

## Supervised classification - improving capacity learning

---

### 0. Import library

---

Import library

```
1 # Import libraries
2
3 # math library
4 import numpy as np
5
6 # visualization library
7 %matplotlib inline
8 from IPython.display import set_matplotlib_formats
9 set_matplotlib_formats('png2x','pdf')
10 import matplotlib.pyplot as plt
11
12 # machine learning library
13 from sklearn.linear_model import LogisticRegression
14
15 # 3d visualization
16 from mpl_toolkits.mplot3d import axes3d
17
18 # computational time
19 import time
20
21 import math
```

### 1. Load and plot the dataset (dataset-noise-02.txt)

---

The data features for each data  $i$  are  $x_i = (x_{i(1)}, x_{i(2)})$ .

The data label/target,  $y_i$ , indicates two classes with value 0 or 1.

Plot the data points.

You may use matplotlib function `scatter(x,y)`.

```
1 # import data with numpy
2 data = np.loadtxt('/content/drive/My Drive/Colab Notebooks/MachineLearningProject/05/dataset-noise-02.txt')
3
4 # number of training data
5 n = data.shape[0]
6 print('Number of the data = {}'.format(n))
7 print('Shape of the data = {}'.format(data.shape))
8 print('Data type of the data = {}'.format(data.dtype))
9
10 # plot
11 x1 = data[:,0].astype(np.float64) # feature 1
12 x2 = data[:,1].astype(np.float64) # feature 2
13 idx = data[:,2].astype(np.float64) # label
14
15 x1_idx0 = x1[idx == 0]
16 x1_idx1 = x1[idx == 1]
17
18 x2_idx0 = x2[idx == 0]
19 x2_idx1 = x2[idx == 1]
```

```

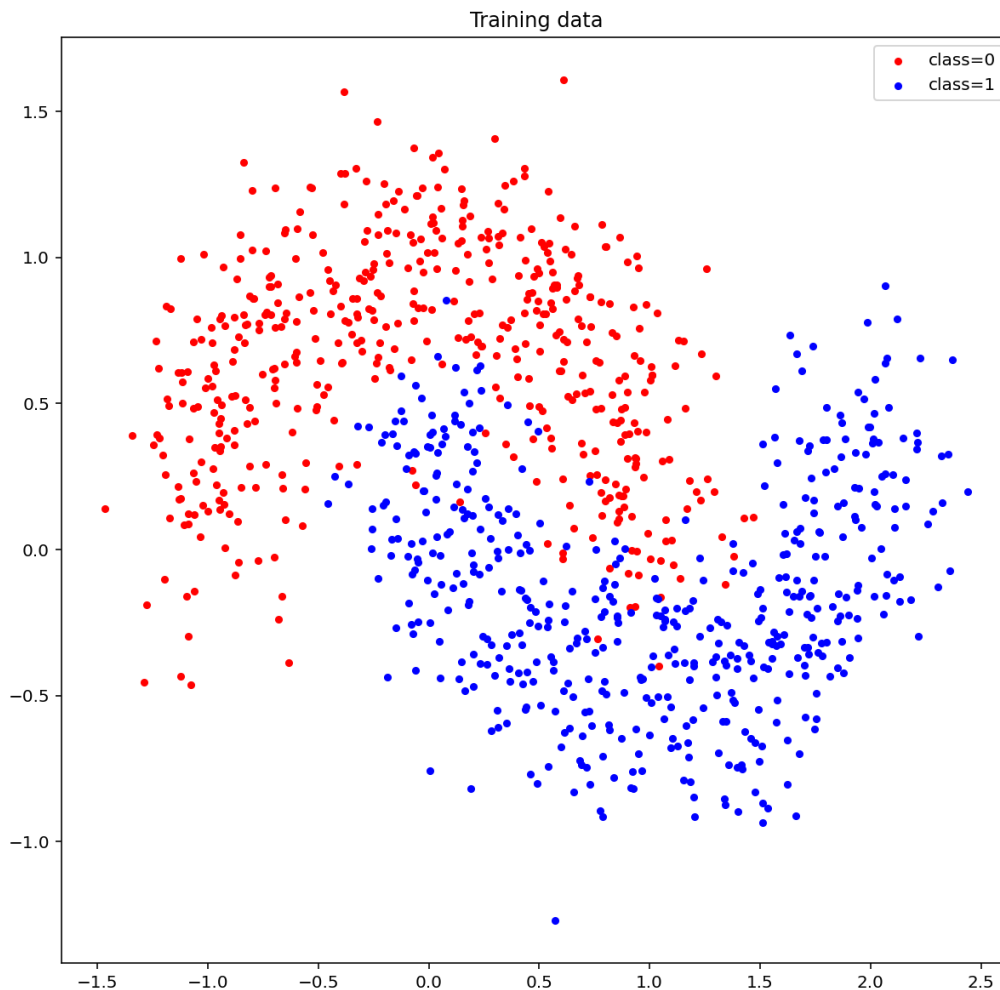
20
21 plt.figure(1,figsize=(10,10))
22 plt.scatter(x1_idx0, x2_idx0 , s=50, c='r', marker='.', label='class=0')
23 plt.scatter(x1_idx1, x2_idx1 , s=50, c='b', marker='.', label='class=1')
24 plt.title('Training data')
25 plt.legend()
26 plt.show()

```

```

↳ Number of the data = 1000
Shape of the data = (1000, 3)
Data type of the data = float64

```



## ▼ 2. Define a logistic regression loss function and its gradient

```

1 # sigmoid function
2 def sigmoid(z):
3     try:
4         return 1 / (1 + np.exp(-z))
5     except OverflowError:
6         return 1e-9
7
8 # predictive function definition
9 def f_pred(X,w):
10     p = np.dot(X,w)
11     return p
12

```

```

12
13 # loss function definition
14 def loss_logreg(y_pred,y):
15     n = len(y)
16     #loss = (np.dot((sigmoid(y_pred) - y).T, (sigmoid(y_pred) - y))) / n
17     loss = -(np.dot(y.T, np.log(sigmoid(y_pred))) + np.dot((1-y).T, np.log(1-sigmoid(y_pred)))) / n
18     return loss
19
20 # gradient function definition
21 def grad_loss(y_pred, y, X):
22     n = len(y)
23     #grad = 2 * np.dot(X.T, np.dot((sigmoid(y_pred)-y), np.dot(sigmoid(y_pred).T, (1-sigmoid(y_pred)))) / n
24     grad = 2 * np.dot(X.T, (sigmoid(y_pred) - y)) / n
25     return grad
26
27 # gradient descent function definition
28 def grad_desc(X, y , w_init, tau, max_iter):
29
30     L_iters = np.zeros([max_iter]) # record the loss values
31     w = w_init # initialization
32     for i in range(max_iter): # loop over the iterations
33         y_pred = f_pred(X,w) # linear prediction function
34         grad_f = grad_loss(y_pred,y,X) # gradient of the loss
35         w = w - tau* grad_f # update rule of gradient descent
36         L_iters[i] = loss_logreg(y_pred,y) # save the current loss value
37
38     return w, L_iters

```

### 3. define a prediction function and run a gradient descent algorithm

The logistic regression/classification predictive function is defined as:

$$p_w(x) = \sigma(Xw)$$

The prediction function can be defined in terms of the following feature functions  $f_i$  as follows:

$$X = \begin{bmatrix} f_0(x_1) & f_1(x_1) & f_2(x_1) & f_3(x_1) & f_4(x_1) & f_5(x_1) & f_6(x_1) & f_7(x_1) & f_8(x_1) & f_9(x_1) \\ f_0(x_2) & f_1(x_2) & f_2(x_2) & f_3(x_2) & f_4(x_2) & f_5(x_2) & f_6(x_2) & f_7(x_2) & f_8(x_2) & f_9(x_2) \\ \vdots & & & & & & & & & \\ f_0(x_n) & f_1(x_n) & f_2(x_n) & f_3(x_n) & f_4(x_n) & f_5(x_n) & f_6(x_n) & f_7(x_n) & f_8(x_n) & f_9(x_n) \end{bmatrix} \quad \text{and} \quad w =$$

where  $x_i = (x_i(1), x_i(2))$  and you can define a feature function  $f_i$  as you want.

You can use at most 10 feature functions  $f_i, i = 0, 1, 2, \dots, 9$  in such a way that the classification accuracy is maximized.

You are allowed to use less than 10 feature functions.

Implement the logistic regression function with gradient descent using a vectorization scheme.

```

1 # construct the data matrix X, and label vector y
2 def poly(X1, X2, degree):
3     func = np.ones(len(X1))
4     for i in range(1, degree+1):
5         for j in range(0, i+1):

```

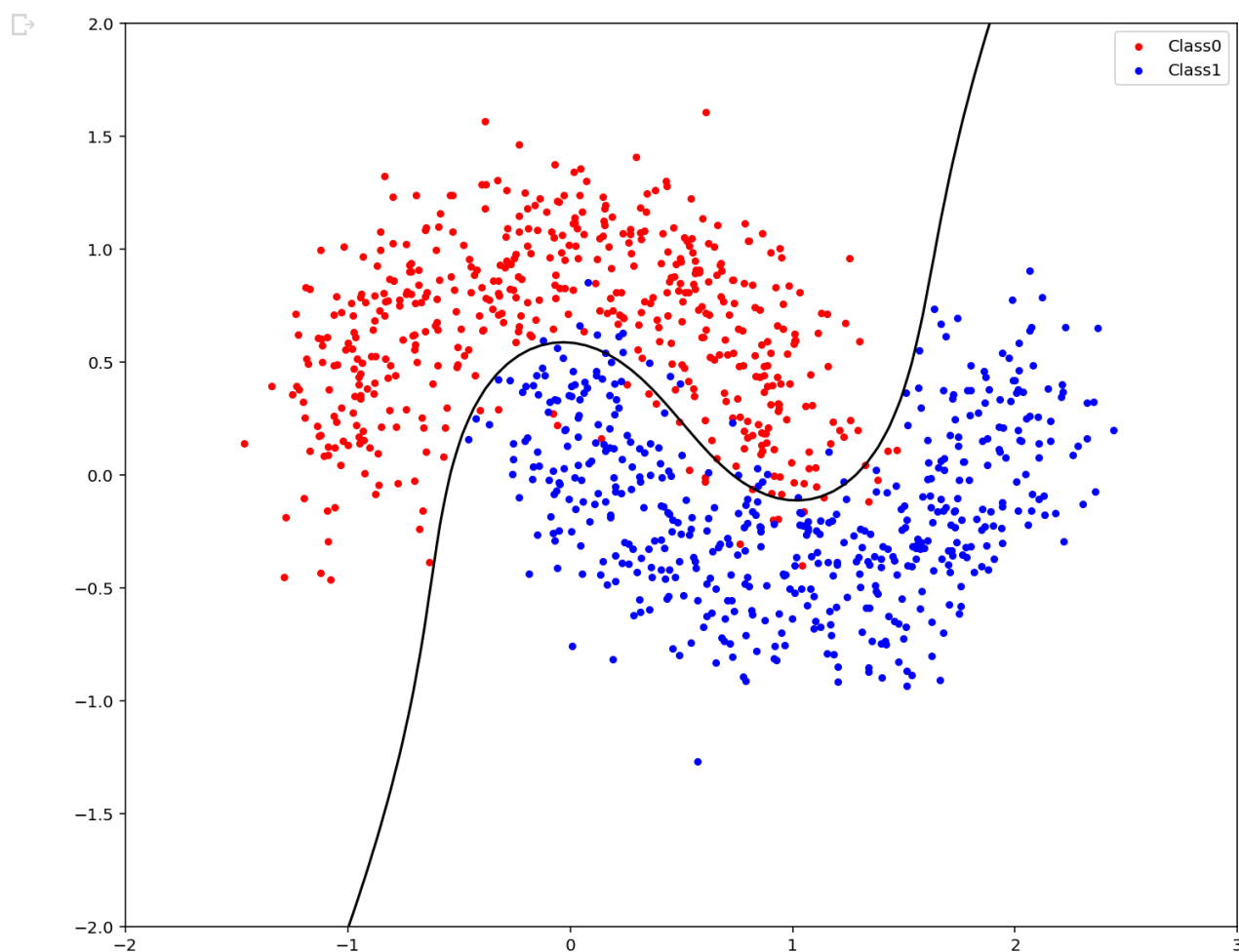


```

3 plt.scatter(x1_0, x2_0, s=50, c='r', marker='.', label='Class0')
4 plt.scatter(x1_1, x2_1, s=50, c='b', marker='.', label='Class1')
5
6 X = np.linspace(-2, 3, 100)
7 Y = np.linspace(-2, 2, 100)
8 XX, YY = np.meshgrid(X,Y)
9 XX = np.ravel(XX)
10 YY = np.ravel(YY)
11
12 Z = np.zeros((len(X)*len(Y)))
13 poly_line = poly(XX, YY, degree)
14 Z = poly_line.dot(w)
15
16 XX = XX.reshape((len(X), len(Y)))
17 YY = YY.reshape((len(X), len(Y)))
18 Z = Z.reshape((len(X), len(Y)))
19 plt.contour(XX, YY, Z, levels=[0], colors='k')
20 plt.legend()
21 plt.show()

```

```
1 boundary(x1_idx0, x2_idx0, x1_idx1, x2_idx1, w, 3)
```



## 5. Plot the probability map

```

1 def boundary_map(x1_0, x2_0, x1_1, x2_1, w, degree):
2     plt.figure(4, figsize=(12, 10))
3
4     X = np.linspace(-2, 3, 100)
5     Y = np.linspace(-2, 2, 100)
6     XX, YY = np.meshgrid(X,Y)
7     XX = np.ravel(XX)
8     YY = np.ravel(YY)
9
10    Z = np.zeros((len(X)*len(Y)))
11    poly_line = poly(XX, YY, degree)
12    Z = poly_line.dot(w)
13
14    XX = XX.reshape((len(X), len(Y)))
15    YY = YY.reshape((len(X), len(Y)))
16    Z = Z.reshape((len(X), len(Y)))
17    plt.contour(XX, YY, Z, levels=[0], colors='k')
18    plt.legend()
19    plt.show()

```

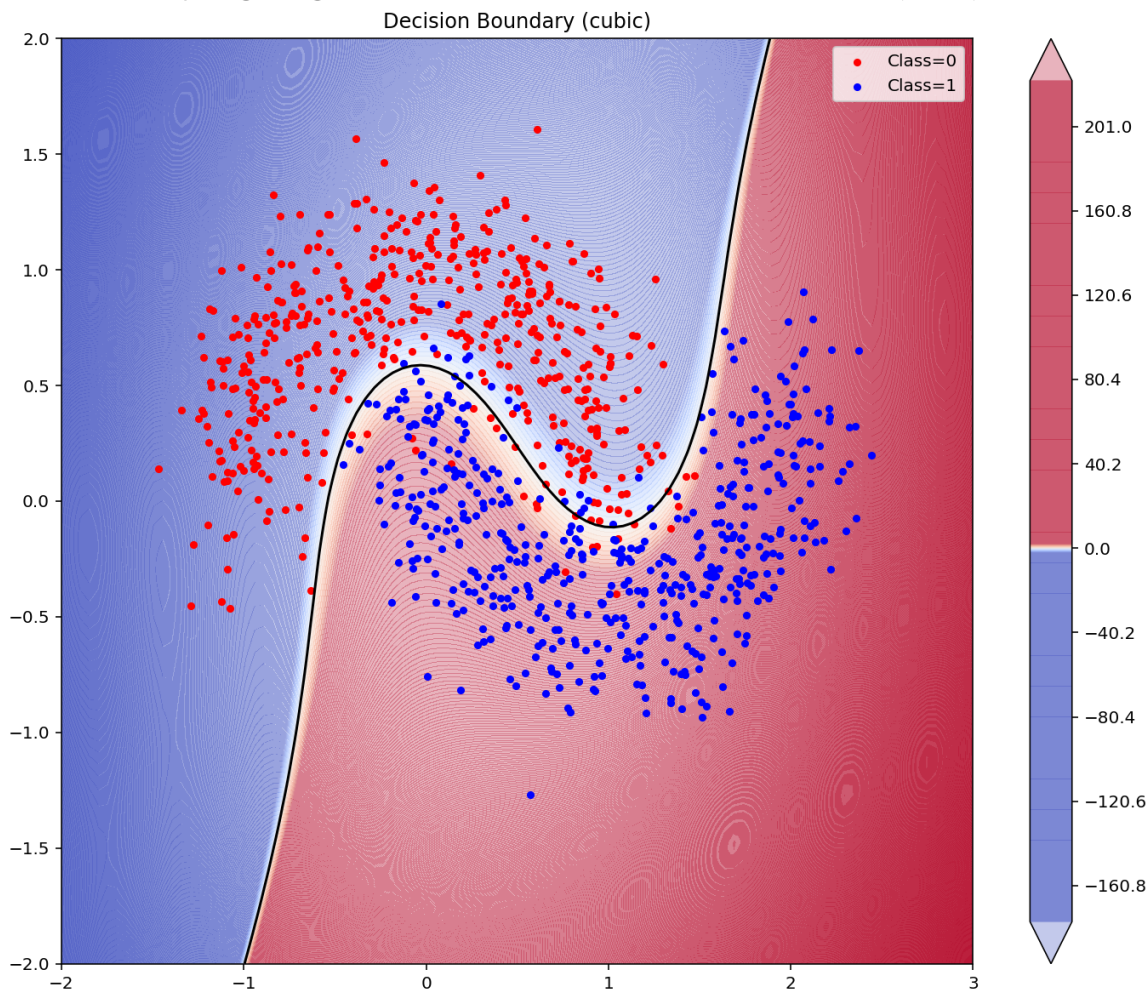
```

6  XX, YY = np.meshgrid(X,Y)
7  XX = np.ravel(XX)
8  YY = np.ravel(YY)
9
10 Z = np.zeros((len(X)*len(Y)))
11 poly_line = poly(XX, YY, degree)
12 Z = poly_line.dot(w)
13
14 XX = XX.reshape((len(X), len(Y)))
15 YY = YY.reshape((len(X), len(Y)))
16 Z = Z.reshape((len(X), len(Y)))
17
18 ax = plt.contourf(XX,YY,Z,2500,vmin=-2,vmax=2,cmap='coolwarm',alpha=0.3,extend='both')
19 cbar = plt.colorbar(ax)
20 cbar.update_ticks()
21
22 plt.scatter(x1_0, x2_0, s=50, c='r', marker='.', label='Class=0')
23 plt.scatter(x1_1, x2_1, s=50, c='b', marker='.', label='Class=1')
24 plt.contour(XX, YY, Z, levels=[0], colors='k')
25 plt.legend()
26 plt.title('Decision Boundary (cubic)')
27 plt.show()

```

```
1 boundary_map(x1_idx0, x2_idx0, x1_idx1, x2_idx1, w, 3)
```

⚠ Locator attempting to generate 2007 ticks ([-178.20000000000002, ..., 223.00000000000003]), which exceeds



## 6. Compute the classification accuracy

The accuracy is computed by:

$$\text{accuracy} = \frac{\text{number of correctly classified data}}{\text{total number of data}}$$

```

1 # compute the accuracy of the classifier
2 n = data.shape[0]
3
4 # plot
5 x1 = data[:,0].astype(np.float64) # feature 1
6 x2 = data[:,1].astype(np.float64) # feature 2
7
8 X = poly(x1, x2, 3)
9 y = data[:,2][:,None] # label
10 p = f_pred(X,w)
11
12 tmp = []
13 for i, j in zip(p, y):
14     if np.round(sigmoid(i)) == j:
15         tmp.append(1)
16
17 print('total number of data = {}'.format(n))
18 print('total number of correctly classified data = ', len(tmp))
19 print('accuracy(%) = ', 100*len(tmp) / len(data))

```

➤ total number of data = 1000  
 total number of correctly classified data = 957  
 accuracy(%) = 95.7

## Output using the dataset (dataset-noise-02.txt)

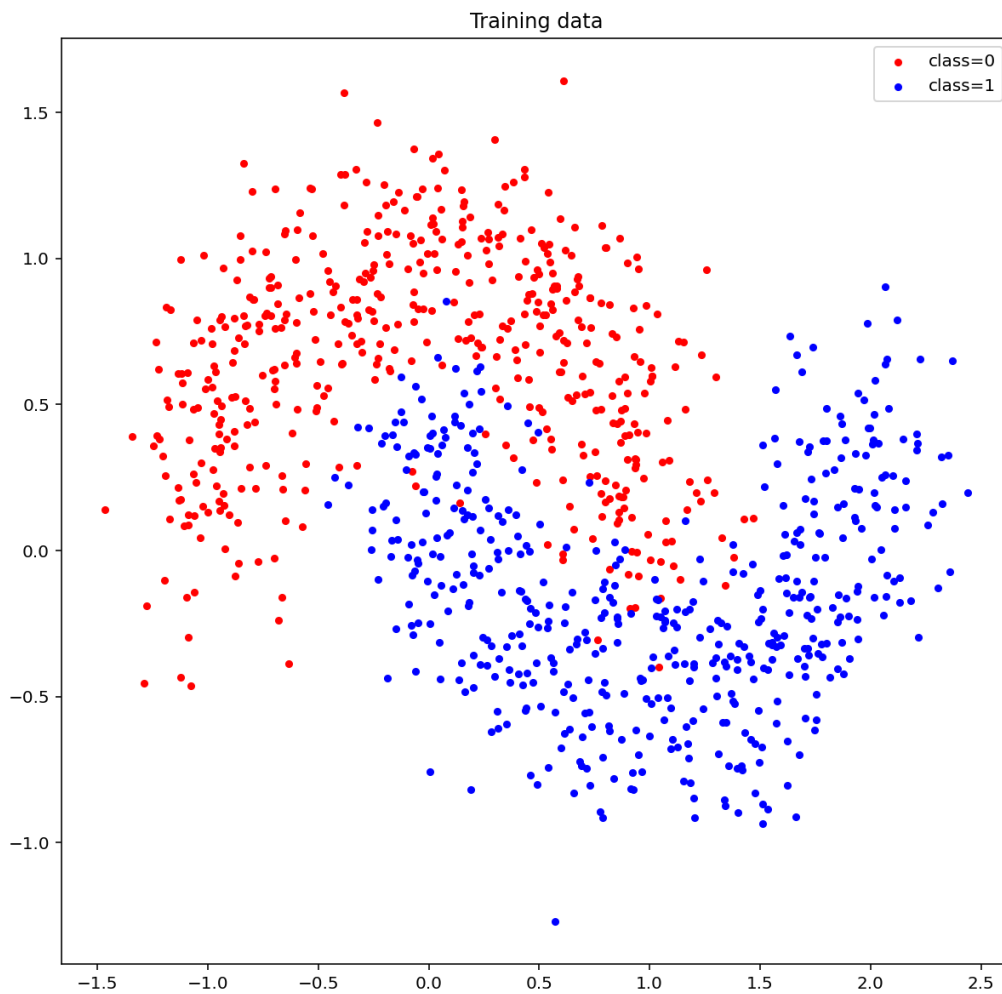
---

### ▼ 1. Visualize the data [1pt]

```

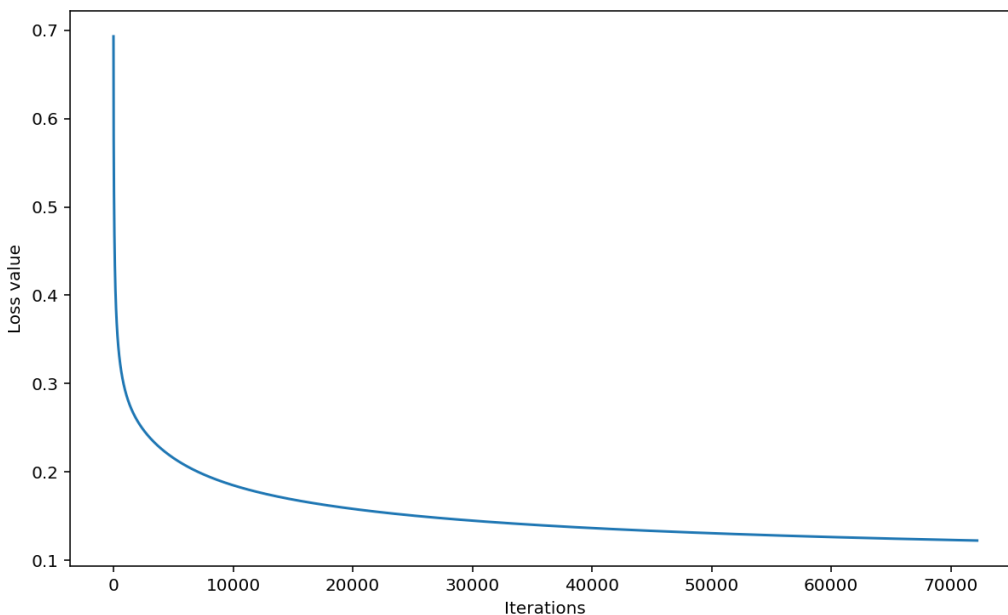
1 plt.figure(1,figsize=(10,10))
2 plt.scatter(x1_idx0, x2_idx0 , s=50, c='r', marker='.', label='class=0')
3 plt.scatter(x1_idx1, x2_idx1 , s=50, c='b', marker='.', label='class=1')
4 plt.title('Training data')
5 plt.legend()
6 plt.show()

```



2. Plot the loss curve obtained by the gradient descent until the convergence [2pt]

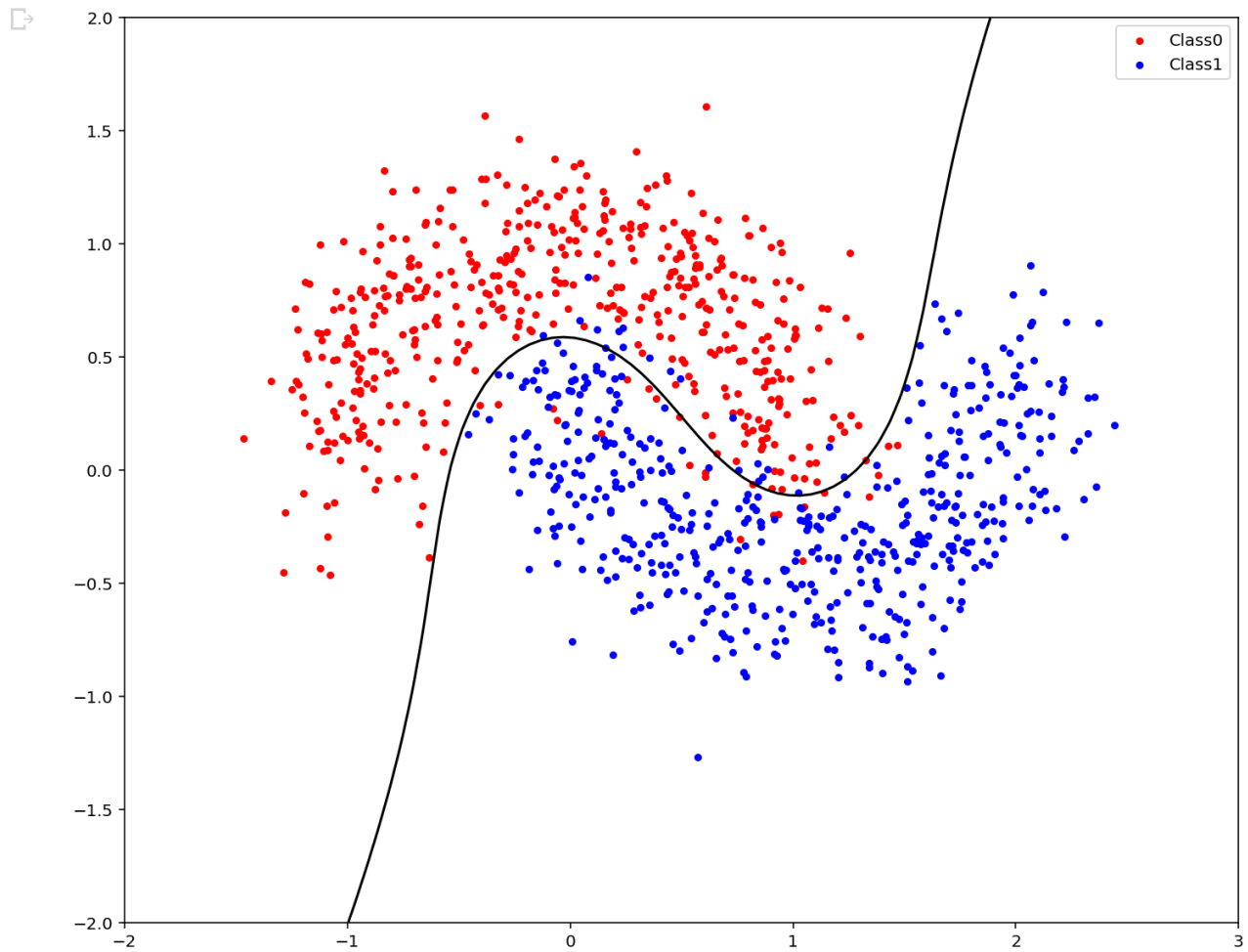
```
1 plt.figure(3, figsize=(10,6))  
2 plt.plot(np.array(range(max_iter)), L_iters)  
3 plt.xlabel('Iterations')  
4 plt.ylabel('Loss value')  
5 plt.show()
```





3. Plot the decision boundary of the obtained classifier [2pt]

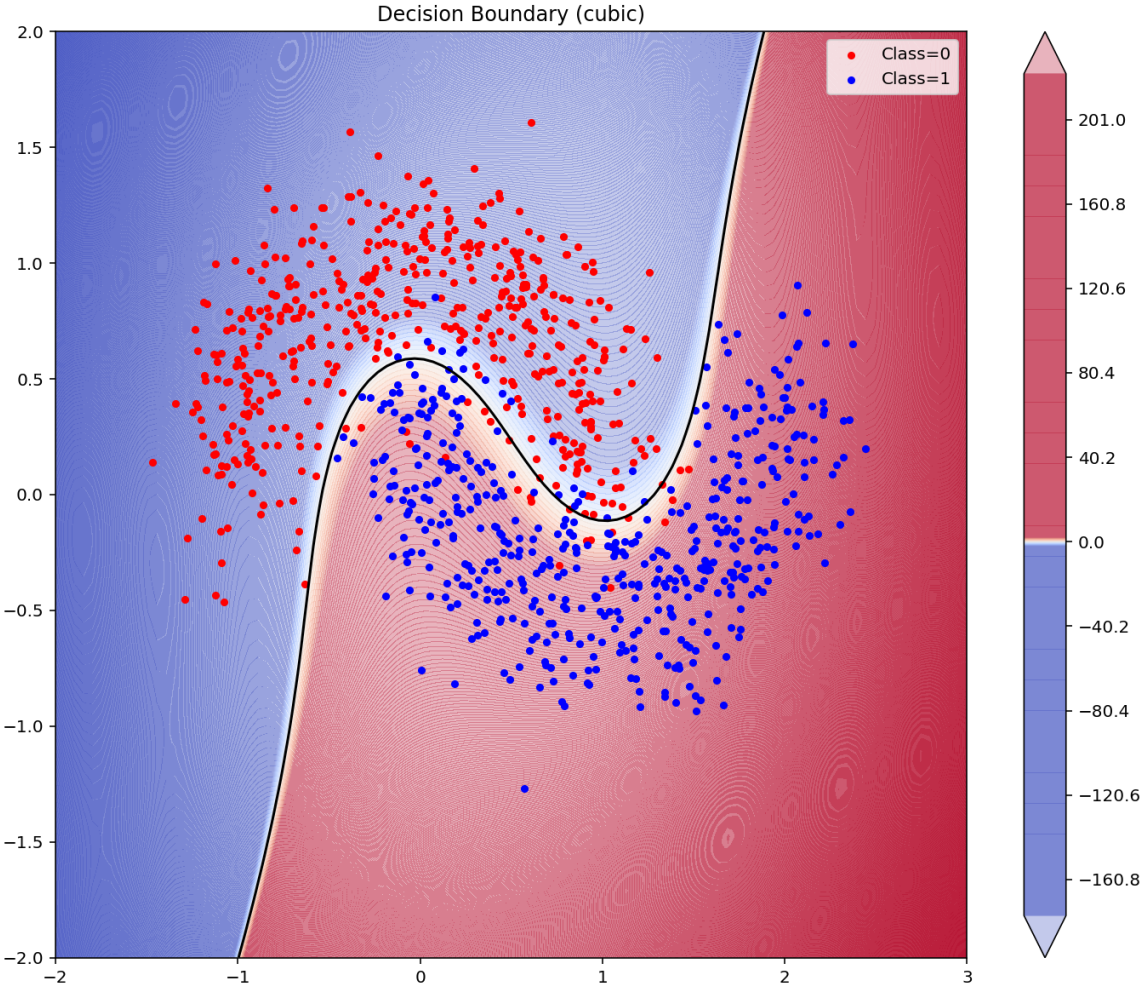
```
1 boundary(x1_idx0, x2_idx0, x1_idx1, x2_idx1, w, 3)
```



4. Plot the probability map of the obtained classifier [2pt]

```
1 boundary_map(x1_idx0, x2_idx0, x1_idx1, x2_idx1, w, 3)
```

Locator attempting to generate 2007 ticks ([-178.20000000000002, ..., 223.00000000000003]), which exceeds



5. Compute the classification accuracy [1pt]

```
1 print('total number of data = {}'.format(n))
2 print('total number of correctly classified data = ', len(tmp))
3 print('accuracy(%) = ', 100*len(tmp) / len(data))
```

total number of data = 1000  
total number of correctly classified data = 957  
accuracy(%) = 95.7