Code

0. Import library

```
# Import libraries

# math library
import numpy as np
import torch

# visualization library
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png2x','pdf')
import matplotlib.pyplot as plt
```

1. Load and plot the dataset

```
import numpy as np
import matplotlib.pyplot as plt
# data path
test_path = '/content/drive/My Drive/ML_Assignment/data/testing.txt'
train_path = '/content/drive/My Drive/ML_Assignment/data/training.txt'

# import data with numpy
data_train  = np.loadtxt(train_path, delimiter=',')
data_test   = np.loadtxt(test_path, 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 clas
```

```
fig_1 = plt.figure()
plt.scatter(x1_train[idx_class0_train], x2_train[idx_class0_train], s=50, c='r', marker='.', label='label = 0')
plt.scatter(x1_train[idx_class1_train], x2_train[idx_class1_train], s=50, c='b', marker='.', label='label = 1')
plt.title('Training data')
plt.legend()
plt.show()
fig_1.savefig('Training data.png')
```

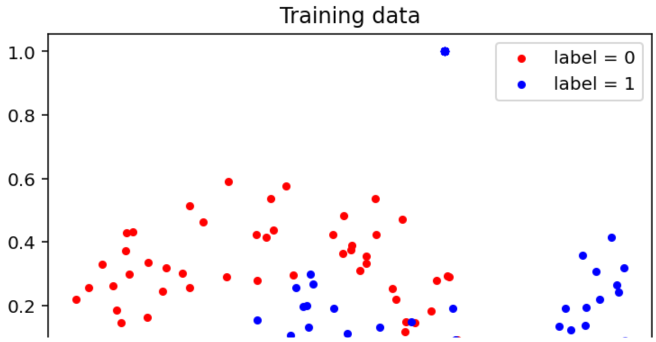
```
# gradient function definition
def grad_loss(y_pred,y,X):
    n = len(y)
    grad = torch.sum((y_pred - y)* X[:,i].unsqueeze(1))/n
    return grad


# gradient descent function definition
def grad_desc(X_train, X_test, y_train, y_test, tau, max_iter, lamb):
    L_iters_train = np.zeros([max_iter]) # record the loss values
    L_iters_test = np.zeros([max_iter])
    w = np.array([[0.01] for j in range(100)]) # initialization
    w = torch.tensor(w)
    print('================== Running Start ==================')
    for i in range(max_iter): # loop over the iterations
        train_pred = g_func(X_train, w) # linear predicition function
        test_pred = g_func(X_test, w)
        for n in range(0, 100, 1) :
          w[n] = (1 - tau*lamb) * w[n] - tau*torch.sum((train_pred - y_train)* X_train[:,n].unsqueeze(1))/len(y_trai
        J_train = loss_logreg(train_pred,y_train,w,lamb)
        J_test = loss_logreg(test_pred,y_test,w,lamb)
        L_iters_train[i] = J_train # save the current loss value
        L_iters_test[i] = J_test
        if i % 10000 == 0:
          print('Epoch: {:6d}, train_cost: {:10f}, test_cost: {:10f}'.format(i,J_train,J_test))

    return w, L_iters_train, L_iters_test
```

[Change Numpy to Tensor]

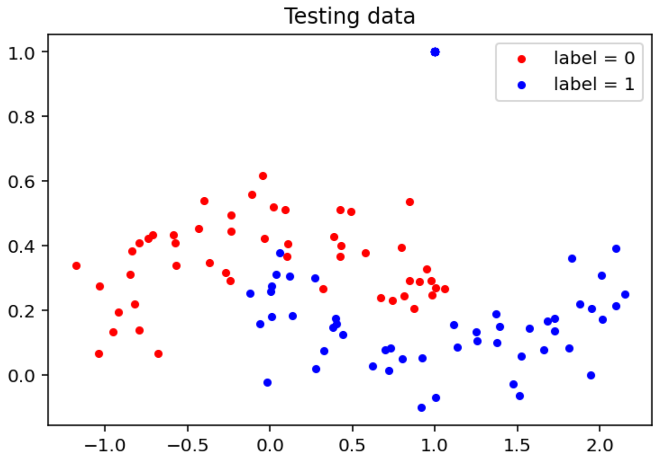
```
train_x1 = torch.DoubleTensor(x1_train)
train_x2 = torch.DoubleTensor(x2_train)
train_label = torch.DoubleTensor(data_train[:,2])

test_x1 = torch.DoubleTensor(x1_test)
test_x2 = torch.DoubleTensor(x2_test)
test_label = torch.DoubleTensor(data_test[:,2])

train_x1 = train_x1.unsqueeze(1)
train_x2 = train_x2.unsqueeze(1)
train_label = train_label.unsqueeze(1)

test_x1 = test_x1.unsqueeze(1)
test_x2 = test_x2.unsqueeze(1)
test_label = test_label.unsqueeze(1)

# construct the data matrix X
X_train = function(train_x1, train_x2)
X_test = function(test_x1, test_x2)
```

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

```
# run gradient descent algorithm
tau = 1e-7; max_iter = 100000;
w_0, train_L_0, test_L_0 = grad_desc(X_train, X_test, train_label, test_label, tau, max_iter, 0.00001)
w_1, train_L_1, test_L_1 = grad_desc(X_train, X_test, train_label, test_label, tau, max_iter, 0.0001)
w_2, train_L_2, test_L_2 = grad_desc(X_train, X_test, train_label, test_label, tau, max_iter, 0.001)
w_3, train_L_3, test_L_3 = grad_desc(X_train, X_test, train_label, test_label, tau, max_iter, 0.01)
w_4, train_L_4, test_L_4 = grad_desc(X_train, X_test, train_label, test_label, tau, max_iter, 0.1)
```




```
fig_2 = plt.figure()
plt.scatter(x1_test[idx_class0_test], x2_test[idx_class0_test], s=50, c='r', marker='.', label='label = 0')
plt.scatter(x1_test[idx_class1_test], x2_test[idx_class1_test], s=50, c='b', marker='.', label='label = 1')
plt.title('Testing data')
plt.legend()
plt.show()
fig_2.savefig('Testing data.png')
```

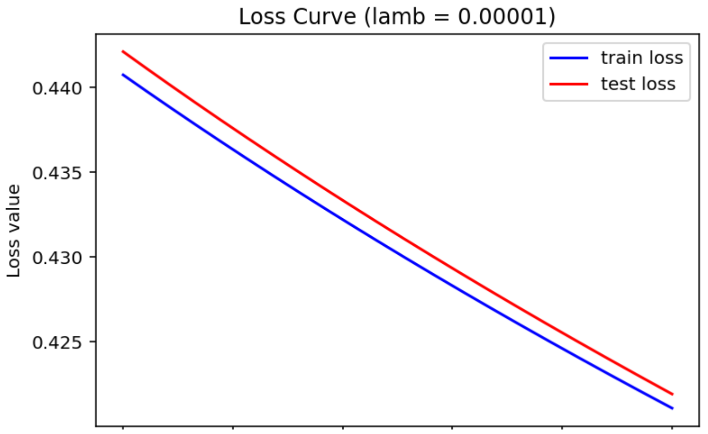


2. Define a logistic regression loss function and its gradient

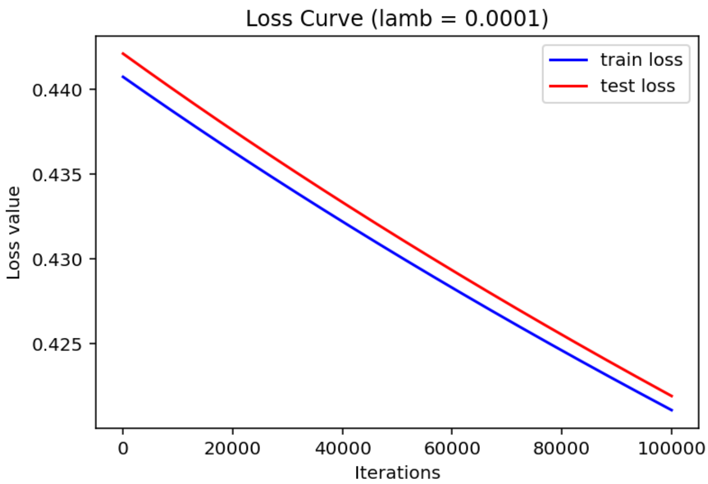
```
# make X
def function(x, y):
  func_list = []
  for i in range (0, 10, 1):
    for j in range (0, 10, 1):
      func_list.append((x**i)*(y**j))
  func_list = torch.stack(func_list).squeeze(-1)
  return func_list.T

# g(x, y, w)
def g_func(X, w):
  pred = sigmoid(torch.matmul(X, w))
  return pred

# sigmoid function
def sigmoid(z):
    sigmoid_f = 1.0 / (1.0 + torch.exp(-z))
    return sigmoid_f

# loss function definition
def loss_logreg(y_pred,y,w,lamb):
    n = len(y)
    cross_entropy = torch.sum(-y * torch.log(y_pred) - (1 - y) *  torch.log(1 - y_pred)) / n
    r = torch.sum(w**2) * lamb * 0.5
    loss = cross_entropy + r
    return loss

```

```
=================== Running Start ===================
Epoch:      0, train_cost:   0.440711, test_cost:   0.442080
Epoch:  10000, train_cost:   0.438475, test_cost:   0.439784
Epoch:  20000, train_cost:   0.436311, test_cost:   0.437562
Epoch:  30000, train_cost:   0.434214, test_cost:   0.435408
Epoch:  40000, train_cost:   0.432178, test_cost:   0.433316
Epoch:  50000, train_cost:   0.430200, test_cost:   0.431284
Epoch:  60000, train_cost:   0.428276, test_cost:   0.429307
Epoch:  70000, train_cost:   0.426403, test_cost:   0.427383
Epoch:  80000, train_cost:   0.424578, test_cost:   0.425507
Epoch:  90000, train_cost:   0.422799, test_cost:   0.423679
=================== Running Start ===================
Epoch:      0, train_cost:   0.440711, test_cost:   0.442080
Epoch:  10000, train_cost:   0.438476, test_cost:   0.439785
Epoch:  20000, train_cost:   0.436312, test_cost:   0.437563
Epoch:  30000, train_cost:   0.434215, test_cost:   0.435408
Epoch:  40000, train_cost:   0.432179, test_cost:   0.433317
Epoch:  50000, train_cost:   0.430201, test_cost:   0.431285
Epoch:  60000, train_cost:   0.428277, test_cost:   0.429308
Epoch:  70000, train_cost:   0.426404, test_cost:   0.427383
Epoch:  80000, train_cost:   0.424579, test_cost:   0.425508
Epoch:  90000, train_cost:   0.422800, test_cost:   0.423680
=================== Running Start ===================
Epoch:      0, train_cost:   0.440716, test_cost:   0.442085
Epoch:  10000, train_cost:   0.438480, test_cost:   0.439790
Epoch:  20000, train_cost:   0.436317, test_cost:   0.437568
Epoch:  30000, train_cost:   0.434220, test_cost:   0.435414
Epoch:  40000, train_cost:   0.432184, test_cost:   0.433323
Epoch:  50000, train_cost:   0.430207, test_cost:   0.431290
Epoch:  60000, train_cost:   0.428283, test_cost:   0.429314
Epoch:  70000, train_cost:   0.426410, test_cost:   0.427389
Epoch:  80000, train_cost:   0.424586, test_cost:   0.425515
Epoch:  90000, train_cost:   0.422807, test_cost:   0.423687
=================== Running Start ===================
Epoch:      0, train_cost:   0.440761, test_cost:   0.442130
Epoch:  10000, train_cost:   0.438528, test_cost:   0.439837
Epoch:  20000, train_cost:   0.436367, test_cost:   0.437618
Epoch:  30000, train_cost:   0.434272, test_cost:   0.435466
Epoch:  40000, train_cost:   0.432240, test_cost:   0.433378
Epoch:  50000, train_cost:   0.430264, test_cost:   0.431348
Epoch:  60000, train_cost:   0.428343, test_cost:   0.429374
Epoch:  70000, train_cost:   0.426473, test_cost:   0.427452
Epoch:  80000, train_cost:   0.424650, test_cost:   0.425579
Epoch:  90000, train_cost:   0.422874, test_cost:   0.423754
=================== Running Start ===================
Epoch:      0, train_cost:   0.441211, test_cost:   0.442580
Epoch:  10000, train_cost:   0.439003, test_cost:   0.440313
Epoch:  20000, train_cost:   0.436868, test_cost:   0.438119
Epoch:  30000, train_cost:   0.434798, test_cost:   0.435993
Epoch:  40000, train_cost:   0.432790, test_cost:   0.433929
Epoch:  50000, train_cost:   0.430839, test_cost:   0.431924
Epoch:  60000, train_cost:   0.428942, test_cost:   0.429974
Epoch:  70000, train_cost:   0.427096, test_cost:   0.428077
Epoch:  80000, train_cost:   0.425298, test_cost:   0.426228
```

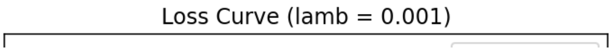
```
# plot
fig_3 = plt.figure()
plt.plot(np.array(range(max_iter)), train_L_0, c = 'b', label='train loss')
plt.plot(np.array(range(max_iter)), test_L_0, c = 'r', label='test loss')
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.title('Loss Curve (lamb = 0.00001)')
plt.legend()
plt.show()
fig_3.savefig('Loss(lamb = 0.00001).png')
```



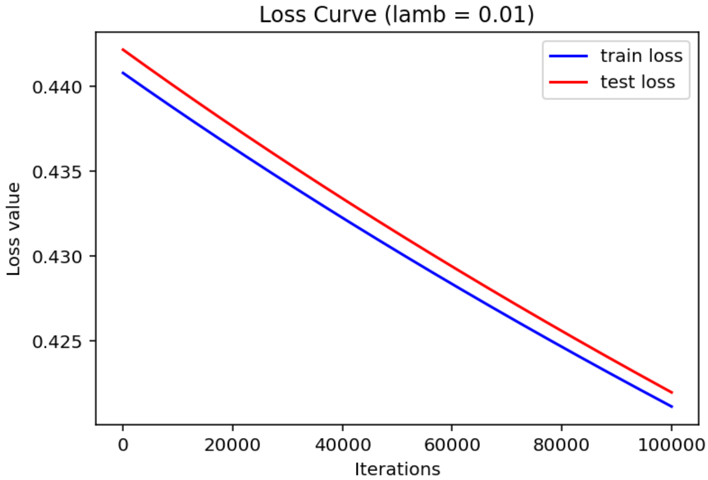

```
# plot
fig_4 = plt.figure()
plt.plot(np.array(range(max_iter)), train_L_1, c = 'b', label='train loss')
plt.plot(np.array(range(max_iter)), test_L_1, c = 'r', label='test loss')
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.title('Loss Curve (lamb = 0.0001)')
plt.legend()
plt.show()
fig_4.savefig('Loss(lamb = 0.0001).png')
```



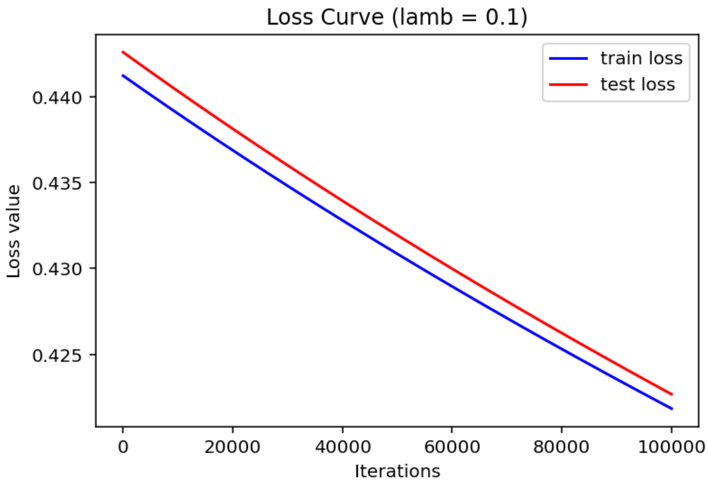

```
# plot
fig_5 = plt.figure()
plt.plot(np.array(range(max_iter)), train_L_2, c = 'b', label='train loss')
plt.plot(np.array(range(max_iter)), test_L_2, c = 'r', label='test loss')
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.title('Loss Curve (lamb = 0.001)')
plt.legend()
plt.show()
fig_5.savefig('Loss(lamb = 0.001).png')
```




```
# plot
fig_6 = plt.figure()
plt.plot(np.array(range(max_iter)), train_L_3, c = 'b', label='train loss')
plt.plot(np.array(range(max_iter)), test_L_3, c = 'r', label='test loss')
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.title('Loss Curve (lamb = 0.01)')
plt.legend()
plt.show()
fig_6.savefig('Loss(lamb = 0.01).png')
```




```
# plot
fig_7 = plt.figure()
plt.plot(np.array(range(max_iter)), train_L_4, c = 'b', label='train loss')
plt.plot(np.array(range(max_iter)), test_L_4, c = 'r', label='test loss')
plt.xlabel('Iterations')
plt.ylabel('Loss value')
plt.title('Loss Curve (lamb = 0.1)')
plt.legend()
plt.show()
fig_7.savefig('Loss(lamb = 0.1).png')
```



4. Plot the decisoin boundary with probability map

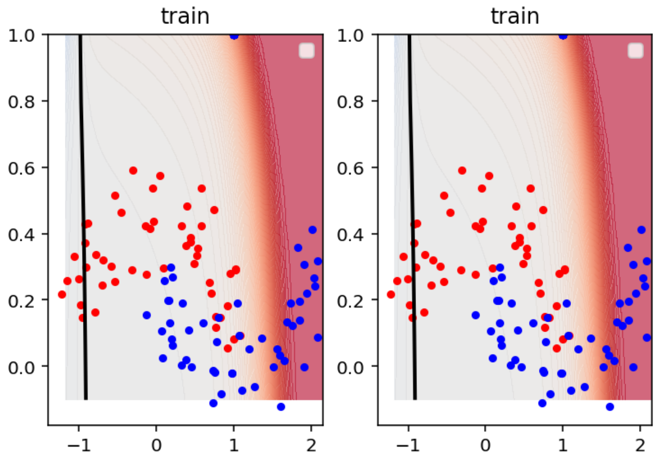
```
# compute values p(x) for multiple data points x
x1_min, x1_max = x1_test.min(), x1_test.max() # min and max of grade 1
x2_min, x2_max = x2_test.min(), x2_test.max() # min and max of grade 2
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_max)) # create meshgrid
xx1 = torch.DoubleTensor(xx1)
```

```
xx2 = torch.DoubleTensor(xx2)
X2 = function(xx1.reshape(-1), xx2.reshape(-1))
```

```
def pic_train_commons(ax, xx1, xx2, p):
  ax.legend(loc = 'upper right')
  ax.set_title('train')
  ax.contourf(xx1,xx2,p,100,vmin=0,vmax=1,cmap='coolwarm', alpha=0.6)
  ax.contour(xx1, xx2, p, [0.5], linewidths=2, colors='k')
  ax.scatter(x1_train[idx_class0_train], x2_train[idx_class0_train], s=50, c='r', marker='.', label='label = 0')
  ax.scatter(x1_train[idx_class1_train], x2_train[idx_class1_train], s=50, c='b', marker='.', label='label = 1')

def pic_test_commons(ax, xx1, xx2, p):
  ax.legend(loc = 'upper right')
  ax.set_title('test')
  ax.contourf(xx1,xx2,p,100,vmin=0,vmax=1,cmap='coolwarm', alpha=0.6)
  ax.contour(xx1, xx2, p, [0.5], linewidths=2, colors='k')
  ax.scatter(x1_test[idx_class0_train], x2_test[idx_class0_train], s=50, c='r', marker='.', label='label = 0')
  ax.scatter(x1_test[idx_class1_train], x2_test[idx_class1_train], s=50, c='b', marker='.', label='label = 1')
```

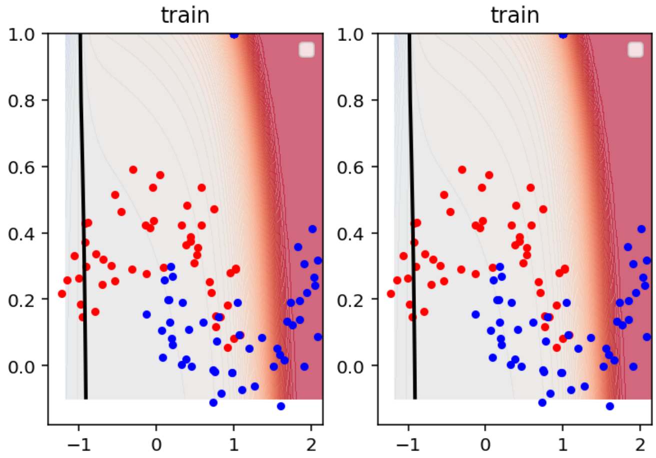
```
p = g_func(X2,w_0)
p = p.reshape(50,50)
p = p.numpy()

# plot
fig_8 = plt.figure()
fig_8, ax = plt. subplots(1,2)
plt.title('lambda = 0.00001')
pic_train_commons(ax[0], xx1, xx2, p)
pic_train_commons(ax[1], xx1, xx2, p)
plt.show()
fig_8.savefig('Probability Map (Lambda = 0.00001).png')
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
<Figure size 432x288 with 0 Axes>
```



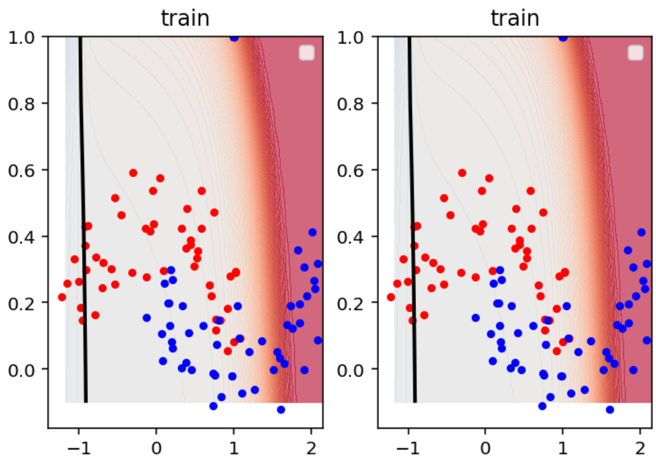

```
p = g_func(X2,w_1)
p = p.reshape(50,50)
p = p.numpy()

# plot
fig_9 = plt.figure()
fig_9, ax = plt. subplots(1,2)
plt.title('lambda = 0.0001')
pic_train_commons(ax[0], xx1, xx2, p)
pic_train_commons(ax[1], xx1, xx2, p)
plt.show()
fig_9.savefig('Probability Map (Lambda = 0.0001).png')
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
<Figure size 432x288 with 0 Axes>
```



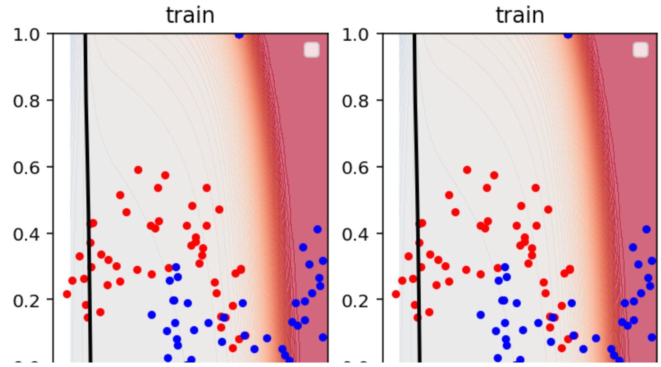

```python
p = g_func(X2,w_2)
p = p.reshape(50,50)
p = p.numpy()

# plot
fig_10 = plt.figure()
fig_10, ax = plt. subplots(1,2)
plt.title('lambda = 0.001')
pic_train_commons(ax[0], xx1, xx2, p)
pic_train_commons(ax[1], xx1, xx2, p)
plt.show()
fig_10.savefig('Probability Map (Lambda = 0.001).png')
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
<Figure size 432x288 with 0 Axes>
```



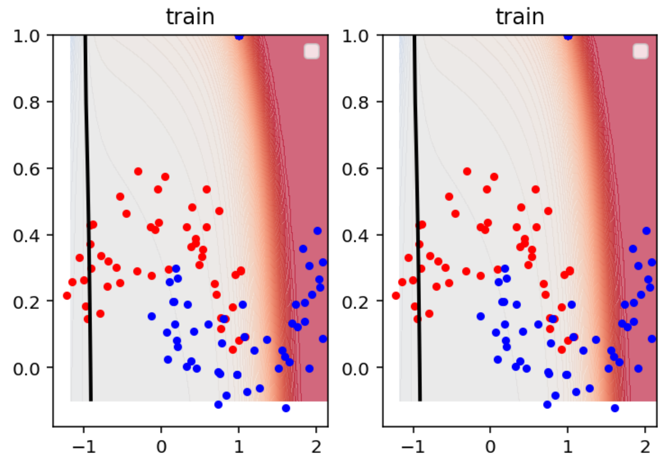

```python
p = g_func(X2,w_3)
p = p.reshape(50,50)
p = p.numpy()

# plot
fig_11 = plt.figure()
fig_11, ax = plt. subplots(1,2)
plt.title('lambda = 0.01')
pic_train_commons(ax[0], xx1, xx2, p)
pic_train_commons(ax[1], xx1, xx2, p)
plt.show()
fig_11.savefig('Probability Map (Lambda = 0.01).png')
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
<Figure size 432x288 with 0 Axes>
```




```
p = g_func(X2,w_4)
p = p.reshape(50,50)
p = p.numpy()

# plot
fig_12 = plt.figure()
fig_12, ax = plt. subplots(1,2)
plt.title('lambda = 0.1')
pic_train_commons(ax[0], xx1, xx2, p)
pic_train_commons(ax[1], xx1, xx2, p)
plt.show()
fig_12.savefig('Probability Map (Lambda = 0.1).png')
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
<Figure size 432x288 with 0 Axes>
```



5. Compute the classification accuracy

```
n = data_train.shape[0]
idx_class0 = (data_train[:,2]==0) # index of class0
idx_class1 = (data_train[:,2]==1) # index of class1

p_train_0 = g_func(X_train, w_0).numpy()
p_train_1 = g_func(X_train, w_1).numpy()
p_train_2 = g_func(X_train, w_2).numpy()
p_train_3 = g_func(X_train, w_3).numpy()
p_train_4 = g_func(X_train, w_4).numpy()

idx_train_0 = (p_train_0 < 0.5)
idx_train_1 = (p_train_1 < 0.5)
idx_train_2 = (p_train_2 < 0.5)
idx_train_3 = (p_train_3 < 0.5)
idx_train_4 = (p_train_4 < 0.5)
```

```
idx_right_0 = (idx_class0 == idx_train_0.reshape(-1))
idx_right_1 = (idx_class0 == idx_train_1.reshape(-1))
idx_right_2 = (idx_class0 == idx_train_2.reshape(-1))
idx_right_3 = (idx_class0 == idx_train_3.reshape(-1))
idx_right_4 = (idx_class0 == idx_train_4.reshape(-1))

acc_train_0 = sum(idx_right_0)/n * 100
acc_train_1 = sum(idx_right_1)/n * 100
acc_train_2 = sum(idx_right_2)/n * 100
acc_train_3 = sum(idx_right_3)/n * 100
acc_train_4 = sum(idx_right_4)/n * 100
acc_train = [acc_train_0, acc_train_1, acc_train_2, acc_train_3, acc_train_4]
```

```
n = data_test.shape[0]
idx_class0 = (data_test[:,2]==0) # index of class0
idx_class1 = (data_test[:,2]==1) # index of class1

p_0 = g_func(X_test, w_0).numpy()
p_1 = g_func(X_test, w_1).numpy()
p_2 = g_func(X_test, w_2).numpy()
p_3 = g_func(X_test, w_3).numpy()
p_4 = g_func(X_test, w_4).numpy()

idx_0 = (p_0 < 0.5)
idx_1 = (p_1 < 0.5)
idx_2 = (p_2 < 0.5)
idx_3 = (p_3 < 0.5)
idx_4 = (p_4 < 0.5)

idx_right_0 = (idx_class0 == idx_0.reshape(-1))
idx_right_1 = (idx_class0 == idx_1.reshape(-1))
idx_right_2 = (idx_class0 == idx_2.reshape(-1))
idx_right_3 = (idx_class0 == idx_3.reshape(-1))
idx_right_4 = (idx_class0 == idx_4.reshape(-1))

acc_test_0 = sum(idx_right_0)/n * 100
acc_test_1 = sum(idx_right_1)/n * 100
acc_test_2 = sum(idx_right_2)/n * 100
acc_test_3 = sum(idx_right_3)/n * 100
acc_test_4 = sum(idx_right_4)/n * 100
acc_test = [acc_test_0, acc_test_1, acc_test_2, acc_test_3, acc_test_4]
```

Data Frame

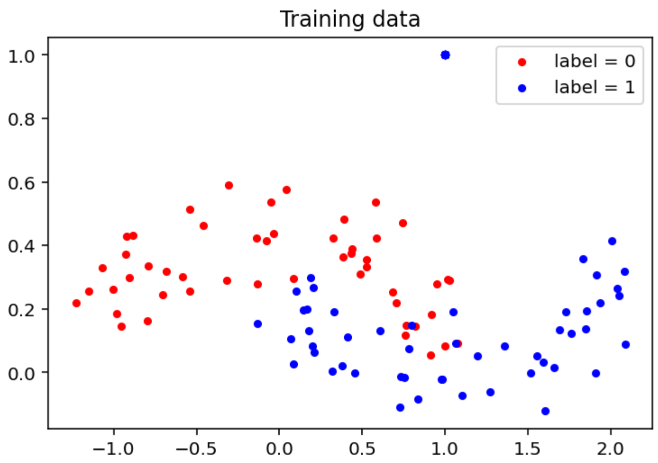
```
from pandas import Series, DataFrame
import pandas as pd

data = {'train_accuracy(%)' : acc_train,
        'test_accuracy(%)' : acc_test}

train_frame = DataFrame (data, columns = ['train_accuracy(%)'],
        index = [0.00001, 0.0001, 0.001, 0.01, 0.1])

test_frame = DataFrame (data, columns = ['test_accuracy(%)'],
        index = [0.00001, 0.0001, 0.001, 0.01, 0.1])
```

```

```

OUTPUT

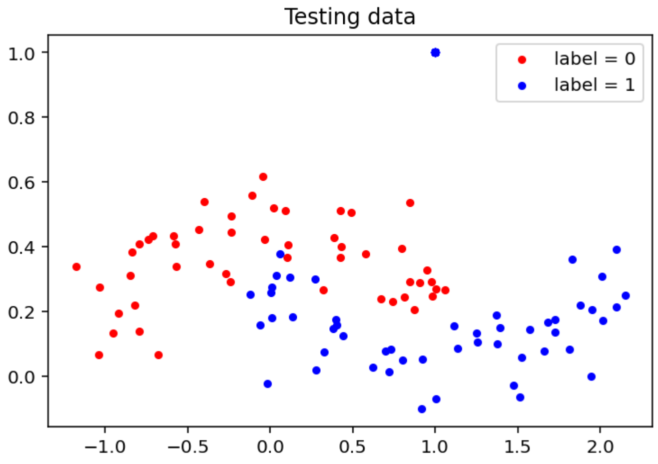
1. Plot the training data [0.5pt]

```
fig_1
```



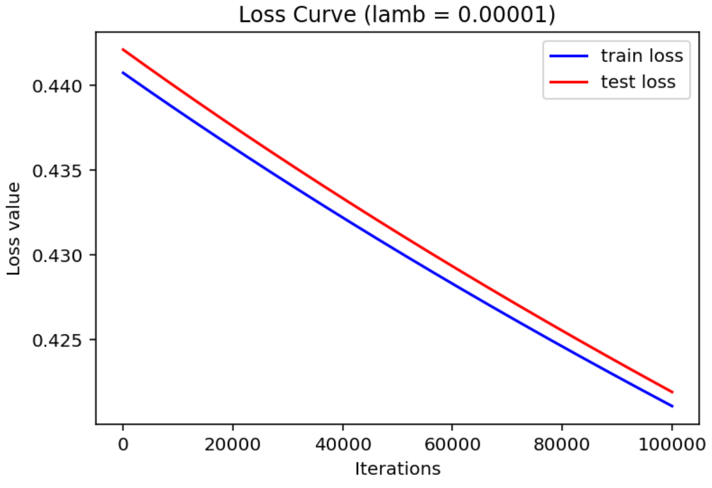
2. Plot the testing data [0.5pt]

```
fig_2
```



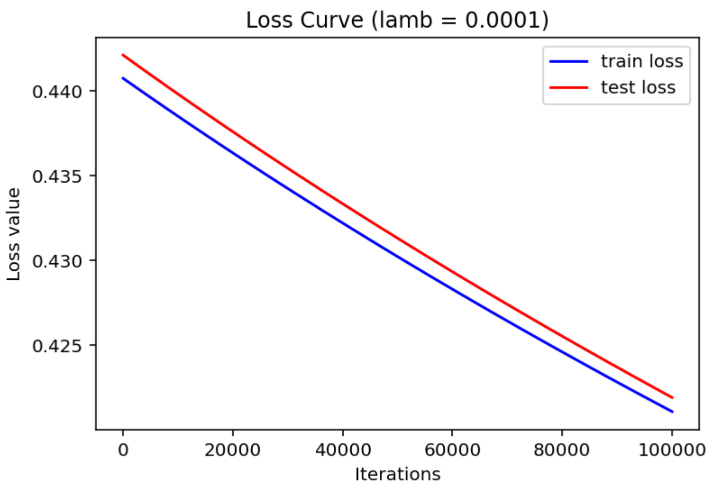
3. Plot the learning curve with λ=0.00001 [1pt]

```
fig_3
```



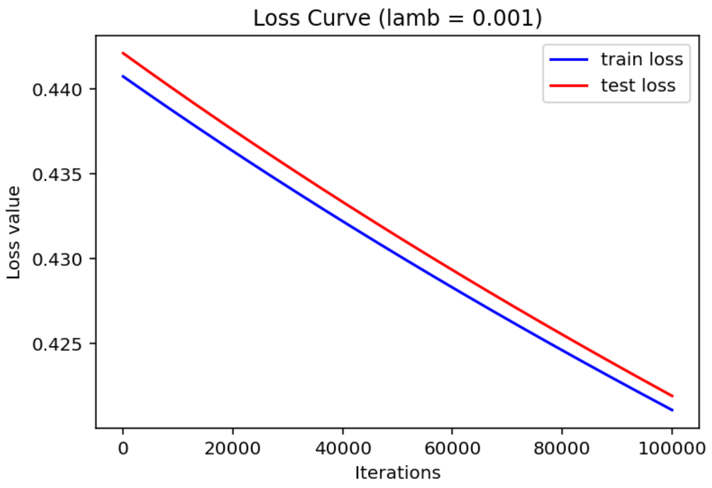
4. Plot the learning curve with λ=0.0001 [1pt]

```
fig_4
```



5. Plot the learning curve with λ=0.001 [1pt]

```
fig_5
```



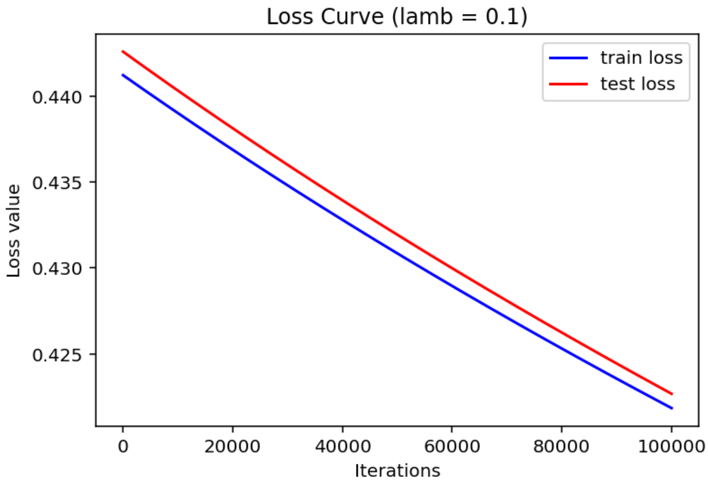
6. Plot the learning curve with λ=0.01 [1pt]

```
fig_6
```

Loss Curve (lamb = 0.01)

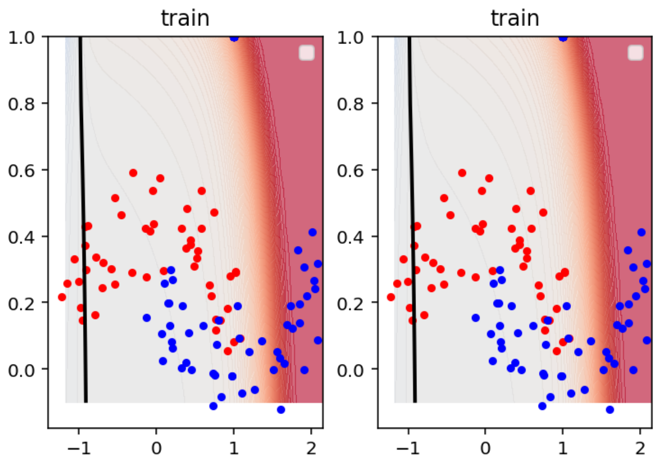
7. Plot the learning curve with λ=0.1 [1pt]

```
fig_7
```



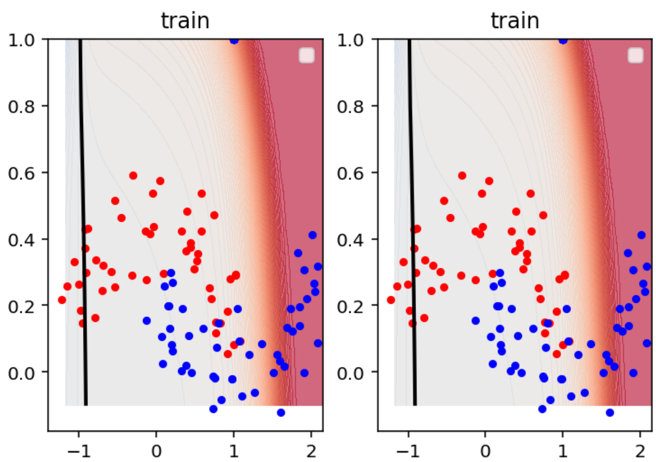
8. Plot the probability map of the obtained classifier with λ=0.00001 [1pt]

```
fig_8
```



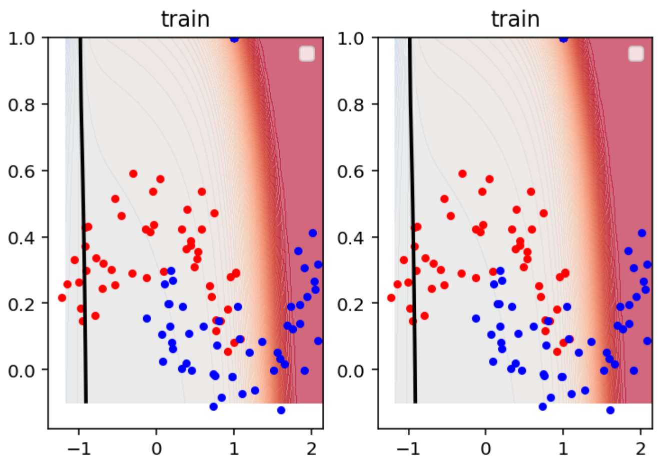
9. Plot the probability map of the obtained classifier with λ=0.0001 [1pt]

```
fig_9
```



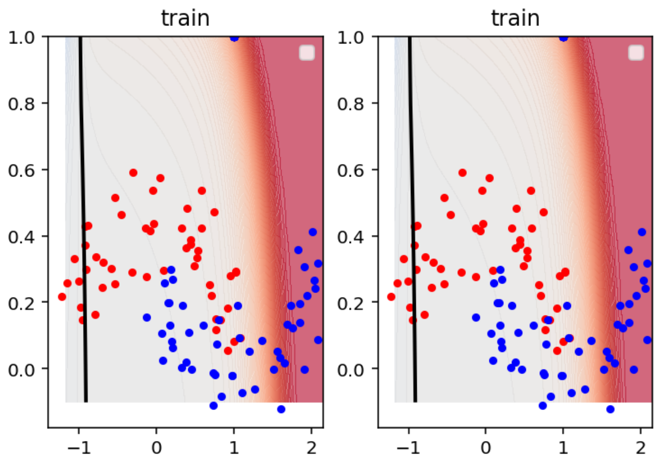
10. Plot the probability map of the obtained classifier with λ=0.001 [1pt]

```
1    fig_10
```



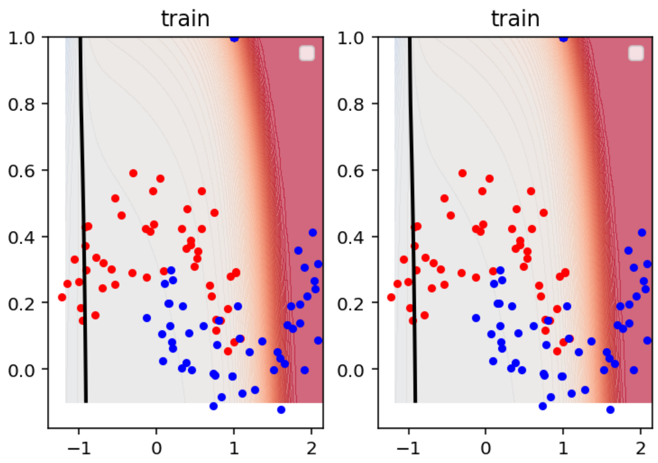
11. Plot the probability map of the obtained classifier with λ=0.01 [1pt]

```
1    fig_11
```



12. Plot the probability map of the obtained classifier with λ=0.1 [1pt]

```
1    fig_12
```



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

```
train_frame
```

	train_accuracy(%)
0.00001	78.0
0.00010	78.0
0.00100	78.0
0.01000	78.0
0.10000	78.0

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

```
test_frame
```

	test_accuracy(%)
0.00001	77.0
0.00010	77.0
0.00100	77.0
0.01000	77.0
0.10000	77.0