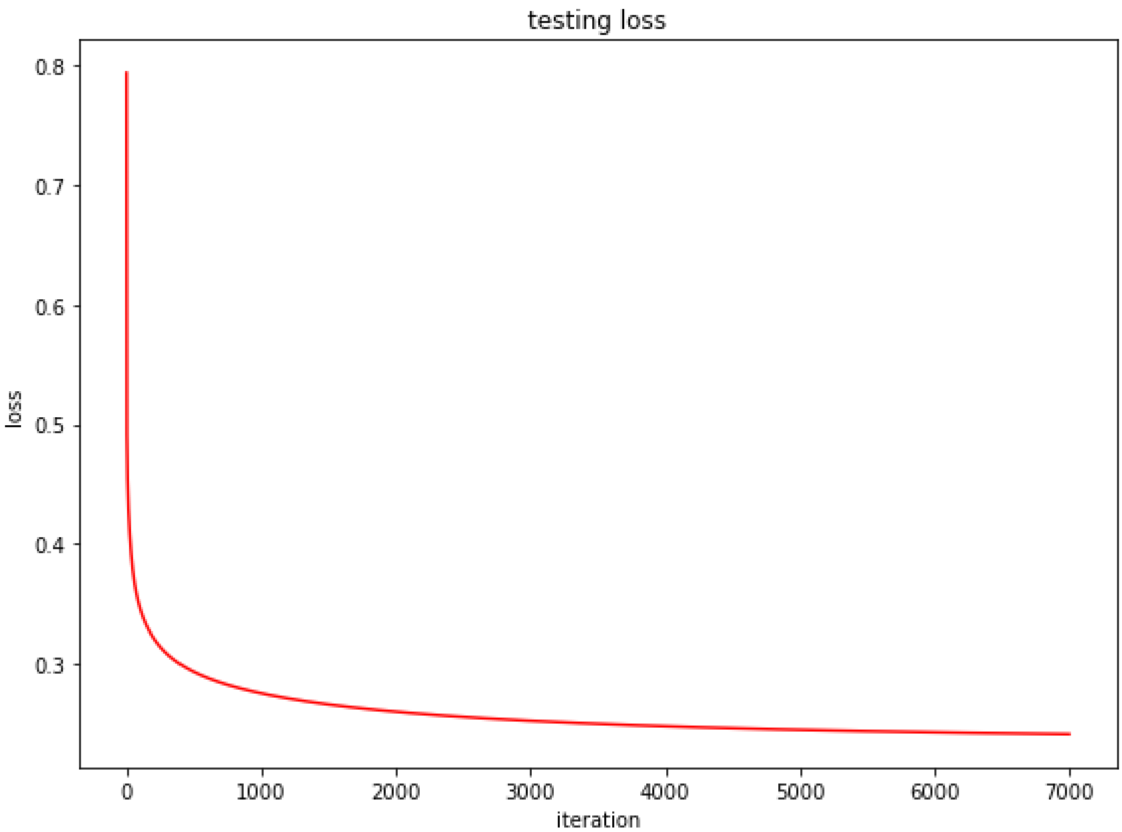
In [ ]:

```
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)
```

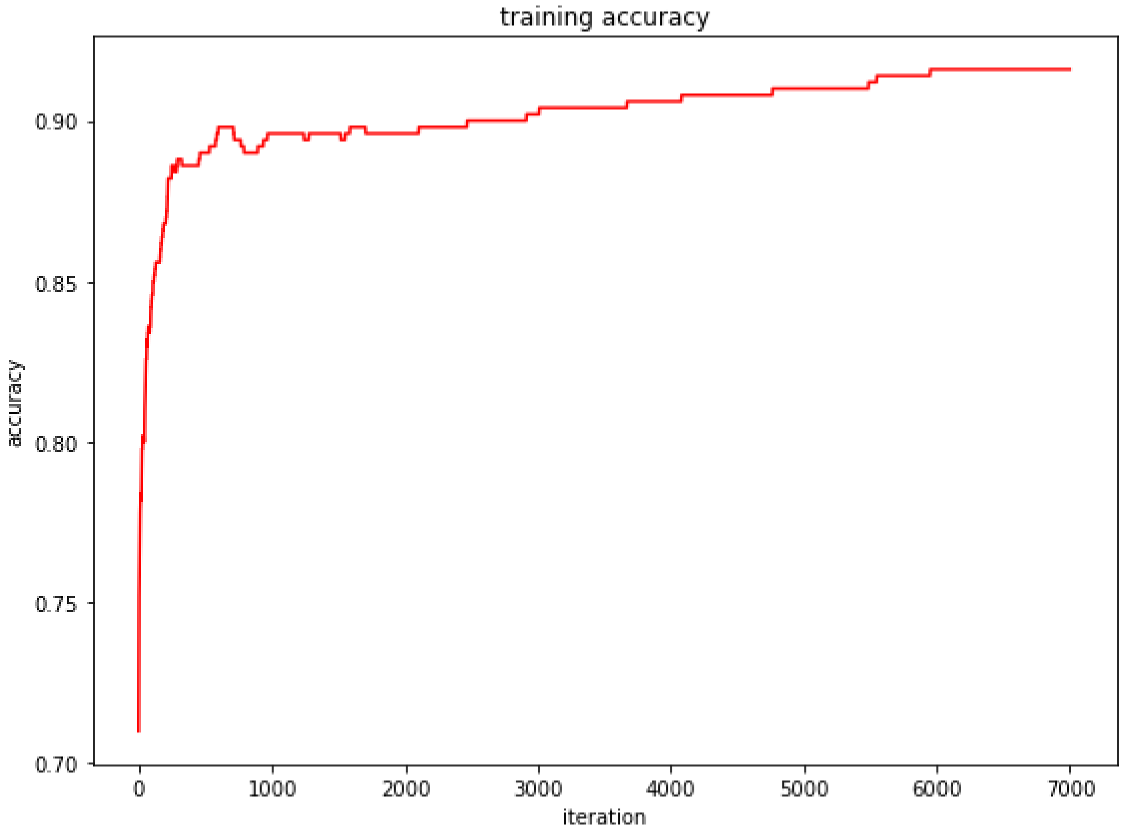
```
**************************************************
## [RESULT 01]
**************************************************
final training accuracy =  0.9160000000
**************************************************
## [RESULT 02]
**************************************************
final testing accuracy =  0.9000000000
**************************************************
## [RESULT 03]
**************************************************
```



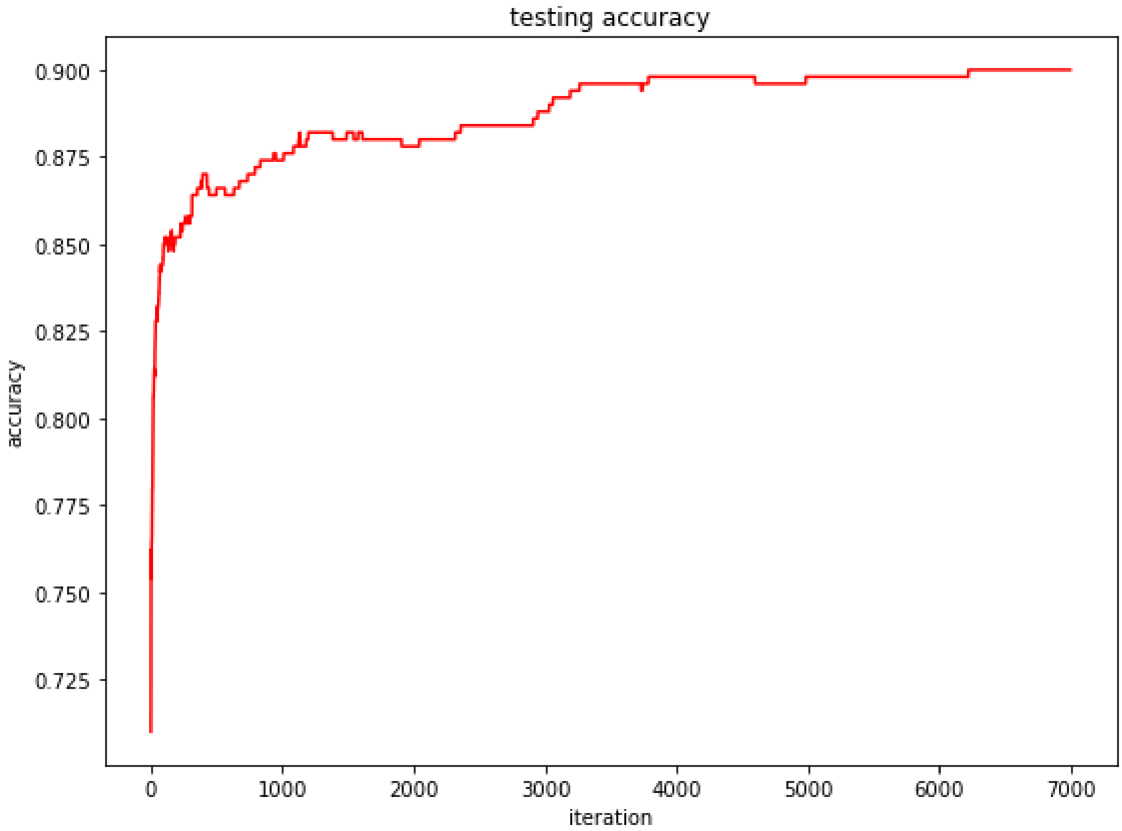
****
[RESULT 04]
****



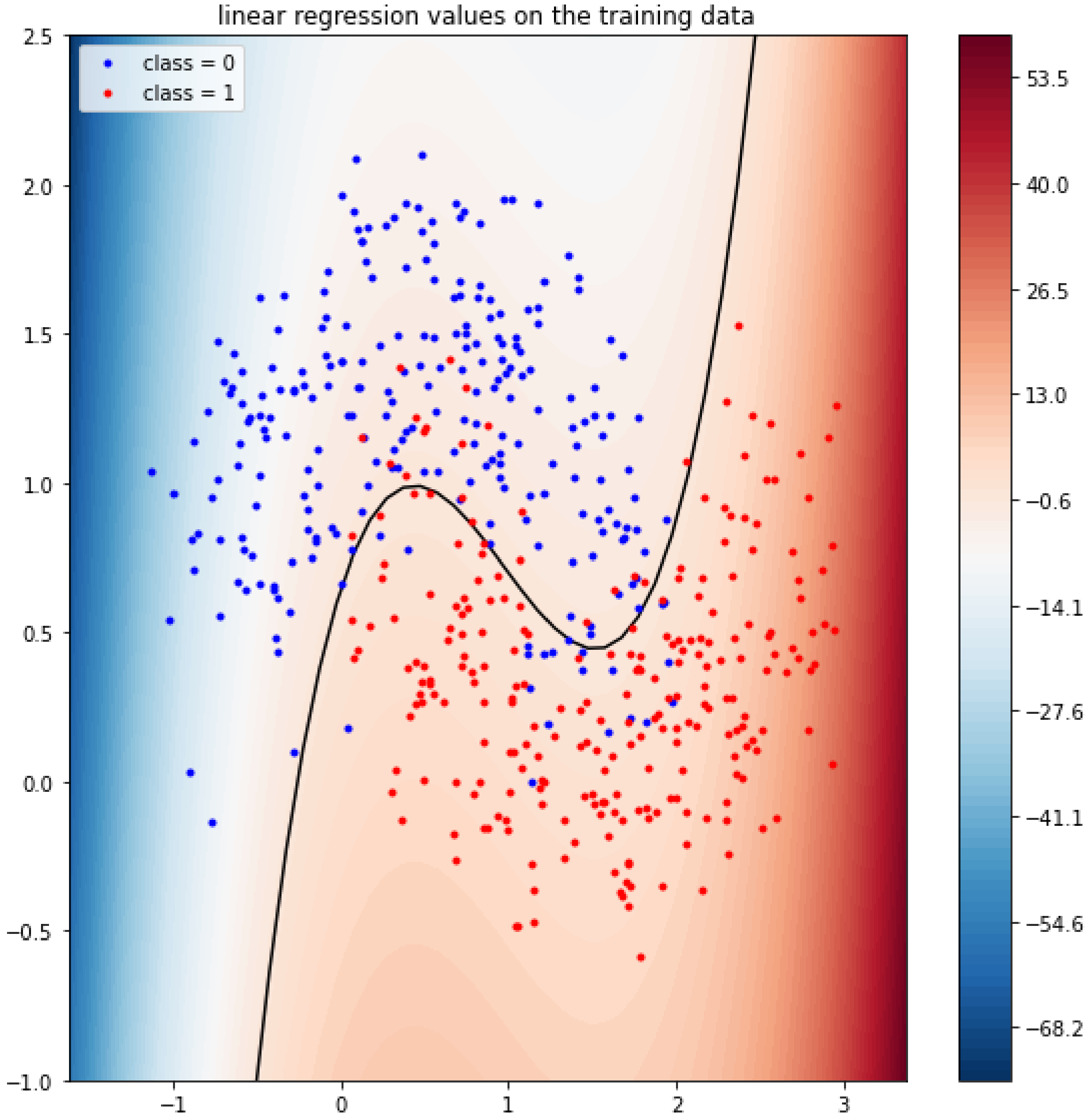
****
[RESULT 05]
****



****
[RESULT 06]
****



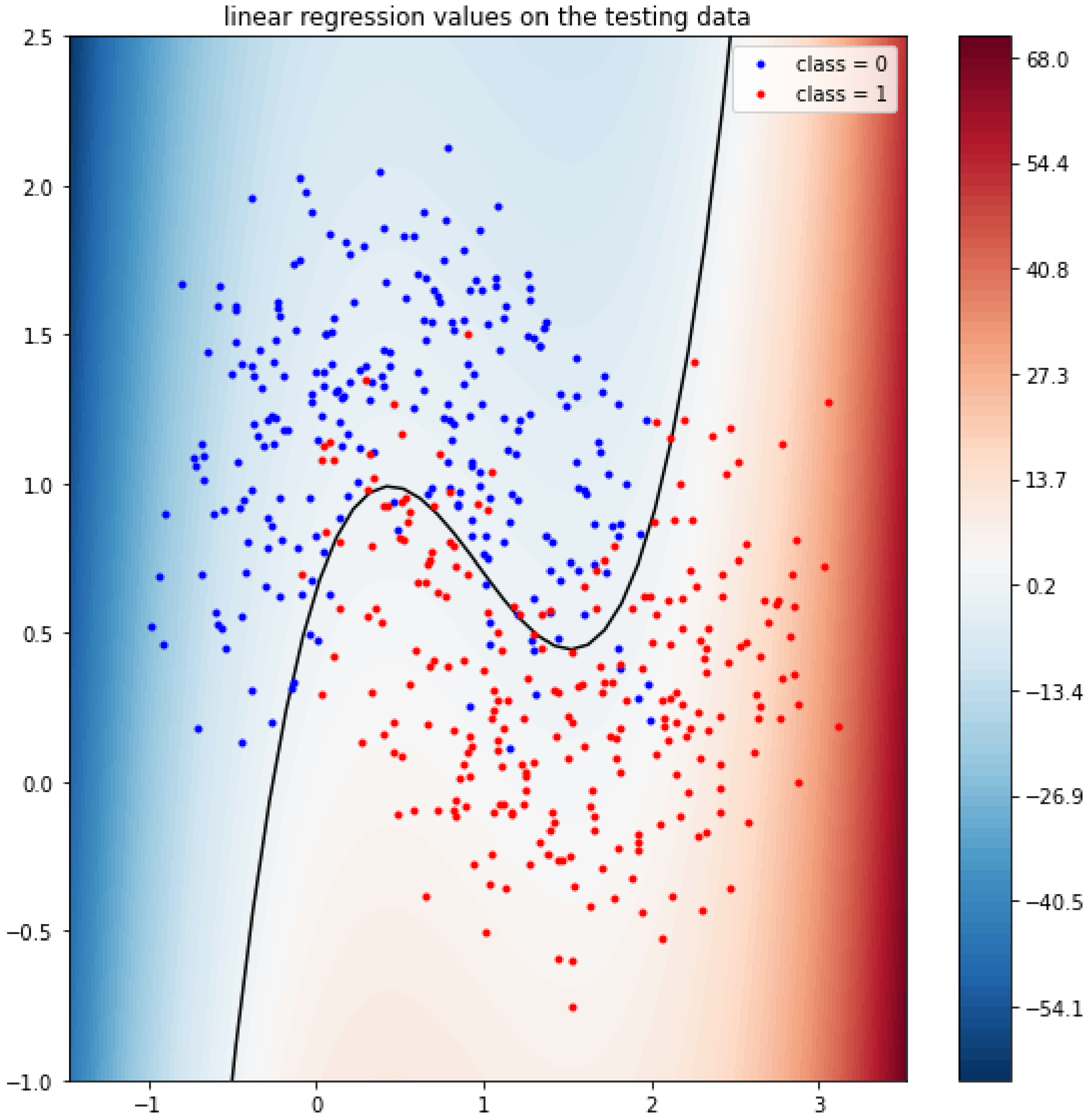
****
[RESULT 07]
****


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
53.5
40.0
26.5
13.0
−0.6
−14.1
−27.6
−41.1
−54.6
−68.2

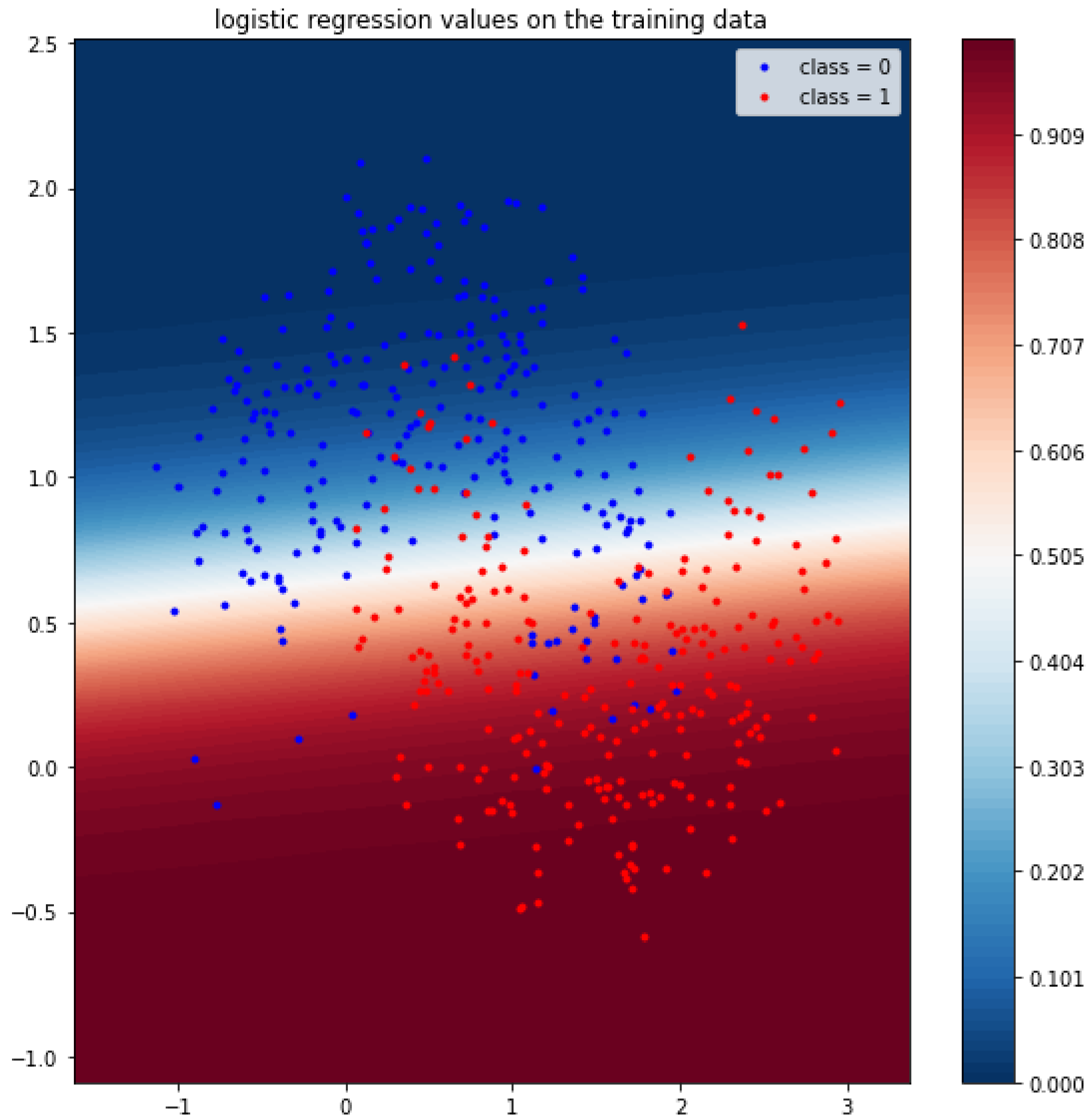
****

[RESULT 08]

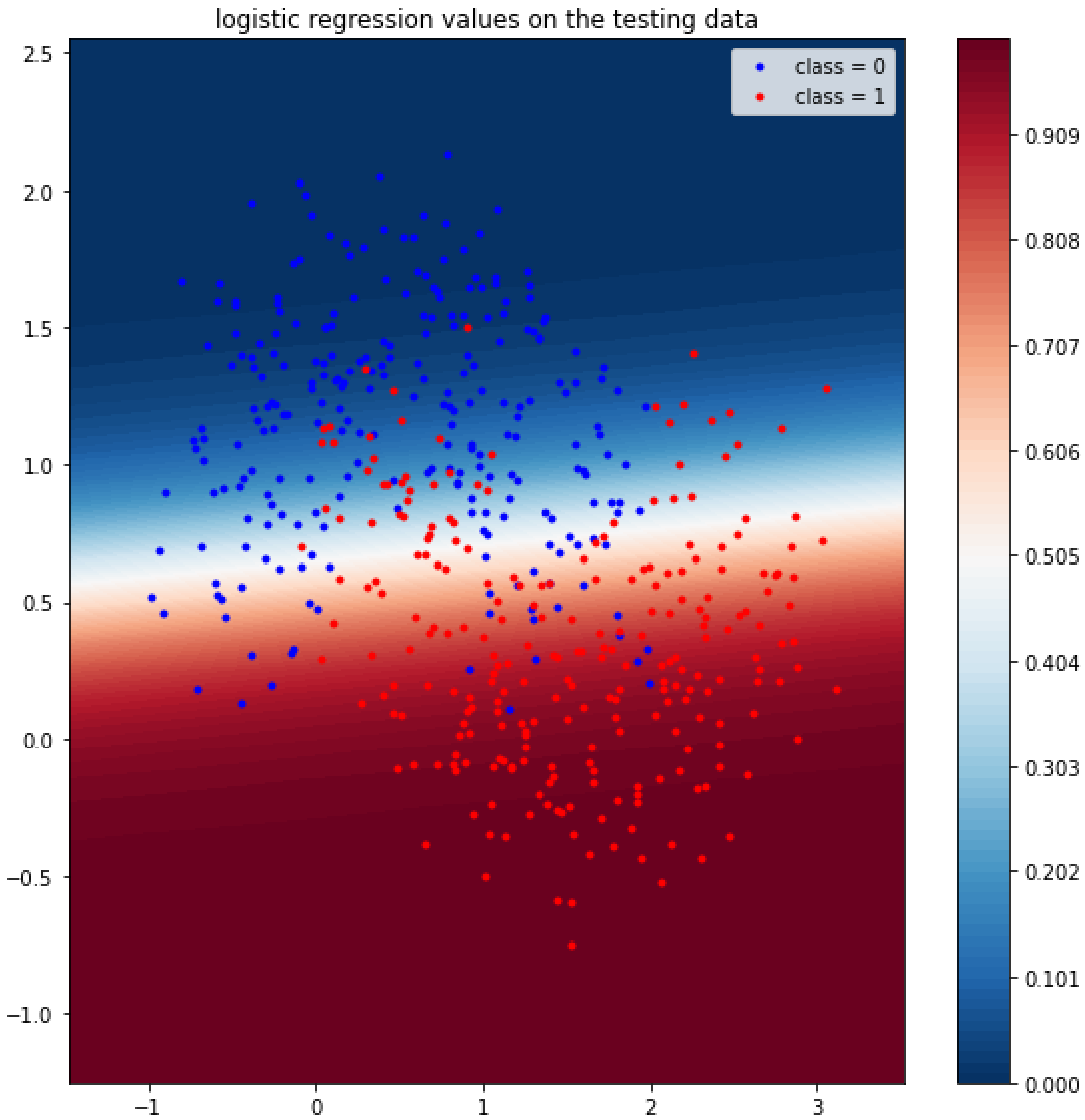
****


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
68.0
54.4
40.8
27.3
13.7
0.2
−13.4
−26.9
−40.5
−54.1

```
****************************************************
## [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
0.909
0.808
0.707
0.606
0.505
0.404
0.303
0.202
0.101
0.000

```
**************************************************
## [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
0.909
0.808
0.707
0.606
0.505
0.404
0.303
0.202
0.101
0.000

In [ ]: