Logistic regression for a binary classification with a regularization

In [549]:

```
import warnings

warnings.filterwarnings(action='ignore')

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import math
from matplotlib import cm
```

1. Training Data

In [114]:

```
data = pd.read_csv("training.txt", names=['x','y','label'])
```

In [115]:

```
data
```

Out[115]:

	x	y	label
0	0.76090	0.116706	0.0
1	0.99777	0.082345	0.0
2	1.02459	0.291094	0.0
3	1.07755	0.091764	0.0
4	0.91385	0.054629	0.0
...	...	...	...
195	1.00000	1.000000	1.0
196	1.00000	1.000000	1.0
197	1.00000	1.000000	1.0
198	1.00000	1.000000	1.0
199	1.00000	1.000000	1.0

200 rows × 3 columns

2. Testing Data

In [116]:

```
test_data = pd.read_csv("testing.txt", names=['x','y','label'])
```

In [117]:

```
test_data
```

Out[117]:

	x	y	label
0	1.00726	0.268272	0.0
1	0.87378	0.204539	0.0
2	0.98254	0.245526	0.0
3	1.05977	0.265070	0.0
4	0.74250	0.228713	0.0
...	...	...	...
195	1.00000	1.000000	1.0
196	1.00000	1.000000	1.0
197	1.00000	1.000000	1.0
198	1.00000	1.000000	1.0
199	1.00000	1.000000	1.0

200 rows × 3 columns

- 라벨값을 저장하고 $g(x,y,\theta)$ 함수를 정의하겠습니다.

In [119]:

```
data_1 = data[data['label'] == 1]
dataX_1 = data_1['x']
dataY_1 = data_1['y']

data_0 = data[data['label'] == 0]
dataX_0 = data_0['x']
dataY_0 = data_0['y']

test_data_1 = test_data[test_data['label'] == 1]
test_dataX_1 = test_data_1['x']
test_dataY_1 = test_data_1['y']

test_data_0 = test_data[test_data['label'] == 0]
test_dataX_0 = test_data_0['x']
test_dataY_0 = test_data_0['y']
```

In [120]:

```
label = data['label']
test_label = test_data['label']
```

In [379]:

```
train_new_data = pd.DataFrame()
test_new_data = pd.DataFrame()

## training data
for i in range(10):
    for j in range(10):
        name = "x" + str(i) + "y" + str(j)
        train_new_data[name] = data['x'] ** i * data['y'] ** j


## testing data
for i in range(10):
    for j in range(10):
        name = "x" + str(i) + "y" + str(j)
        test_new_data[name] = test_data['x'] ** i * test_data['y'] ** j
```

/

In [380]:

```
train_new_data
```

Out[380]:

	x0y0	x0y1	x0y2	x0y3	x0y4	x0y5	x0y6	x0y7	x0y8	x0y9	...	x9y0	x9y1	x9y2
0	1.0	0.116706	0.013620	0.001590	0.000186	2.165069e-05	2.526773e-06	2.948903e-07	3.441556e-08	4.016514e-09	...	0.085496	0.009978	0.001164
1	1.0	0.082345	0.006781	0.000558	0.000046	3.785956e-06	3.117530e-07	2.567118e-08	2.113883e-09	1.740668e-10	...	0.980108	0.080707	0.006646
2	1.0	0.291094	0.084736	0.024666	0.007180	2.090092e-03	6.084129e-04	1.771053e-04	5.155426e-05	1.500713e-05	...	1.244374	0.362230	0.105443
3	1.0	0.091764	0.008421	0.000773	0.000071	6.506846e-06	5.970966e-07	5.479219e-08	5.027971e-09	4.613886e-10	...	1.958560	0.179726	0.016492
4	1.0	0.054629	0.002984	0.000163	0.000009	4.865555e-07	2.658024e-08	1.452063e-09	7.932531e-11	4.333495e-12	...	0.444503	0.024283	0.001327
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
195	1.0	1.000000	1.000000	1.000000	1.000000	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	...	1.000000	1.000000	1.000000
196	1.0	1.000000	1.000000	1.000000	1.000000	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	...	1.000000	1.000000	1.000000
197	1.0	1.000000	1.000000	1.000000	1.000000	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	...	1.000000	1.000000	1.000000
198	1.0	1.000000	1.000000	1.000000	1.000000	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	...	1.000000	1.000000	1.000000
199	1.0	1.000000	1.000000	1.000000	1.000000	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	...	1.000000	1.000000	1.000000

200 rows × 100 columns

In [381]:

```
test_new_data
```

Out[381]:

	x0y0	x0y1	x0y2	x0y3	x0y4	x0y5	x0y6	x0y7	x0y8	x0y9	...	x9y0	x9y1	x9y2	x9y3	x9
0	1.0	0.268272	0.071970	0.019308	0.005180	0.001390	0.000373	0.000100	0.000027	7.197514e-06	...	1.067270	0.286319	0.076811	0.020606	0.0055
1	1.0	0.204539	0.041836	0.008557	0.001750	0.000358	0.000073	0.000015	0.000003	6.265953e-07	...	0.296906	0.060729	0.012421	0.002541	0.0005
2	1.0	0.245526	0.060283	0.014801	0.003634	0.000892	0.000219	0.000054	0.000013	3.242486e-06	...	0.853399	0.209532	0.051445	0.012631	0.0031
3	1.0	0.265070	0.070262	0.018624	0.004937	0.001309	0.000347	0.000092	0.000024	6.460147e-06	...	1.686183	0.446956	0.118475	0.031404	0.0083
4	1.0	0.228713	0.052310	0.011964	0.002736	0.000626	0.000143	0.000033	0.000007	1.712468e-06	...	0.068591	0.015688	0.003588	0.000821	0.0001
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
195	1.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000e+00	...	1.000000	1.000000	1.000000	1.000000	1.0000
196	1.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000e+00	...	1.000000	1.000000	1.000000	1.000000	1.0000
197	1.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000e+00	...	1.000000	1.000000	1.000000	1.000000	1.0000
198	1.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000e+00	...	1.000000	1.000000	1.000000	1.000000	1.0000
199	1.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000e+00	...	1.000000	1.000000	1.000000	1.000000	1.0000

200 rows × 100 columns

3. Logistic regression with a high dimensional feature function

- 필요 함수들을 선언하겠습니다.

In [382]:

```
def logistic(x):
    return 1 / (1 + math.e ** -x)
```

In [383]:

```
def check_accuracy(result):
    match = 0
    for i in range(len(result)):
        if result[i] < 0.5 and label[i] == 0:
            match += 1
        if result[i] >= 0.5 and label[i] == 1:
            match += 1
    return match/len(label)
```

In [443]:

```
def function_f(x, y, k):
    return (x ** int(k/10)) * (y ** int(k%10))
```

4. Hyper-parameter

In [488]:

```
ephochs = 5000
m = len(label)
learning_rate = 0.1
train_new_data_array = np.array(train_new_data)

lamda_list = [0.00001, 0.0001, 0.001, 0.01, 0.1]
training_loss_list = []
testing_loss_list = []
train_accuracy_list = []
test_accuracy_list = []
coef_list = []
```

5. Training

In [489]:

```
for idx, lamda in enumerate(lamda_list):
    training_loss = []
    testing_loss = []
    train_accuracy = []
    test_accuracy = []
    coef = np.zeros(100)

    for i in range(ephochs):
        if i % 1000 == 0:
            print(str(i) + " times running..")
        h = np.array(logistic(train_new_data.dot(coef)))
        test_h = np.array(logistic(test_new_data.dot(coef)))

        train_loss = - (1/m) * np.sum(label * np.log(h) + (1 - label) * np.log(1 - h)) + lamda * (t0 ** 2 + np.sum(coef ** 2))
        test_loss = - (1/m) * np.sum(test_label * np.log(test_h) + (1 - test_label) * np.log(1 - test_h)) + lamda * (t0 ** 2 + np.sum(coef *
* 2))

        training_loss.append(train_loss)
        testing_loss.append(test_loss)
        train_accuracy.append(check_accuracy(h))
        test_accuracy.append(check_accuracy(test_h))
        for i in range(len(coef)):
            coef[i] = coef[i] - learning_rate * (np.sum((h - label) * train_new_data_array[:,i] / m) + lamda * coef[i])

    ## 결과 저장
    coef_list.append(coef)
    testing_loss_list.append(testing_loss)
    training_loss_list.append(training_loss)
    train_accuracy_list.append(train_accuracy[-1])
    test_accuracy_list.append(test_accuracy[-1])
```

/

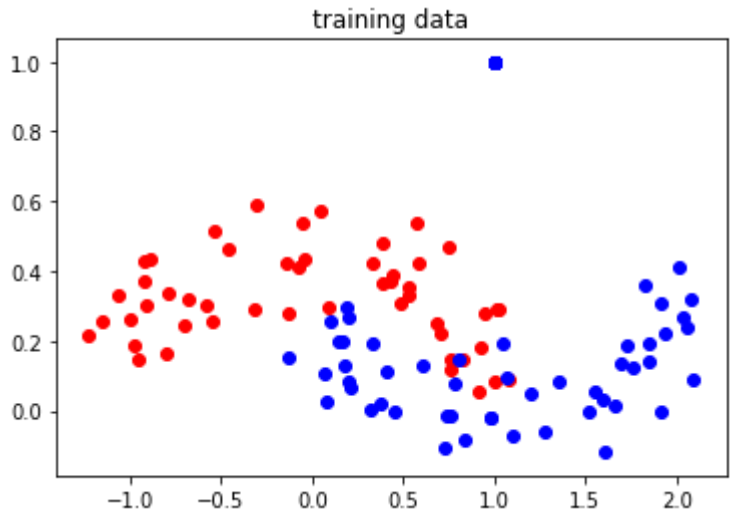
```
0 times running..

C:\Users\alber\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: RuntimeWarning: divide by zero encountered in log

C:\Users\alber\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWarning: divide by zero encountered in log
  from ipykernel import kernelapp as app

1000 times running..
2000 times running..
3000 times running..
4000 times running..
0 times running..
1000 times running..
2000 times running..
3000 times running..
4000 times running..
0 times running..
1000 times running..
2000 times running..
3000 times running..
4000 times running..
0 times running..
1000 times running..
2000 times running..
3000 times running..
4000 times running..
0 times running..
1000 times running..
2000 times running..
3000 times running..
4000 times running..
```

[Output]

1. Plot the training data

In [490]:

```
plt.scatter(dataX_0, dataY_0, color ='r')
plt.scatter(dataX_1, dataY_1, color ='b')
plt.title('training data')
plt.show()
```

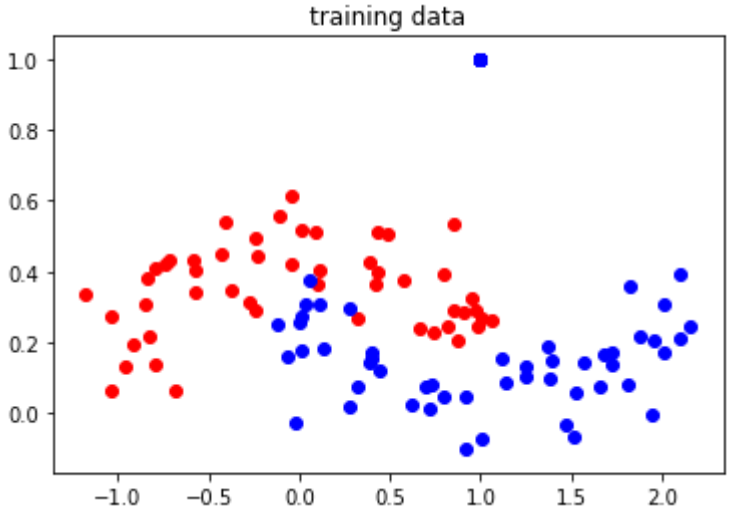


2. Plot the testing data

/

In [491]:

```
plt.scatter(test_dataX_0, test_dataY_0, color ='r')
plt.scatter(test_dataX_1, test_dataY_1, color ='b')
plt.title('training data')
plt.show()
```

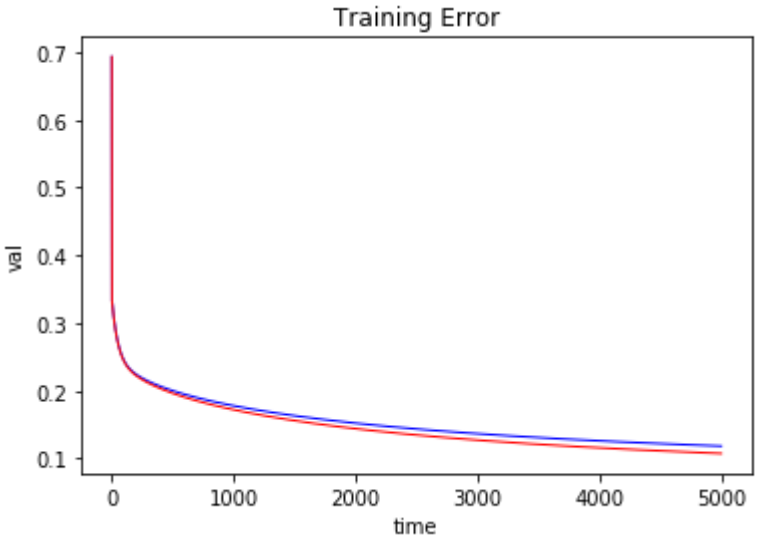


3. Plot the learning curve with λ=0.00001

/

In [493]:

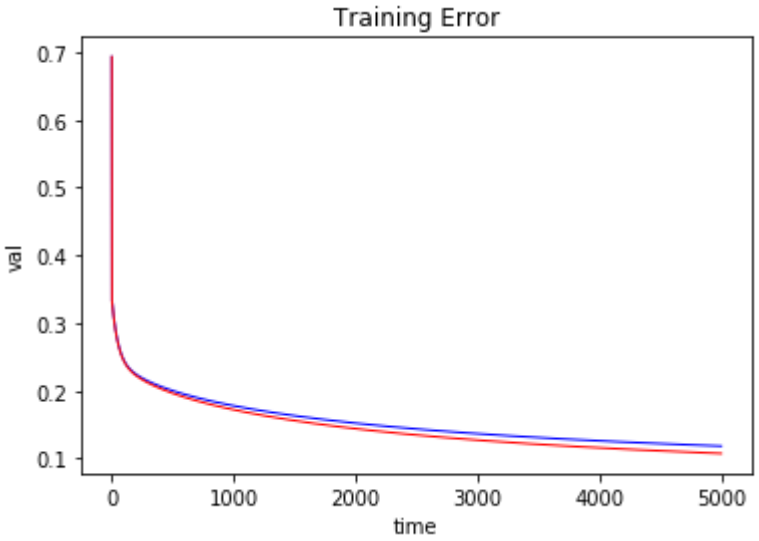
```
plt.plot([i for i in range(len(training_loss_list[0]))], training_loss_list[0], c = 'b', linewidth=1)
plt.plot([i for i in range(len(testing_loss_list[0]))], testing_loss_list[0], c = 'r', linewidth=1)
plt.title('Training Error')
plt.xlabel('time')
plt.ylabel('val')
plt.show()
```



4. Plot the learning curve with λ=0.0001

In [494]:

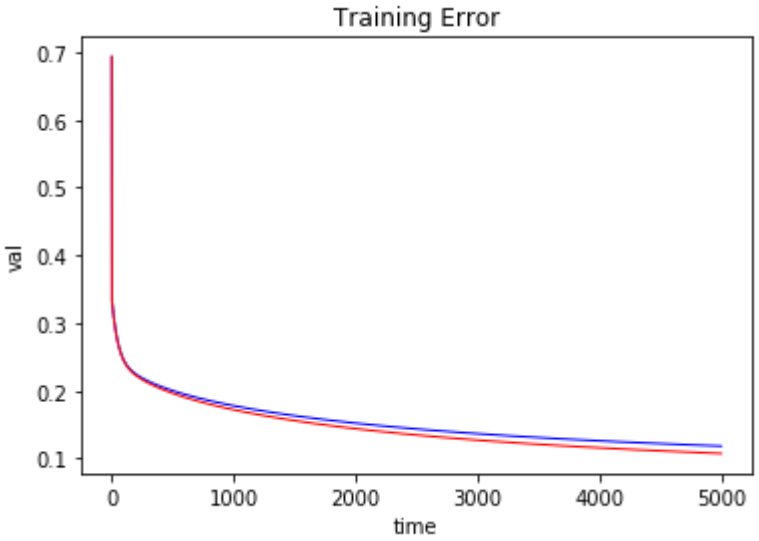
```
plt.plot([i for i in range(len(training_loss_list[1]))], training_loss_list[0], c = 'b', linewidth=1)
plt.plot([i for i in range(len(testing_loss_list[1]))], testing_loss_list[0], c = 'r', linewidth=1)
plt.title('Training Error')
plt.xlabel('time')
plt.ylabel('val')
plt.show()
```



5. Plot the learning curve with λ=0.001

In [495]:

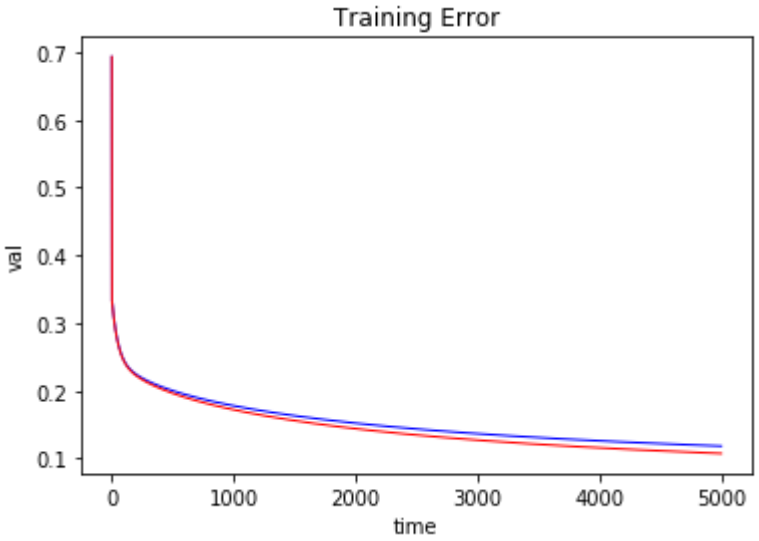
```
plt.plot([i for i in range(len(training_loss_list[2]))], training_loss_list[0], c = 'b', linewidth=1)
plt.plot([i for i in range(len(testing_loss_list[2]))], testing_loss_list[0], c = 'r', linewidth=1)
plt.title('Training Error')
plt.xlabel('time')
plt.ylabel('val')
plt.show()
```



6. Plot the learning curve with λ=0.01

In [496]:

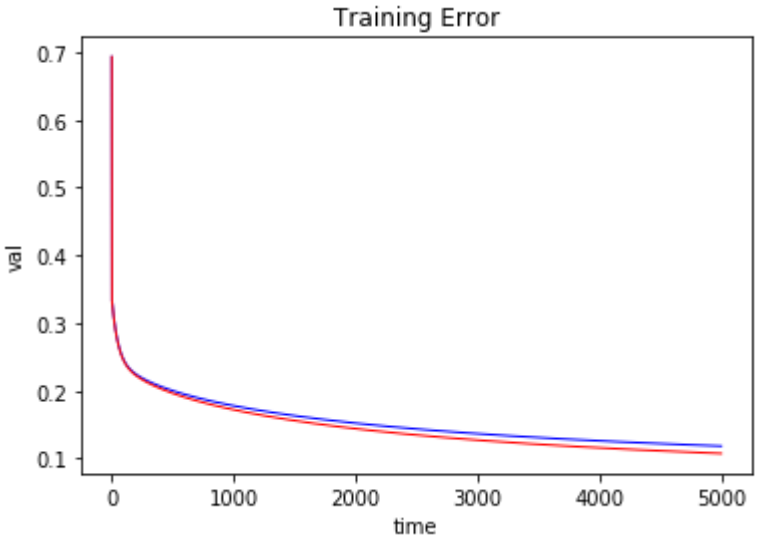
```
plt.plot([i for i in range(len(training_loss_list[3]))], training_loss_list[0], c = 'b', linewidth=1)
plt.plot([i for i in range(len(testing_loss_list[3]))], testing_loss_list[0], c = 'r', linewidth=1)
plt.title('Training Error')
plt.xlabel('time')
plt.ylabel('val')
plt.show()
```



7. Plot the learning curve with λ=0.1

In [497]:

```
plt.plot([i for i in range(len(training_loss_list[4]))], training_loss_list[0], c = 'b', linewidth=1)
plt.plot([i for i in range(len(testing_loss_list[4]))], testing_loss_list[0], c = 'r', linewidth=1)
plt.title('Training Error')
plt.xlabel('time')
plt.ylabel('val')
plt.show()
```



8. Plot the probability map of the obtained classifier with λ=0.00001

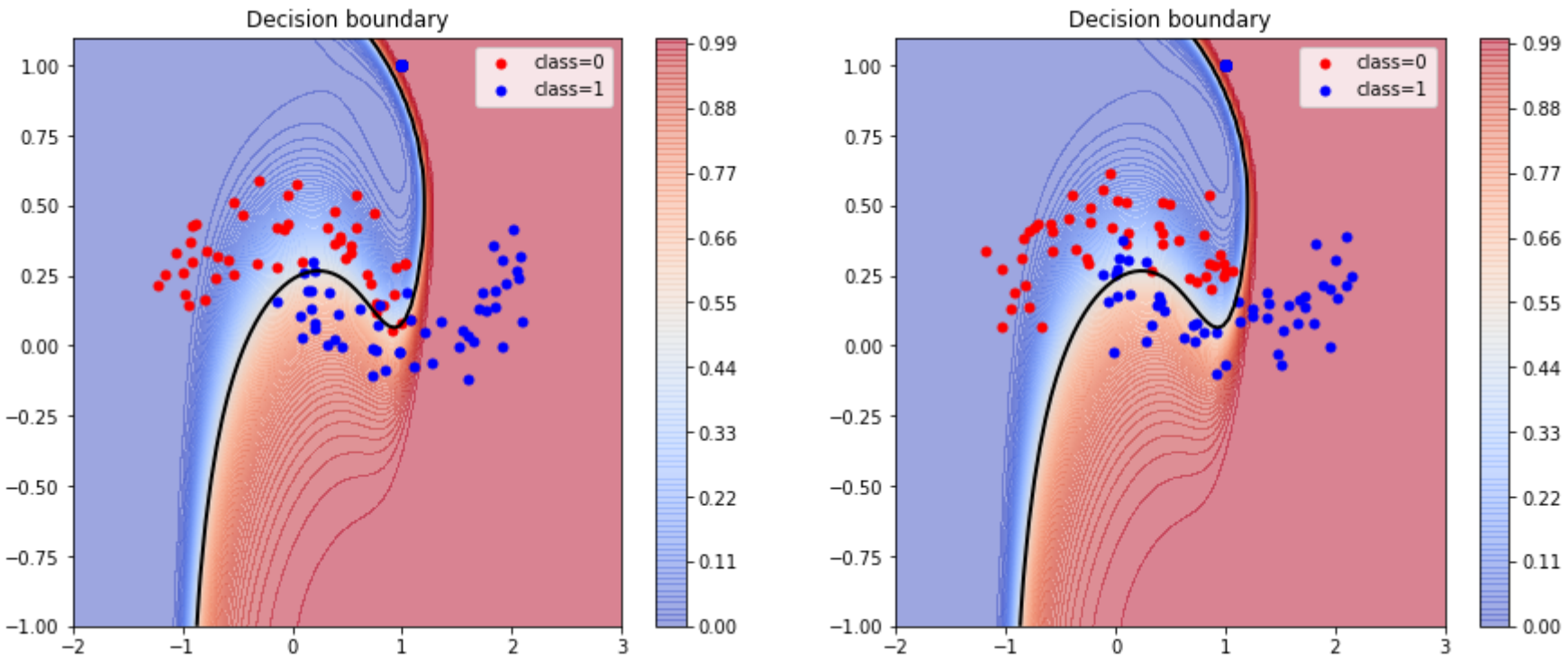
In [550]:

```
x1_coordinate = np.linspace(-2, 3, 100)
x2_coordinate = np.linspace(-1 , 1.1, 100)

xx1, xx2 = np.meshgrid(x1_coordinate, x2_coordinate) # create meshgrid
X2 = np.zeros((len(xx1), len(xx2)))

for i in range(len(x1_coordinate)):
    for j in range(len(x2_coordinate)):
        sum = 0
        for k in range(len(coef_list[0])):
            sum += coef_list[0][k] * function_f(x1_coordinate[i], x2_coordinate[j], k)
        X2[i,j] = logistic(sum)

fig = plt.figure(figsize=(15,6))
ax1 = plt.subplot(121)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(dataX_0, dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(dataX_1, dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('Decision boundary')

# right
ax1 = plt.subplot(122)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(test_dataX_0, test_dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(test_dataX_1, test_dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('Decision boundary')
```

/

Out[550]:

```
Text(0.5, 1.0, 'Decision boundary')
```



9. Plot the probability map of the obtained classifier with λ=0.0001

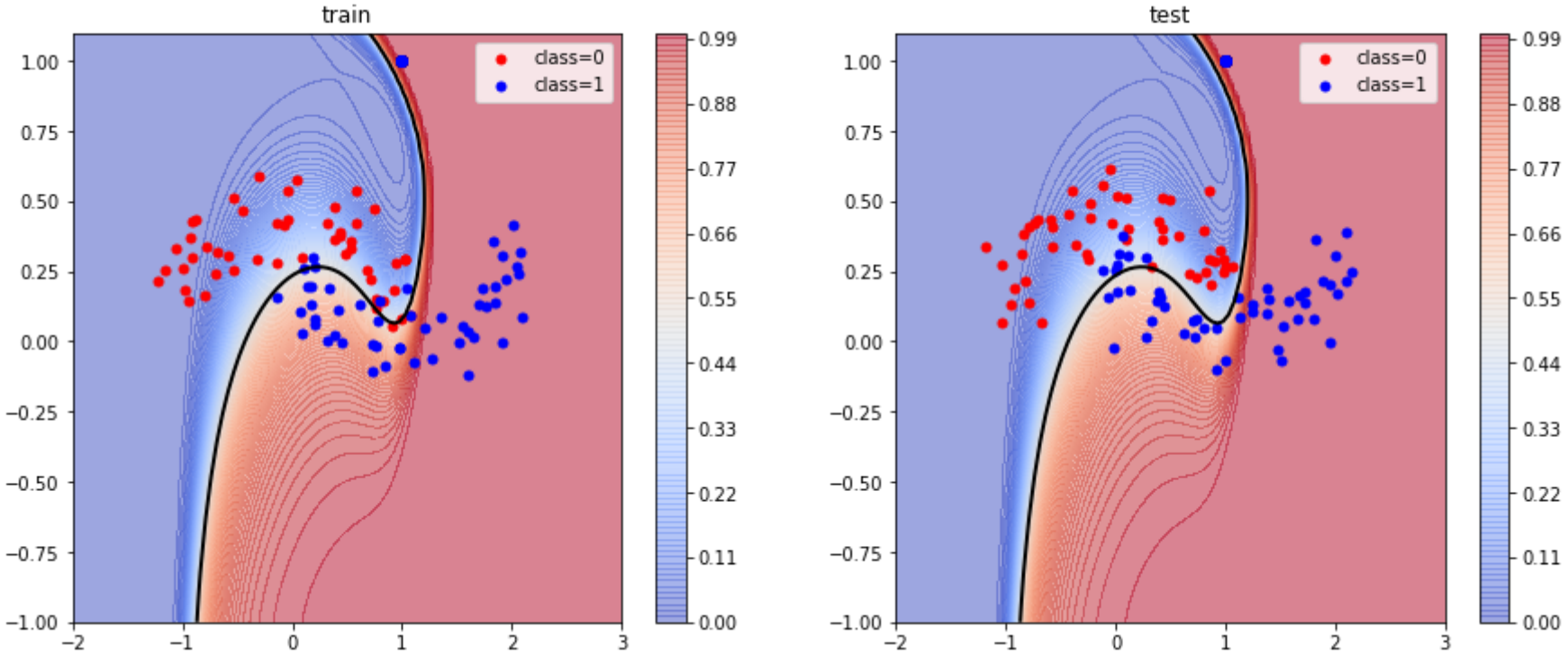
In [551]:

```
x1_coordinate = np.linspace(-2, 3, 100)
x2_coordinate = np.linspace(-1 , 1.1, 100)

xx1, xx2 = np.meshgrid(x1_coordinate, x2_coordinate) # create meshgrid
X2 = np.zeros((len(xx1), len(xx2)))

for i in range(len(x1_coordinate)):
    for j in range(len(x2_coordinate)):
        sum = 0
        for k in range(len(coef_list[1])):
            sum += coef_list[1][k] * function_f(x1_coordinate[i], x2_coordinate[j], k)
        X2[i,j] = logistic(sum)

fig = plt.figure(figsize=(15,6))
ax1 = plt.subplot(121)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(dataX_0, dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(dataX_1, dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('train')

# right
ax1 = plt.subplot(122)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(test_dataX_0, test_dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(test_dataX_1, test_dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('test')
```

/

Out[551]:

```
Text(0.5, 1.0, 'test')
```



10. Plot the probability map of the obtained classifier with λ=0.001

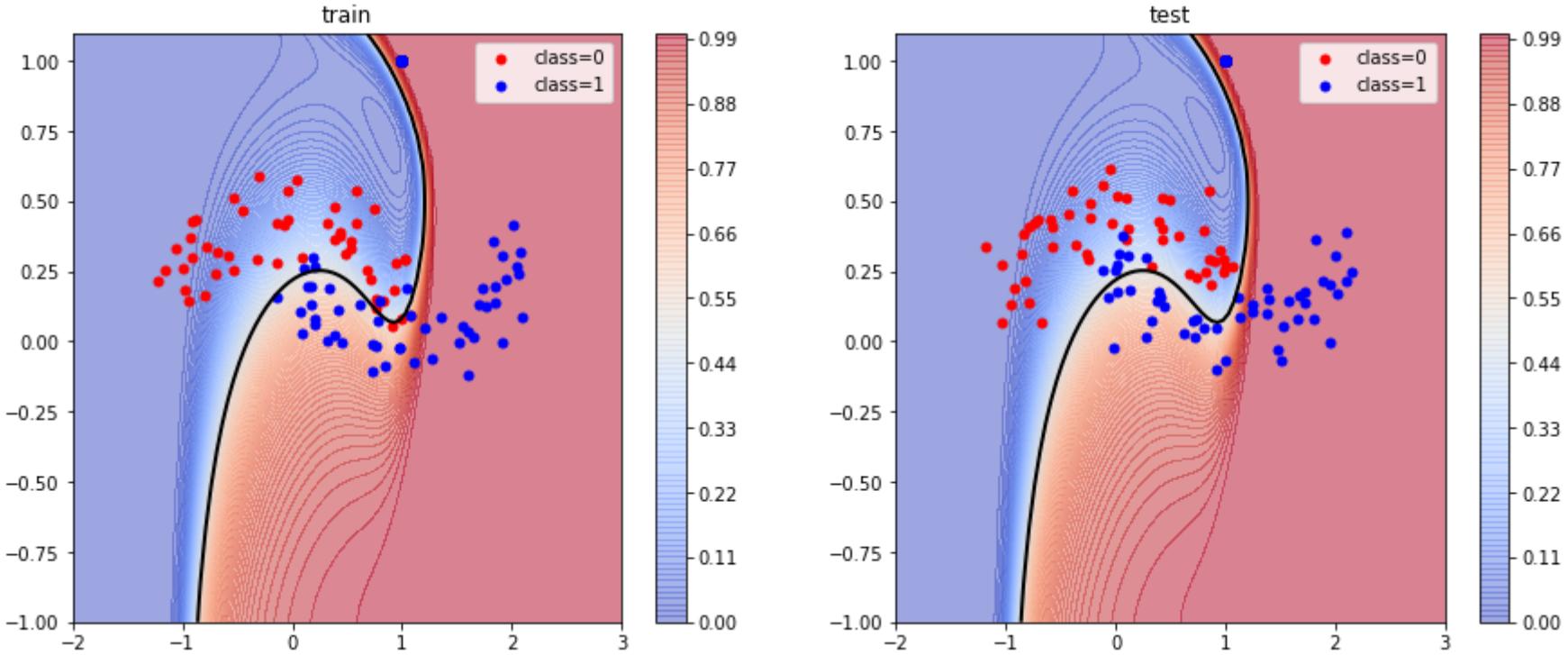
In [552]:

```
x1_coordinate = np.linspace(-2, 3, 100)
x2_coordinate = np.linspace(-1 , 1.1, 100)

xx1, xx2 = np.meshgrid(x1_coordinate, x2_coordinate) # create meshgrid
X2 = np.zeros((len(xx1), len(xx2)))

for i in range(len(x1_coordinate)):
    for j in range(len(x2_coordinate)):
        sum = 0
        for k in range(len(coef_list[2])):
            sum += coef_list[2][k] * function_f(x1_coordinate[i], x2_coordinate[j], k)
        X2[i,j] = logistic(sum)

fig = plt.figure(figsize=(15,6))
ax1 = plt.subplot(121)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(dataX_0, dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(dataX_1, dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('train')

# right
ax1 = plt.subplot(122)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(test_dataX_0, test_dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(test_dataX_1, test_dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('test')
```

/

Out[552]:

```
Text(0.5, 1.0, 'test')
```



11. Plot the probability map of the obtained classifier with λ=0.01

/

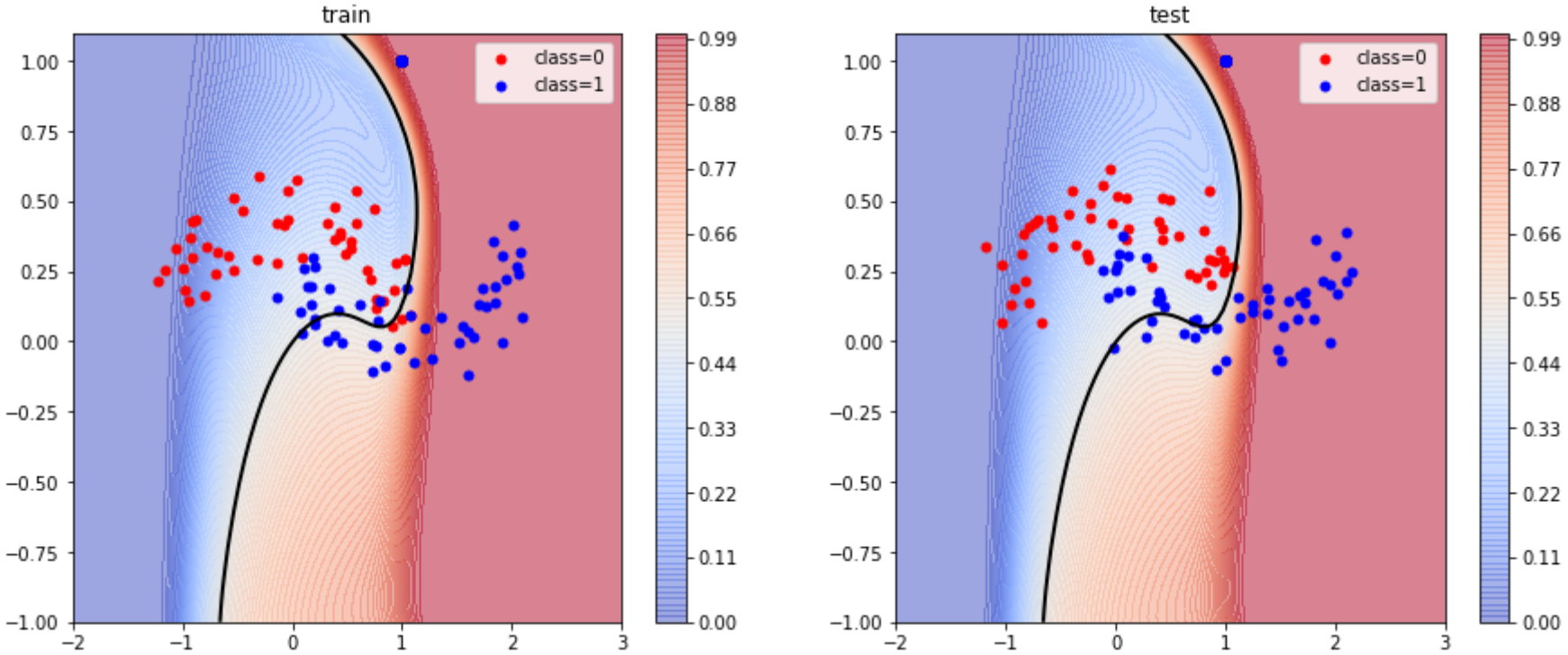
In [553]:

```
x1_coordinate = np.linspace(-2, 3, 100)
x2_coordinate = np.linspace(-1 , 1.1, 100)

xx1, xx2 = np.meshgrid(x1_coordinate, x2_coordinate) # create meshgrid
X2 = np.zeros((len(xx1), len(xx2)))

for i in range(len(x1_coordinate)):
    for j in range(len(x2_coordinate)):
        sum = 0
        for k in range(len(coef_list[3])):
            sum += coef_list[3][k] * function_f(x1_coordinate[i], x2_coordinate[j], k)
        X2[i,j] = logistic(sum)

fig = plt.figure(figsize=(15,6))
ax1 = plt.subplot(121)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(dataX_0, dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(dataX_1, dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('train')

# right
ax1 = plt.subplot(122)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(test_dataX_0, test_dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(test_dataX_1, test_dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('test')
```

/

Out[553]:

```
Text(0.5, 1.0, 'test')
```



12. Plot the probability map of the obtained classifier with λ=0.1

/

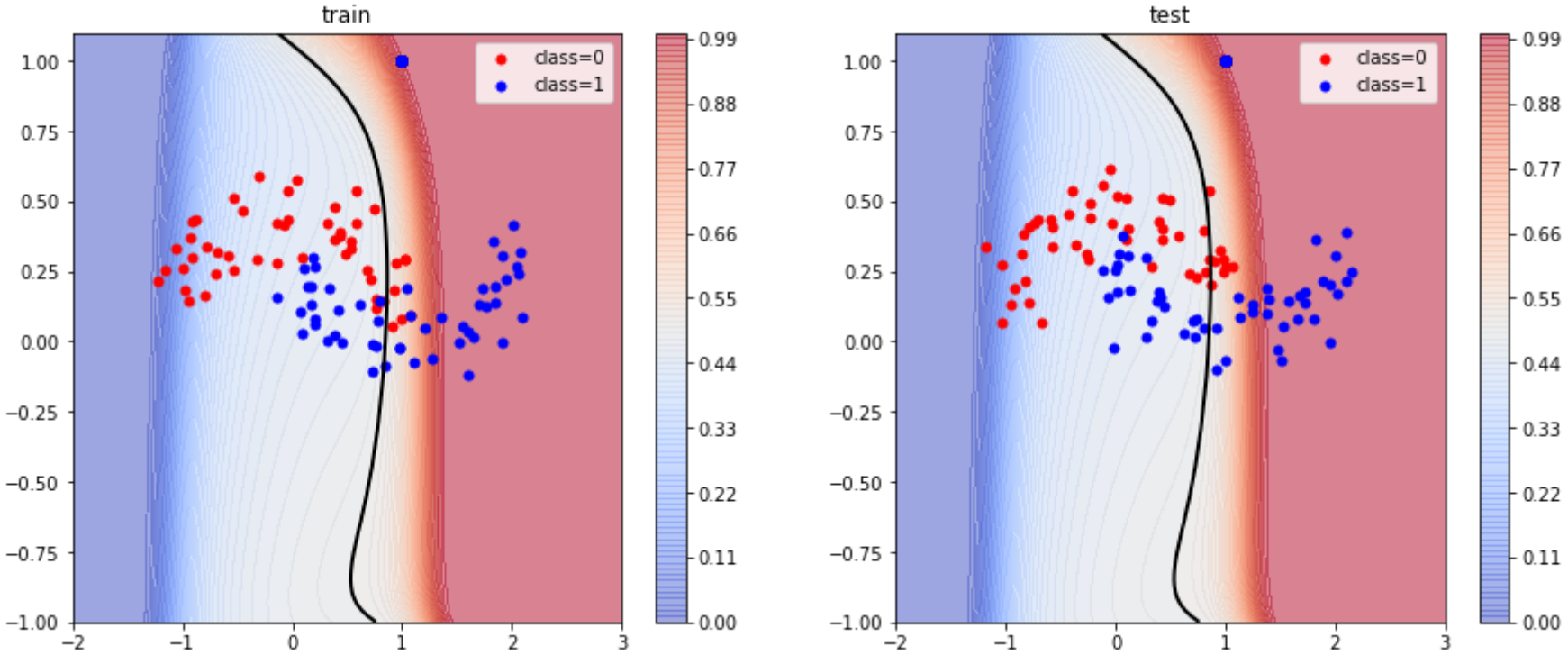
In [554]:

```
x1_coordinate = np.linspace(-2, 3, 100)
x2_coordinate = np.linspace(-1 , 1.1, 100)

xx1, xx2 = np.meshgrid(x1_coordinate, x2_coordinate) # create meshgrid
X2 = np.zeros((len(xx1), len(xx2)))

for i in range(len(x1_coordinate)):
    for j in range(len(x2_coordinate)):
        sum = 0
        for k in range(len(coef_list[4])):
            sum += coef_list[4][k] * function_f(x1_coordinate[i], x2_coordinate[j], k)
        X2[i,j] = logistic(sum)

fig = plt.figure(figsize=(15,6))
ax1 = plt.subplot(121)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(dataX_0, dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(dataX_1, dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('train')

# right
ax1 = plt.subplot(122)

Cs = plt.contourf(xx1, xx2, np.transpose(X2), cmap=cm.coolwarm, alpha=0.5, levels = np.arange(0, 1.01, 0.01))
cbar = plt.colorbar( )
cbar.update_ticks()

plt.scatter(test_dataX_0, test_dataY_0 , s=100, c='r', marker='.', label='class=0')
plt.scatter(test_dataX_1, test_dataY_1, s=100, c='b', marker='.', label='class=1')
plt.contour(xx1, xx2, np.transpose(X2), [0.5], linewidths=2, colors='k')
plt.legend()
plt.title('test')
```

/

Out[554]:

```
Text(0.5, 1.0, 'test')
```



13. Print the final training accuracy with the given regularization parameters

/

In [547]:

```
for i, acc in enumerate(train_accuracy_list):
    print("lamda : {}. Training Accuracy {}%.".format(lamda_list[i], acc))
```

```
lamda : 1e-05. Training Accuracy 0.965%.
lamda : 0.0001. Training Accuracy 0.965%.
lamda : 0.001. Training Accuracy 0.965%.
lamda : 0.01. Training Accuracy 0.91%.
lamda : 0.1. Training Accuracy 0.855%.
```

14. Print the final testing accuracy with the given regularization parameters

In [555]:

```
for i, acc in enumerate(test_accuracy_list):
    print("lamda : {}. Testing Accuracy {}%.".format(lamda_list[i], acc))
```

```
lamda : 1e-05. Testing Accuracy 0.965%.
lamda : 0.0001. Testing Accuracy 0.965%.
lamda : 0.001. Testing Accuracy 0.965%.
lamda : 0.01. Testing Accuracy 0.92%.
lamda : 0.1. Testing Accuracy 0.85%.
```