Update assignment-06.md
Byung-Woo Hong authored 1 day ago

669c988a

assignment-06.md 7.94 KB

Logistic regression for a binary classification with a regularization

1. Training Data

- the training data are given by the file `training.txt`
- each element of the training data consists of $\{(x^{(i)}, y^{(i)}, l^{(i)})\}$ where

  ```
  (x, y)
  ```

  denotes a 2-dimensional point and

  ```
  l
  ```

  denotes its label
- ```
  (x, y) \in \mathbb{R}^2
  ```

  and

  ```
  l \in \{0, 1\}
  ```

2. Testing Data

- the training data are given by the file `testing.txt`
- each element of the training data consists of

  ```
  \{ (x^{(i)}, y^{(i)}, l^{(i)}) \}
  ```

  where

  ```
  (x, y)
  ```

  denotes a 2-dimensional point and

  ```
  l
  ```

  denotes its label
- ```
  (x, y) \in \mathbb{R}^2
  ```

  and

  ```
  l \in \{0, 1\}
  ```

3. Logistic regression with a high dimensional feature function

- ```
  p_w(x, y) = \sigma(z)
  ```

- ```
  z = g(x, y; \theta)
  ```

  , where

  ```
  g
  ```

  is a high dimensional function and

  ```
  \theta \in \mathbb{R}^{100}
  ```

- ```
  \theta = (\theta_{0,0}, \theta_{0,1}, \cdots, \theta_{9,9})
  ```

- ```
  g(x, y ; \theta) = \sum_{i=0}^{9} \sum_{j=0}^{9} \theta_{i,j} x^{i} y^{j}
  ```

- $$\sigma(z) = \frac{1}{1 + \exp(-z)}$$
- $$\sigma^{\prime}(z) = \sigma(z) (1 - \sigma(z))$$

4. Objective Function with a regularization term

- $$J(\theta) = \frac{1}{m} \sum_{i=1}^m \left[- l^{(i)} \log(\sigma(z^{(i)})) - (1 - l^{(i)}) \log(1 - \sigma(z^{(i)})) \right] + \frac{\lambda}{2} \sum_{i=0}^{9} \sum_{j=0}^{9} \theta_{i,j}^2$$
- $$z^{(i)} = g(x^{(i)}, y^{(i)}; \theta)$$
- $$g(x, y ; \theta) = \sum_{i=0}^{9} \sum_{j=0}^{9} \theta_{i,j} x^{i} y^{j}$$
- the degree of regularization is determined by the control parameter
  $$\lambda$$
- the larger value of
  $$\lambda$$
  yields smoother classification boundary

5. Gradient Descent

- $$\theta_{i, j}^{(t+1)} \coloneqq \theta_{i, j}^{(t)} - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (\sigma(z^{(i)}) - l^{(i)}) \frac{\partial z^{(i)}}{\partial \theta_{i, j}} + \lambda \theta_{i, j}^{(t)} \right]$$
  , for all
  $$i, j$$
- you can use random initialization for the initial status of the model parameters
  $$\theta_{i, j}^{(0)}$$
  for all
  $$i, j$$

6. Hyper-parameter

- you can apply a annealing scheme for the learning rate that is scheduled as the gradient descent iteration proceeds
- the application of the learning rate annealing should lead to the convergence of the optimization
- demonstrate the effect of the regularization parameter with varying parameter values
  $$\lambda = 0.00001, 0.0001, 0.001, 0.01, 0.1$$

7. Training

- find an optimal set of parameters
  $$\theta$$
  using the training data with a given value of regularization parameter
  $$\lambda$$

8. Compute the training accuracy

- the training accuracy is computed by
  $$\frac{\textrm{number of correct predictions}}{\textrm{total number of predictions}}$$
  using the training data

9. Compute the testing accuracy

- the testing accuracy is computed by
  $$\frac{\textrm{number of correct predictions}}{\textrm{total number of predictions}}$$
  using the testing data

Code

- load the data from the files

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

# import data with numpy
data_train  = np.loadtxt('training.txt', delimiter=',')
data_test   = np.loadtxt('testing.txt', delimiter=',')

# number of training data
number_data_train   = data_train.shape[0]
number_data_test    = data_test.shape[0]

# training data
x1_train            = data_train[:,0] # feature 1
x2_train            = data_train[:,1] # feature 2
idx_class0_train    = (data_train[:,2]==0) # index of class0
idx_class1_train    = (data_train[:,2]==1) # index of class1

# testing data
x1_test             = data_test[:,0] # feature 1
x2_test             = data_test[:,1] # feature 2
idx_class0_test     = (data_test[:,2]==0) # index of class0
idx_class1_test     = (data_test[:,2]==1) # index of class1
```

Submission

Github history [1pt]

- Use the `git` commands `commit` and `push` at your `github` account for the notebook
- Lease a history for the development of each meaningful block of the codes at `github`
- Make at least 10 `commit`
- Save and submit the history page in PDF format

Python notebook

- the submission should be made in PDF format
- the notebook should consists of the following two parts:
  - main codes, comments and results
  - visualization codes and outputs

[output]

1. Plot the training data [0.5pt]

- plot the training data points

  ```
  (x, y)
  ```

  with their labels

  ```
  l
  ```

  in colors (red for label 0 and blue for label 1)

2. Plot the testing data [0.5pt]

- plot the testing data points

  ```
  (x, y)
  ```

  with their labels

  ```
  l
  ```

  in colors (red for label 0 and blue for label 1)

3. Plot the learning curve with

```
\lambda = 0.00001
```

[1pt]

- plot both the training loss in blue and the testing loss in red at the same figure

- the x-axis represents the gradient descent iteration
- the y-axis represents the loss value

4. Plot the learning curve with

```
\lambda = 0.0001
```

[1pt]

- plot both the training loss in blue and the testing loss in red at the same figure
- the x-axis represents the gradient descent iteration
- the y-axis represents the loss value

5. Plot the learning curve with

```
\lambda = 0.001
```

[1pt]

- plot both the training loss in blue and the testing loss in red at the same figure
- the x-axis represents the gradient descent iteration
- the y-axis represents the loss value

6. Plot the learning curve with

```
\lambda = 0.01
```

[1pt]

- plot both the training loss in blue and the testing loss in red at the same figure
- the x-axis represents the gradient descent iteration
- the y-axis represents the loss value

7. Plot the learning curve with

```
\lambda = 0.1
```

[1pt]

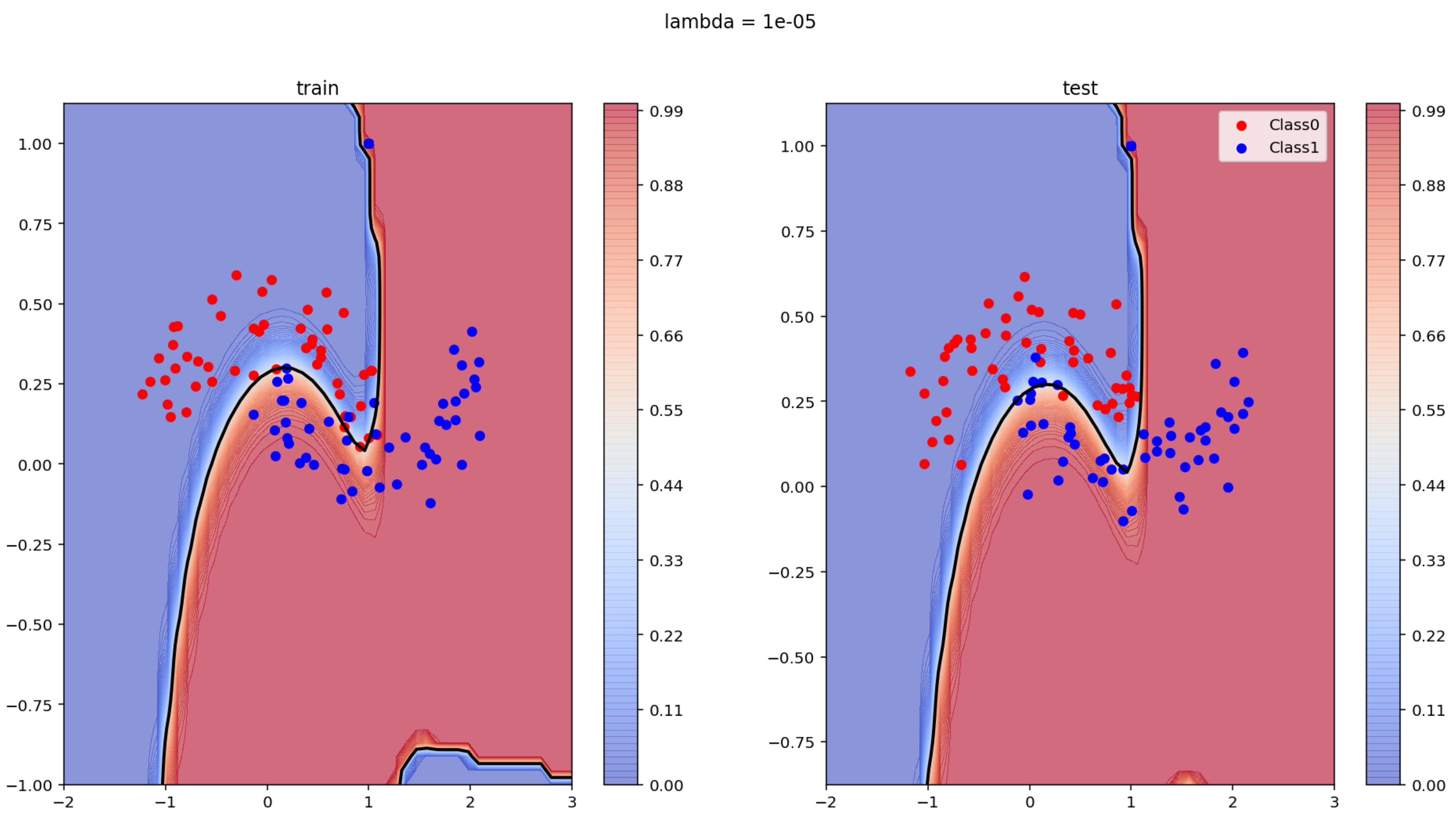
- plot both the training loss in blue and the testing loss in red at the same figure
- the x-axis represents the gradient descent iteration
- the y-axis represents the loss value

8. Plot the probability map of the obtained classifier with

```
\lambda = 0.00001
```

[1pt]

- plot the probability map on the training data on the left
- plot the probability map on the testing data on the right

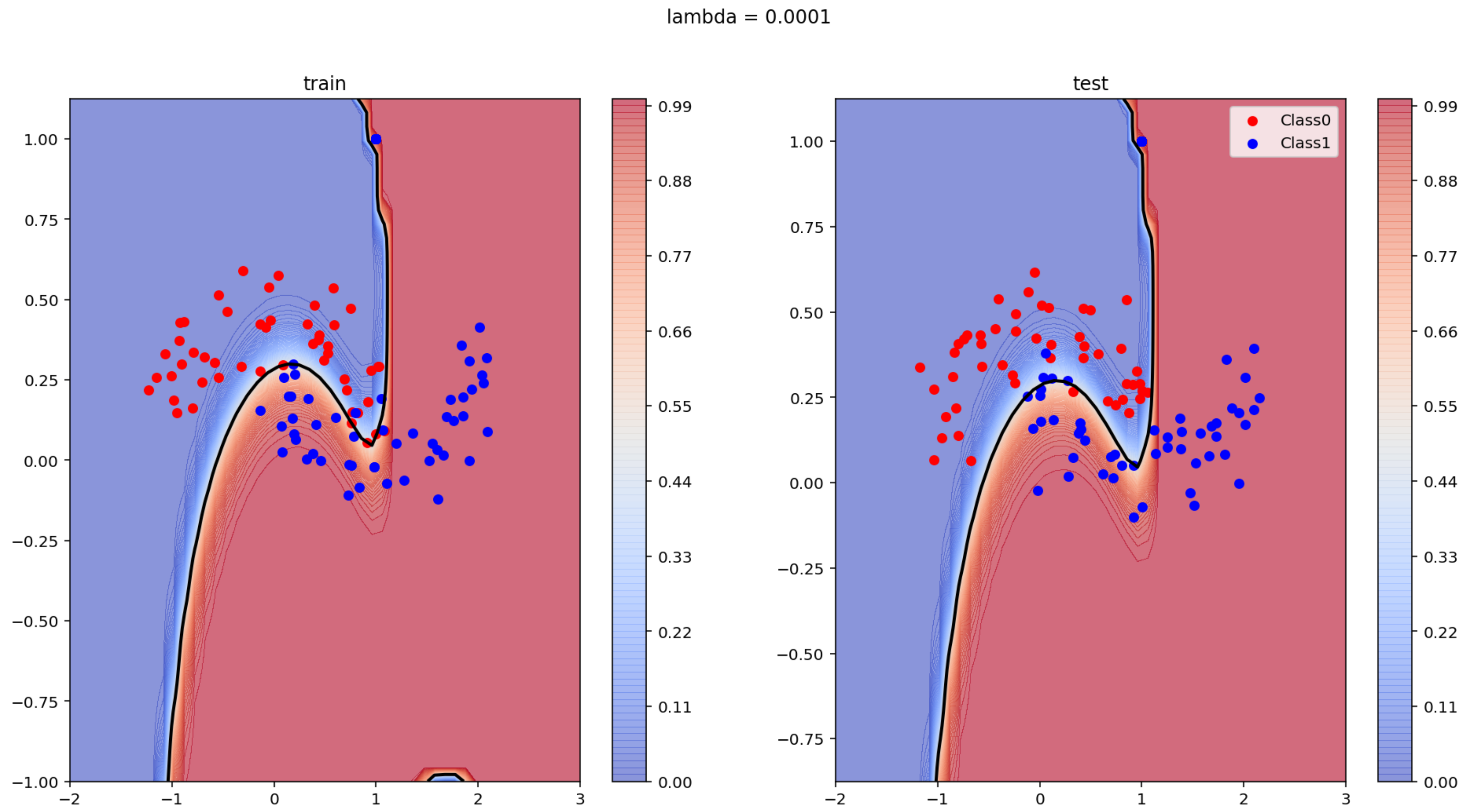


9. Plot the probability map of the obtained classifier with

```
\lambda = 0.0001
```

[1pt]

- plot the probability map on the training data on the left
- plot the probability map on the testing data on the right

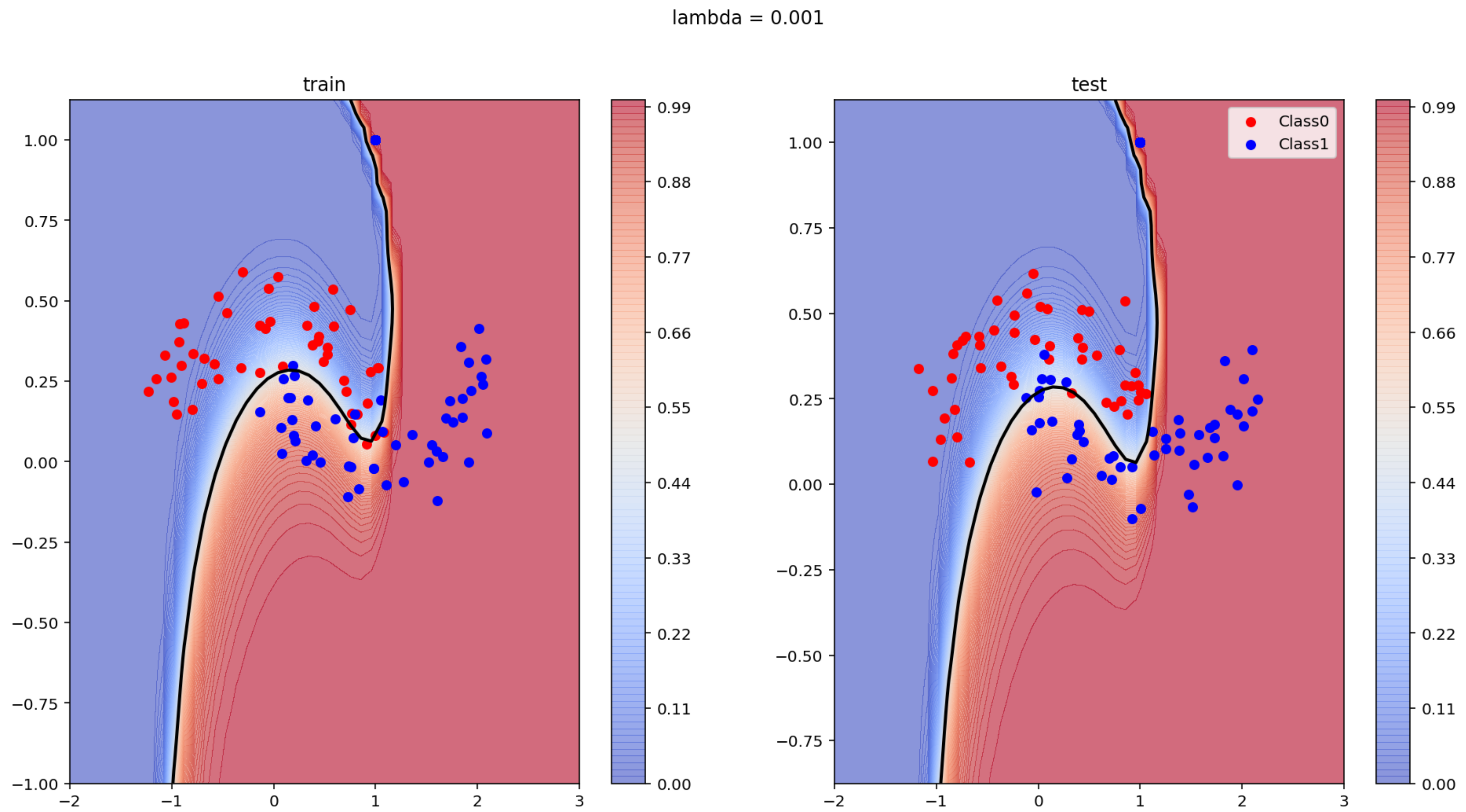


10. Plot the probability map of the obtained classifier with

```
\lambda = 0.001
```

[1pt]

- plot the probability map on the training data on the left
- plot the probability map on the testing data on the right

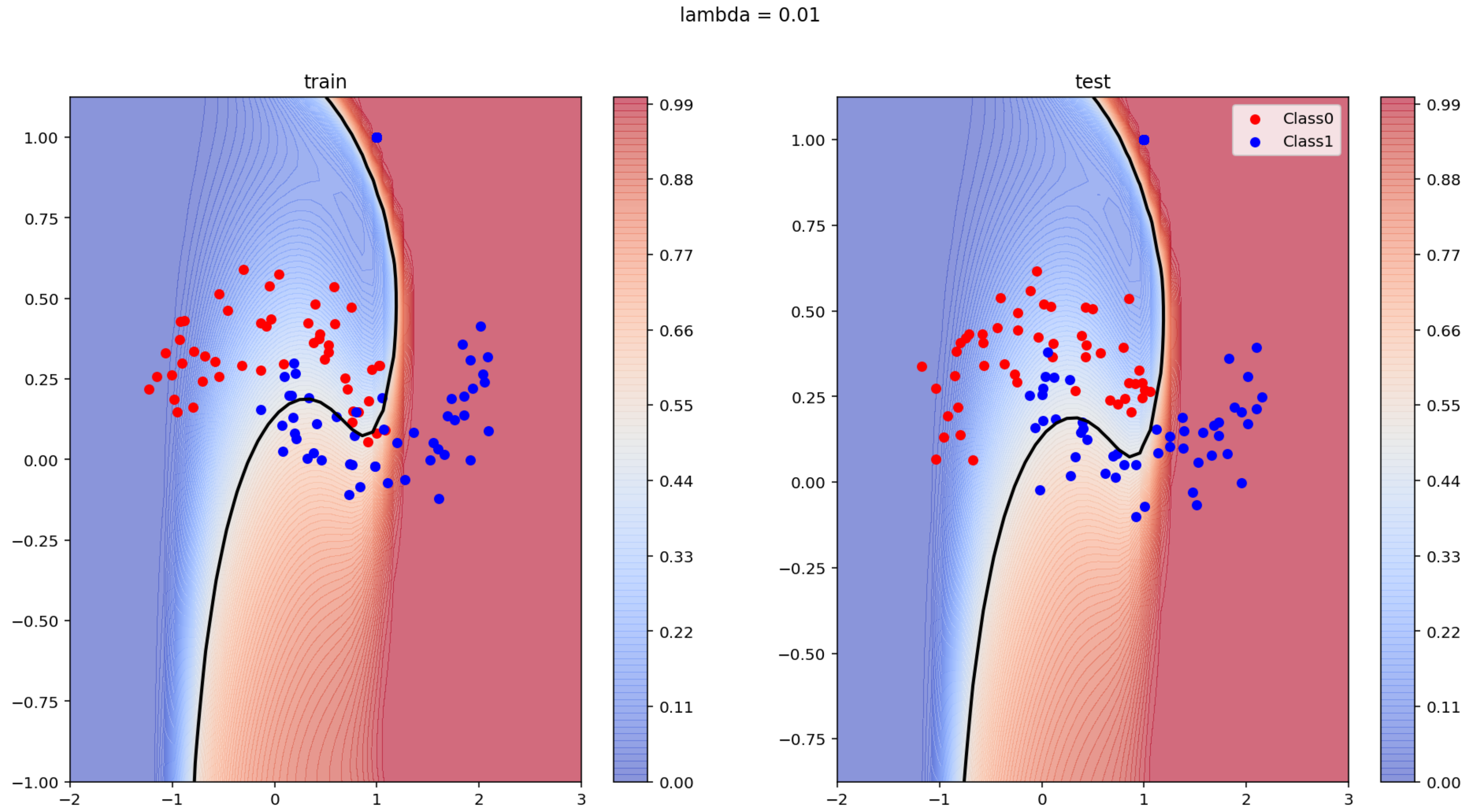


11. Plot the probability map of the obtained classifier with

```
\lambda = 0.01
```

[1pt]

- plot the probability map on the training data on the left
- plot the probability map on the testing data on the right

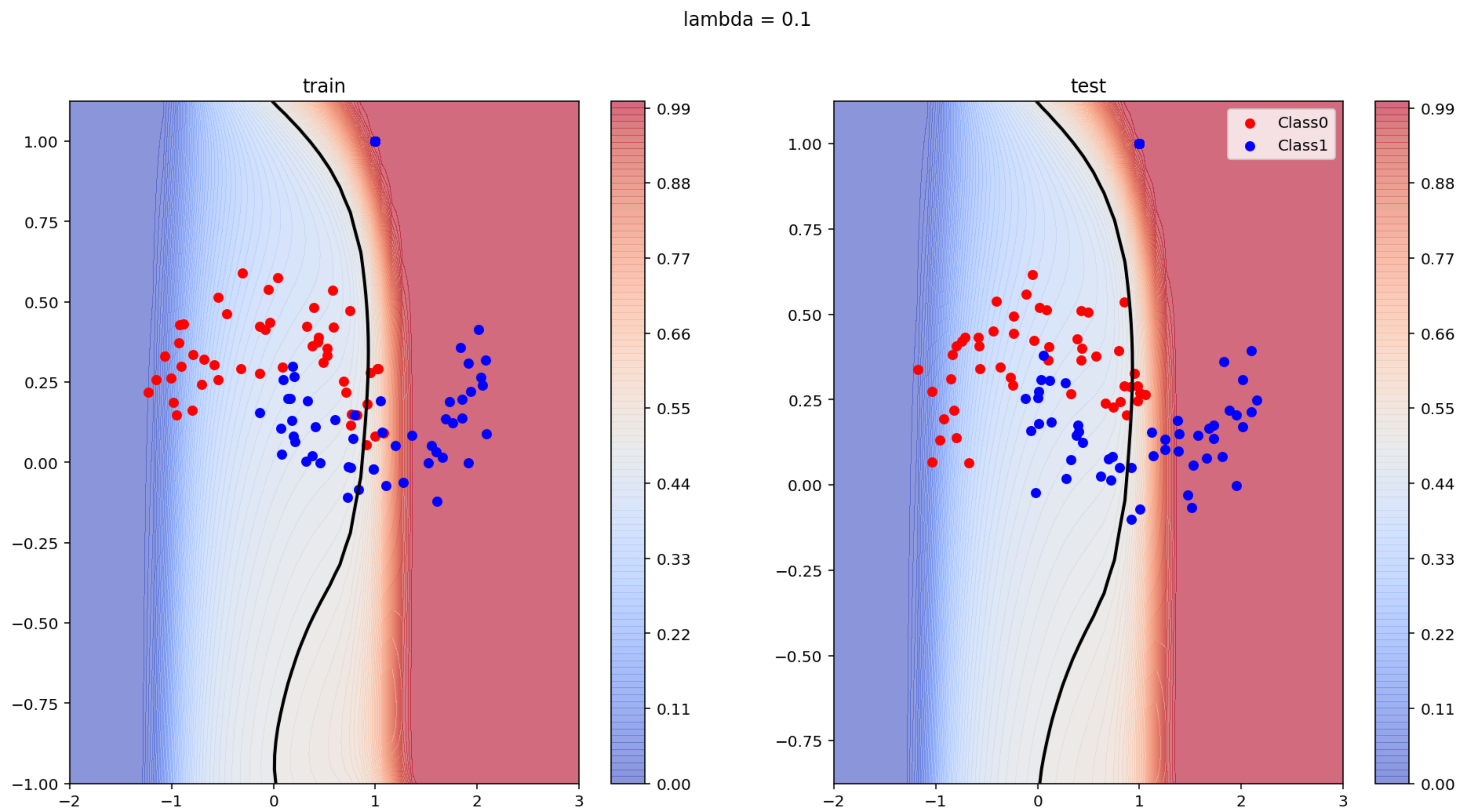


12. Plot the probability map of the obtained classifier with

```
\lambda = 0.1
```

[1pt]

- plot the probability map on the training data on the left
- plot the probability map on the testing data on the right



13. Print the final training accuracy with the given regularization parameters [2.5pt]

- the accuracy is computed based on the training data with varying regularization parameters

`\lambda`	**Training Accuracy (%)**
0.00001	
0.0001	
0.001	

`\lambda`	**Training Accuracy (%)**
0.01	
0.1	

14. Print the final testing accuracy with the given regularization parameters [2.5pt]

- the accuracy is computed based on the testing data with varying regularization parameters

`\lambda`	**Testing Accuracy (%)**
0.00001	
0.0001	
0.001	
0.01	
0.1	