Logistic regression for a binary classification with a regularization

20145822 김영현

Training Code

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

# data input
data    = np.genfromtxt("/content/drive/My Drive/Colab Notebooks/data07/data-nonlinear.txt", del

pointX  = data[:, 0]
pointY  = data[:, 1]
label   = data[:, 2]

pointX0 = pointX[label == 0]
pointY0 = pointY[label == 0]

pointX1 = pointX[label == 1]
pointY1 = pointY[label == 1]

# function definition
# calculate f_k value
def func_k_calc(x, y, x_exp, y_exp):
  return (x ** x_exp) * (y ** y_exp)
# calculate g function value
def func_calc(theta_list, x, y):
  func_val = 0
  for i in range(10):
    for j in range(10):
      func_val += theta_list[i][j] * func_k_calc(x, y, i, j)
  return func_val
# calculate z values
def z_calc(theta_list, pointX, pointY):
  z = []
  for i in range(len(pointX)):
    z_iteration = func_calc(theta_list, pointX[i], pointY[i])
    z.append(z_iteration)
  return z
# calculate sigmoid values
def calc_sigmoid(z):
  sigmoid = []
  for i in range(len(z)):
    sigmoid.append(1/(1+math.exp(-z[i])))
```

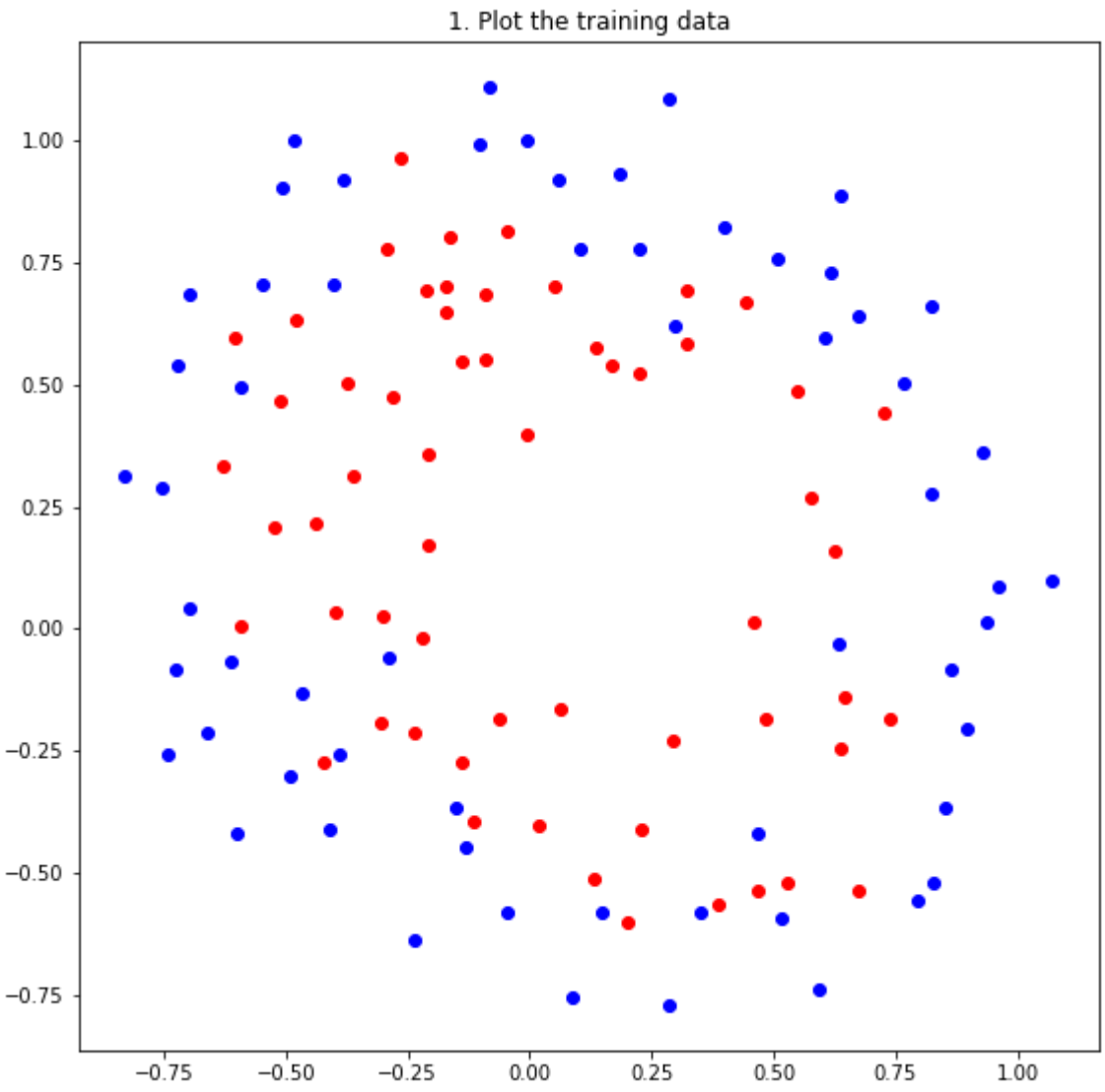
```
    sigmoid.append(1/(1+math.exp(-z[i])))
  return sigmoid
# calculate objective function value
def ob_func(label, sigmoid, reg_param, theta_list):
  sum_left = 0
  for i in range(len(label)):
    oprd_left = (-1*label[i]) * math.log(sigmoid[i])
    oprd_right = (1-label[i]) * math.log(1-sigmoid[i])
    sum_left += oprd_left - oprd_right
  sum_left = sum_left/len(label)
  sum_right = 0
  for i in range(10):
    for j in range(10):
      sum_right += theta_list[i][j] ** 2
  sum_right = sum_right * reg_param / 2
  return sum_left + sum_right
# calculate next theta value
def theta_desc(theta_list, alpha, pointX, pointY, label, sigmoid, reg_param):
  sum = np.zeros((10,10))
  for k in range(len(sigmoid)):
    for i in range(10):
      for j in range(10):
        sum[i][j] += (sigmoid[k] - label[k]) * func_k_calc(pointX[k], pointY[k], i, j)
  for i in range(10):
      for j in range(10):
        sum[i][j] = alpha * (sum[i][j] / len(sigmoid) + reg_param * theta_list[i][j])
        theta_list[i][j] = theta_list[i][j] - sum[i][j]
  return theta_list

# variable declaration
# array for store each regularization's theta
theta_list_over = np.ones((10,10))
theta_list_just = np.ones((10,10))
theta_list_under = np.ones((10,10))
# regularization control parameter [over, just, under]
reg_params = [0.01, 0.05, 0.1]
# learning rate
alpha = 0.1
# iteration counter
iteration = 0
# list for store data at every iteration
ob_func_list = []
accuracy_list = []

# start iteration
while True:
  # calculate each value for this iteration
  z_list_over = z_calc(theta_list_over, pointX, pointY)
  z_list_just = z_calc(theta_list_just, pointX, pointY)
  z_list_under =  z_calc(theta_list_under, pointX, pointY)
  z_list = [z_list_over, z_list_just, z_list_under]
  theta_list = [theta_list_over, theta_list_just, theta_list_under]
  sigmoid_list = []
```

```
  sigmoid_list = []
  for i in range(3):
    sigmoid_list.append(calc_sigmoid(z_list[i]))
  ob_func_val = []
  for i in range(3):
    ob_func_val.append(ob_func(label, sigmoid_list[i], reg_params[i], theta_list[i]))
  # store predictions
  predictions = []
  for i in range(3):
    predictions_temp = []
    for j in sigmoid_list[i]:
      if j < 0.5:
        predictions_temp.append(0)
      else:
        predictions_temp.append(1)
    predictions.append(predictions_temp)
  # calculate accuracy
  acc_temp = [0, 0, 0]
  for i in range(3):
    acc_hit = 0
    for j in range(len(label)):
      if label[j] == predictions[i][j]:
        acc_hit += 1
    acc_temp[i] = (acc_hit/len(label) * 100)
  accuracy_list.append(acc_temp)
  # store each value
  ob_func_list.append(ob_func_val)
  # escape rule
  if iteration > 0:
    escape_hit = 0
    for i in range(3):
      if abs(ob_func_list[iteration][i] - ob_func_list[iteration-1][i]) < 0.0001:
        escape_hit += 1
    if escape_hit == 3:
      break
  # update next theta values & iteration value
  theta_list_over = theta_desc(theta_list_over, alpha, pointX, pointY, label, sigmoid_list[0], r
  theta_list_just = theta_desc(theta_list_just, alpha, pointX, pointY, label, sigmoid_list[1], r
  theta_list_under = theta_desc(theta_list_under, alpha, pointX, pointY, label, sigmoid_list[2],
  iteration += 1

print("Training finished with")
print("iteration : ", iteration)
print("training error : ", ob_func_list[-1])
print("final accuracy : ", accuracy_list[-1])

iterations = [i for i in range(iteration+1)]
```

Submission

1. Plot the training data

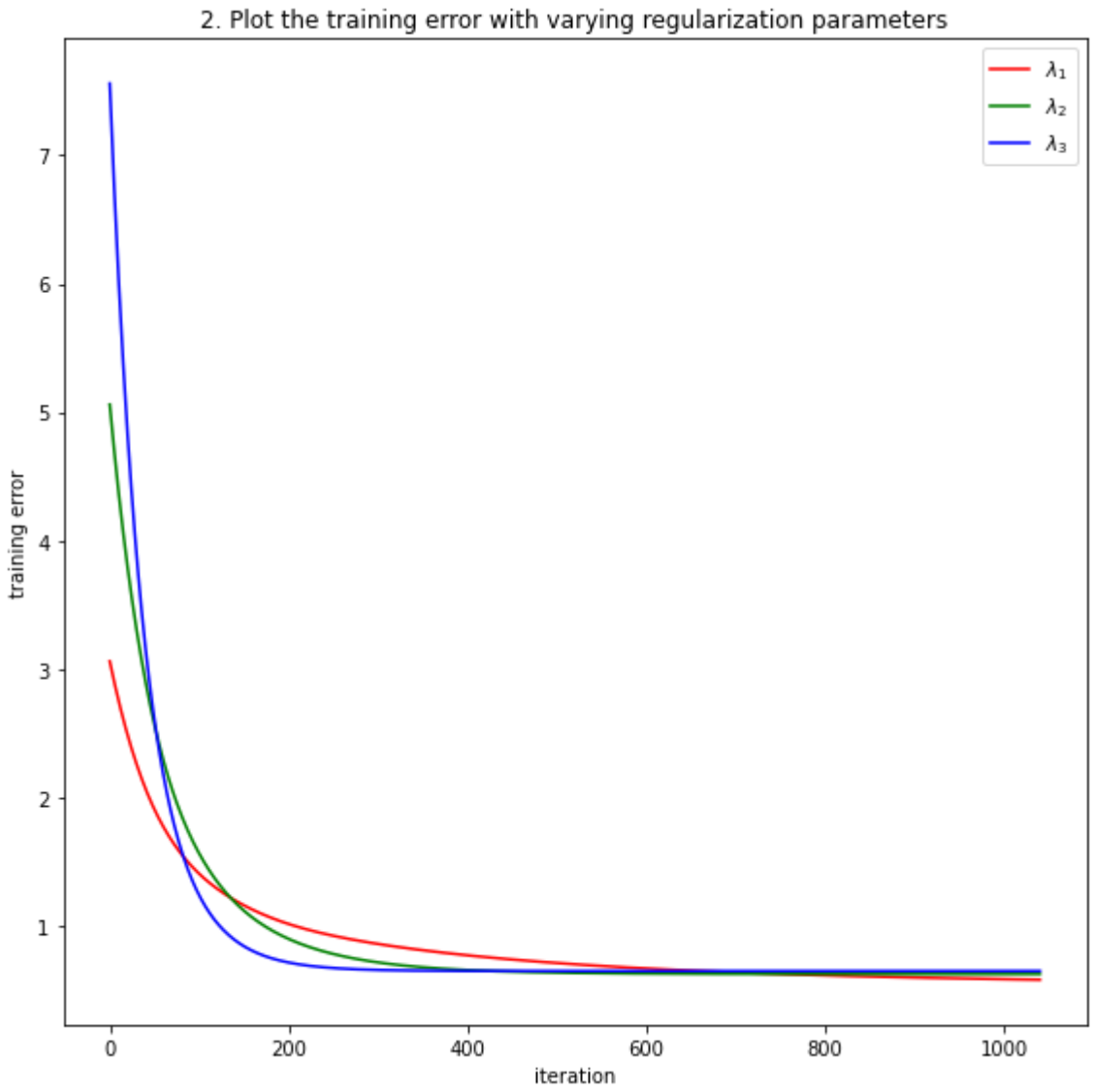
```
# 1. Plot the training data
plt.figure(figsize=(8,8))
plt.title("1. Plot the training data")
plt.scatter(pointX0, pointY0, c='b')
plt.scatter(pointX1, pointY1, c='r')
plt.tight_layout()
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



2. Plot the training error with varying regularization parameters

```
plt.figure(figsize=(8,8))
plt.title("2. Plot the training error with varying regularization parameters")
plt.xlabel('iteration')
plt.ylabel('training error')
plt.plot(iterations, [i[0] for i in ob_func_list], c='r', label='$₩lambda_1$')
```

```
plt.plot(iterations, [i[1] for i in ob_func_list], c='g', label='$Wlambda_2$')
plt.plot(iterations, [i[2] for i in ob_func_list], c='b', label='$Wlambda_3$')
plt.tight_layout()
plt.legend()
plt.show()
```



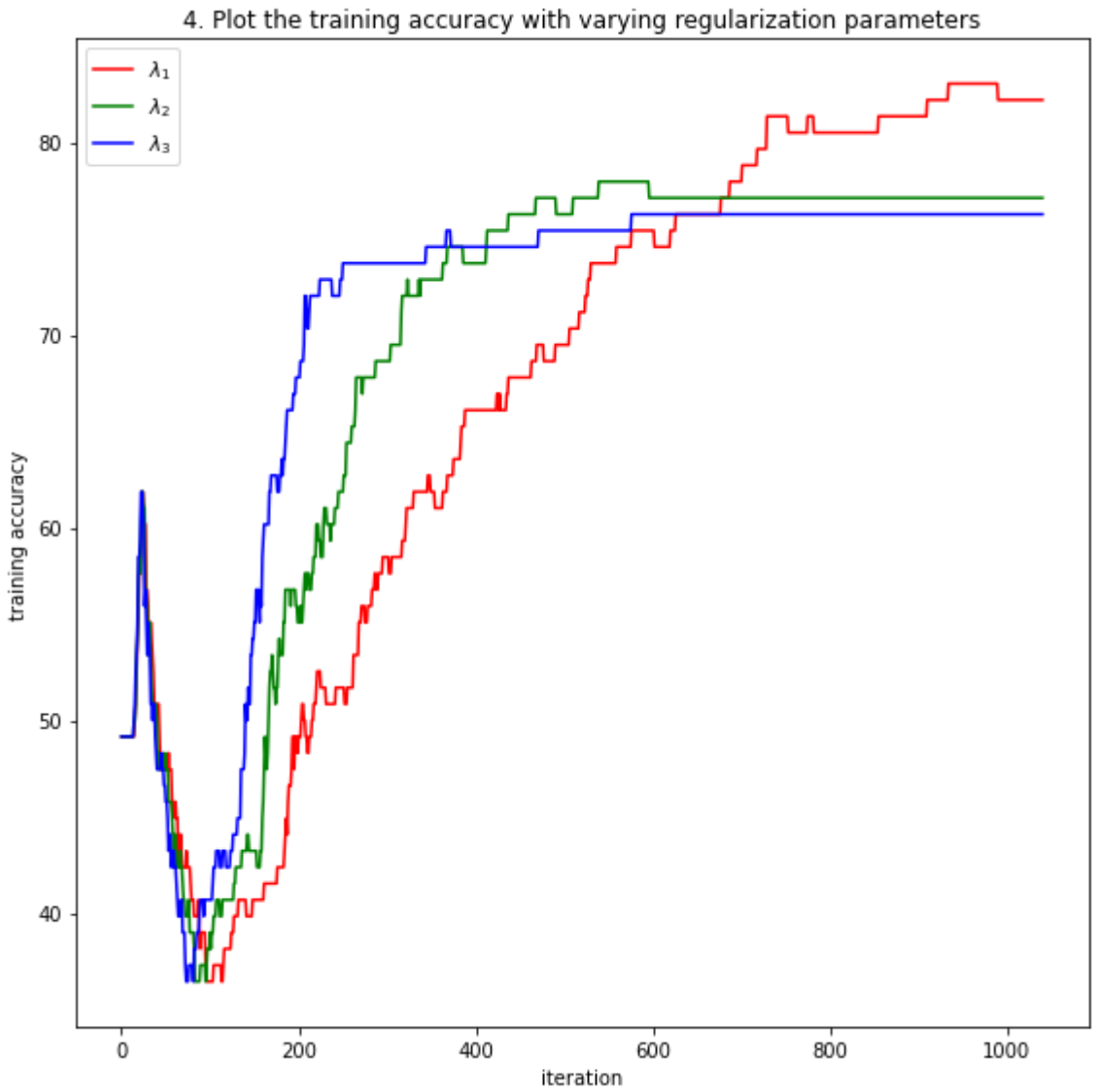
3. Display the values of the chosen regularization parameters

```
print("3. Display the values of the chosen regularization parameters")
print("Wu03BB1 = Wx1b[1;31m", reg_params[0])
print("Wx1b[Wu03BB2 = Wx1b[1;32m", reg_params[1])
print("Wx1b[Wu03BB3 = Wx1b[1;34m", reg_params[2])
```

```
3. Display the values of the chosen regularization parameters
λ1 =  0.01
λ2 =  0.05
λ3 =  0.1
```

4. Plot the training accuracy with varying regularization parameters

```
plt.figure(figsize=(8,8))
plt.title("4. Plot the training accuracy with varying regularization parameters")
plt.xlabel('iteration')
plt.ylabel('training accuracy')
plt.plot(iterations, [i[0] for i in accuracy_list], c='r', label='$₩lambda_1$')
plt.plot(iterations, [i[1] for i in accuracy_list], c='g', label='$₩lambda_2$')
plt.plot(iterations, [i[2] for i in accuracy_list], c='b', label='$₩lambda_3$')
plt.tight_layout()
plt.legend()
plt.show()
```



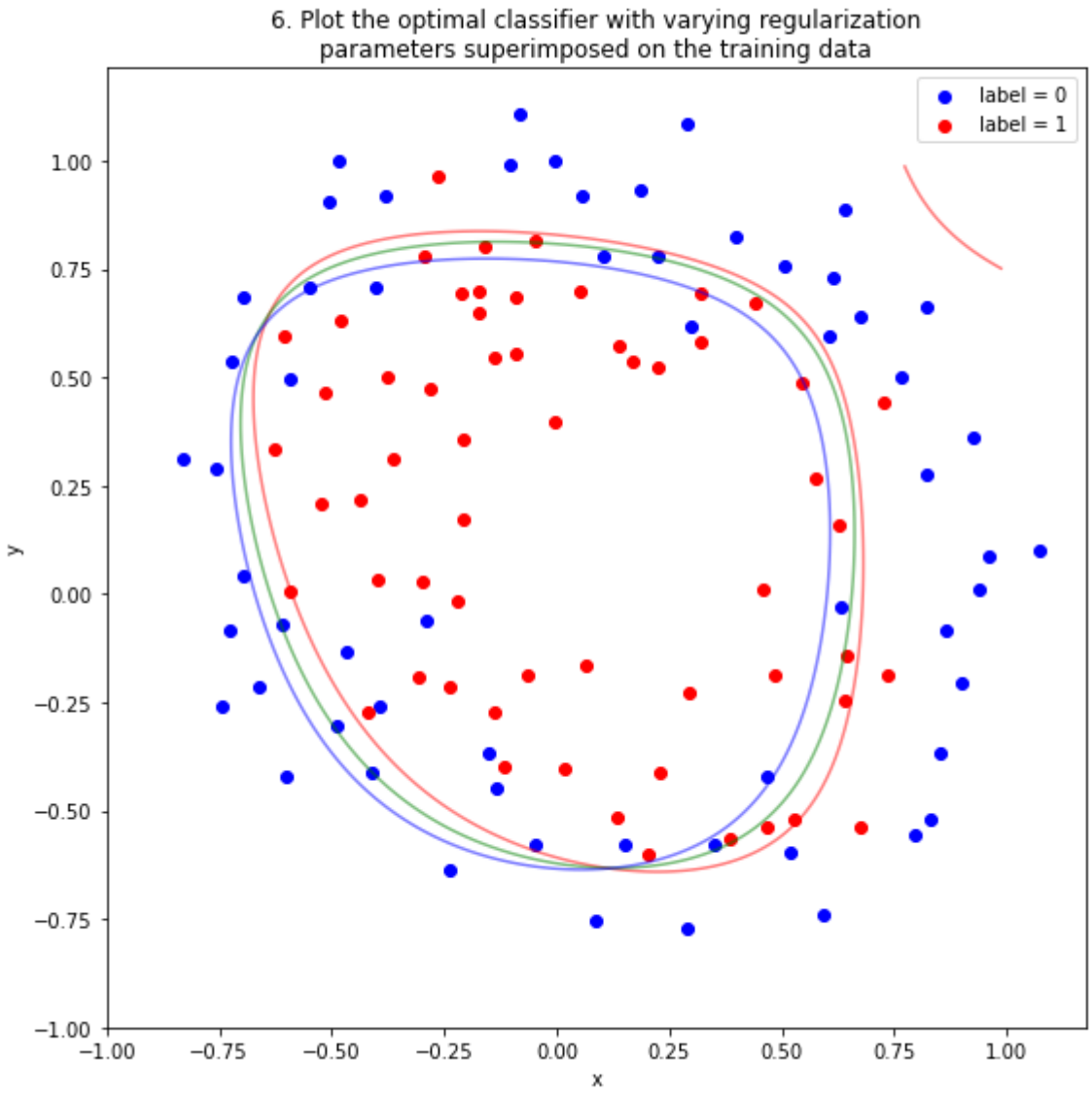
5. Display the final training accuracy with varying regularization parameters

```
print("5. Display the final training accuracy with varying regularization parameters")
print("accuracy of ₩u03BB1 =₩x1b[1;31m", accuracy_list[-1][0], "%")
print("₩x1b[accuracy of ₩u03BB2 =₩x1b[1;32m", accuracy_list[-1][1], "%")
print("₩x1b[accuracy of ₩u03BB3 =₩x1b[1;34m", accuracy_list[-1][2], "%")
```

```
5. Display the final training accuracy with varying regularization parameters
accuracy of λ1 = 82.20338983050848 %
accuracy of λ2 = 77.11864406779661 %
accuracy of λ3 = 76.27118644067797 %
```

6. Plot the optimal classifier with varying regularization parameters superimposed on the training data

```
val = np.arange(-1, 1, 0.01)
X, Y = np.meshgrid(val, val)
Z_1 = func_calc(theta_list_over, X, Y)
Z_2 = func_calc(theta_list_just, X, Y)
Z_3 = func_calc(theta_list_under, X, Y)
# 6. Plot the optimal classifier with varying regularization parameters
# superimposed on the training data
plt.figure(figsize=(8,8))
plt.title("6. Plot the optimal classifier with varying regularization₩n"+
          "parameters superimposed on the training data")
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(pointX0, pointY0, c='b', label='label = 0')
plt.scatter(pointX1, pointY1, c='r', label='label = 1')
plt.contour(X, Y, Z_1, 0, alpha=.5, colors='r')
plt.contour(X, Y, Z_2, 0, alpha=.5, colors='g')
plt.contour(X, Y, Z_3, 0, alpha=.5, colors='b')
plt.tight_layout()
plt.legend()
plt.show()
```


6. Plot the optimal classifier with varying regularization parameters superimposed on the training data
label = 0
label = 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x