Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

Branch: assignment05 ▾ Machine-Learning-2020-Spring-Class / assignment05.ipynb

Find file | Copy path

Reichidad Colaboratory를 통해 생성됨 3edf2ac 3 minutes ago

1 contributor

290 lines (290 sloc) 111 KB

Raw | Blame | History

Open in Colab
(https://colab.research.google.com/github/Reichidad/Machine-Learning-2020-Spring-Class/blob/assignment05/assignment05.ipynb)

Assignment 05. Logistic regression for a binary classification - 20145822 김영현

1. Plot the training data

2. Plot the estimated parameters

3. Plot the training error

4. Plot the obtained classifier

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

# data input
data = np.genfromtxt("/content/drive/My Drive/Colab Notebooks/data05/data.txt", delimiter=',')

x = data[:, 0]
y = data[:, 1]
label = data[:, 2]

x_label0 = x[label == 0]
x_label1 = x[label == 1]

y_label0 = y[label == 0]
y_label1 = y[label == 1]

# function definition
```

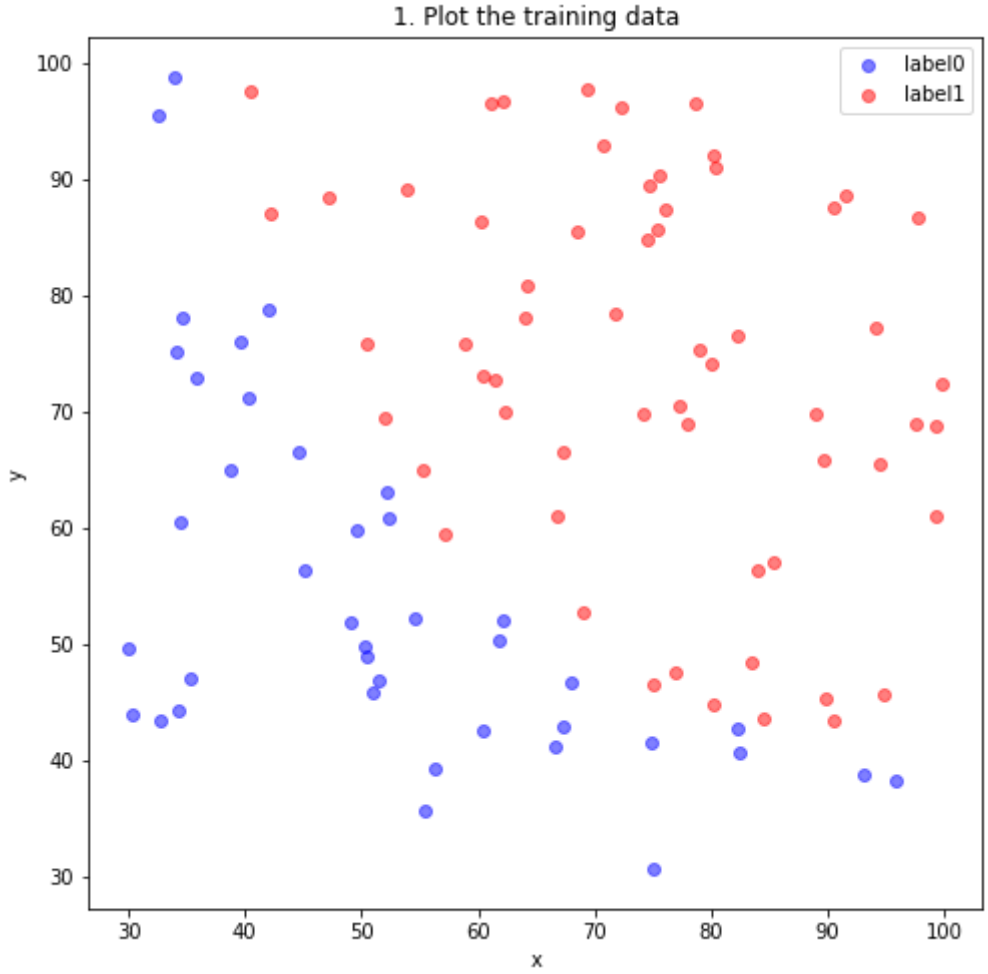
```
# calculate z value
def calc_z(theta_0, theta_1, theta_2, x, y):
  z= []
  for i in range(len(x)):
    z.append(theta_0 + (theta_1 * x[i]) + (theta_2 * y[i]))
  return z
# calculate sigmoid value
def calc_sigmoid(z):
  sigmoid = []
  for i in range(len(z)):
    sigmoid.append(1/(1+math.exp(-z[i])))
  return sigmoid
# calculate objective function value
def ob_func(label, sigmoid):
  sum = 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 += oprd_left - oprd_right
  return sum/len(label)
# calculate next theta value
def theta_desc(theta, alpha, x, y, label, sigmoid, key):
  sum = 0
  for i in range(len(sigmoid)):
    if key == 0:
      sum += sigmoid[i] - label[i]
    elif key == 1:
      sum += (sigmoid[i] - label[i]) * x[i]
    elif key == 2:
      sum += (sigmoid[i] - label[i]) * y[i]
  return theta - (alpha * sum / len(sigmoid))

# variable initialization
theta_0 = 0.01
theta_1 = 0.01
theta_2 = 0.01
alpha = 0.001
iteration = 0
# variable list for store iteration data
ob_func_list = []
theta_0_list = []
theta_1_list = []
theta_2_list = []
```

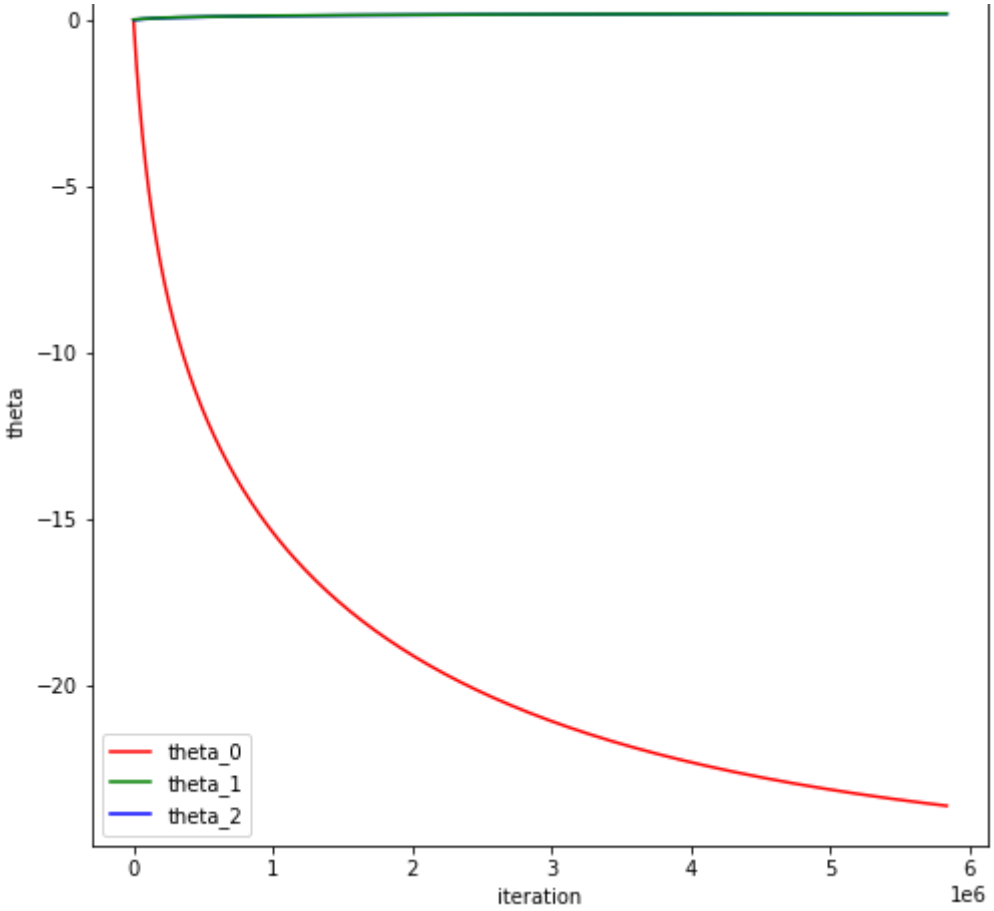
```
theta_2_list = []
# iteration
while True:
  # calculate each value for this iteration
  z_list = calc_z(theta_0, theta_1, theta_2, x, y)
  sigmoid_list = calc_sigmoid(z_list)
  ob_func_val = ob_func(label, sigmoid_list)
  # store each value
  theta_0_list.append(theta_0)
  theta_1_list.append(theta_1)
  theta_2_list.append(theta_2)
  ob_func_list.append(ob_func_val)

  # escape rule
  if iteration > 0:
    if abs(theta_0_list[iteration-1] - theta_0)< 0.0000005:
      break
  # update next theta values & iteration value
  theta_0 = theta_desc(theta_0, alpha, x, y, label, sigmoid_list, 0)
  theta_1 = theta_desc(theta_1, alpha, x, y, label, sigmoid_list, 1)
  theta_2 = theta_desc(theta_2, alpha, x, y, label, sigmoid_list, 2)
  iteration += 1
print('Iteration Finished with each value')
print('iteration: ', iteration)
print('theta_0: ', theta_0)
print('theta_1: ', theta_1)
print('theta_2: ', theta_2)
print('training error: ', ob_func_val)
iterations = []
for i in range(iteration+1):
  iterations.append(i)

# 1. Plot the training data
# plot the training data points (x,y) with their labels
# blue for label 0 and red for label 1
plt.figure(1,figsize=(8, 8))
plt.title('1. Plot the training data')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(x_label0, y_label0, alpha=0.5, c='b', label='label0')
plt.scatter(x_label1, y_label1, alpha=0.5, c='r', label='label1')
plt.legend()
```

```
# 2. Plot the estimated parameters
# plot the thetas at every iteration of gradient descent until convergence
# (θ0,θ1,θ2) should be red, green, blue, respectively
plt.figure(2,figsize=(8, 8))
plt.title('2. Plot the estimated parameters')
plt.xlabel('iteration')
plt.ylabel('theta')
plt.plot(iterations, theta_0_list, c='r', label='theta_0', zorder = 1)
plt.plot(iterations, theta_1_list, c='g', label='theta_1', zorder = 3)
plt.plot(iterations, theta_2_list, c='b', label='theta_2', zorder = 2)
plt.legend()

# 3. Plot the training error
# plot the training error J(θ0,θ1,θ2)
# at every iteration of gradient descent until convergence (in blue color)
plt.figure(3,figsize=(8, 8))
plt.title('3. Plot the training error')
plt.xlabel('iteration')
plt.ylabel('training error')
plt.plot(iterations, ob_func_list, c='b', label='training error')
plt.legend()

# 4. Plot the obtained classifier
val = np.arange(30, 100, 0.5)
X, Y = np.meshgrid(val, val)
sigmoid_val = []

for t_one in list(val):
  for t_zero in list(val):
    z_temp = theta_0 + (theta_1 * t_one) + (theta_2 * t_zero)
    sigmoid_val.append(1/(1+math.exp(-z_temp)))

sigmoid_array = np.array(sigmoid_val)
S = sigmoid_array.reshape(X.shape)
plt.figure(4,figsize=(8, 8))
plt.title('4. Plot the obtained classifier')
color_map = plt.pcolormesh(X, Y, S, vmin=0., vmax=1., cmap='RdBu_r', zorder = 1)
plt.colorbar()
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(x_label0, y_label0, alpha=0.5, c='b', label='label0', zorder = 3)
plt.scatter(x_label1, y_label1, alpha=0.5, c='r', label='label1', zorder = 3)
```

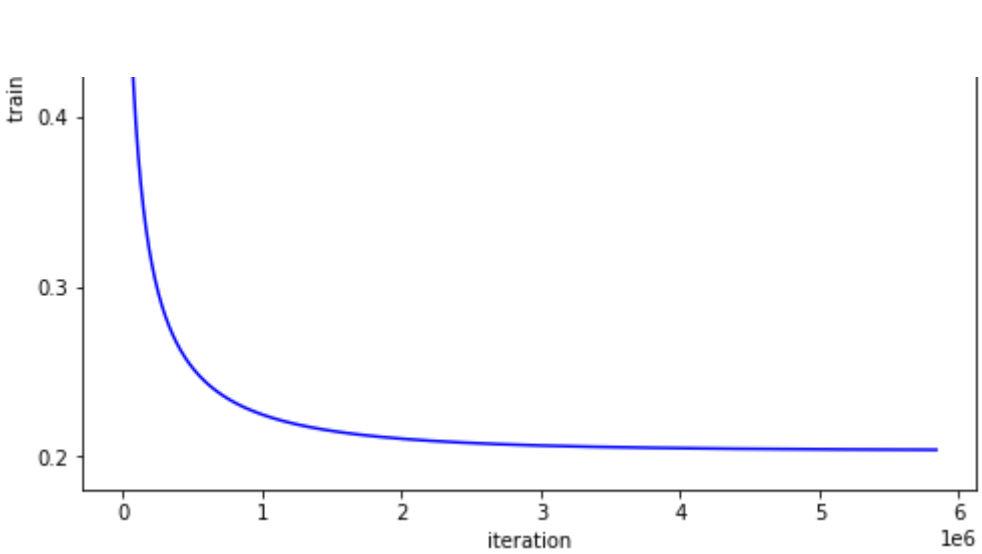
```
plt.legend()

plt.show()
```

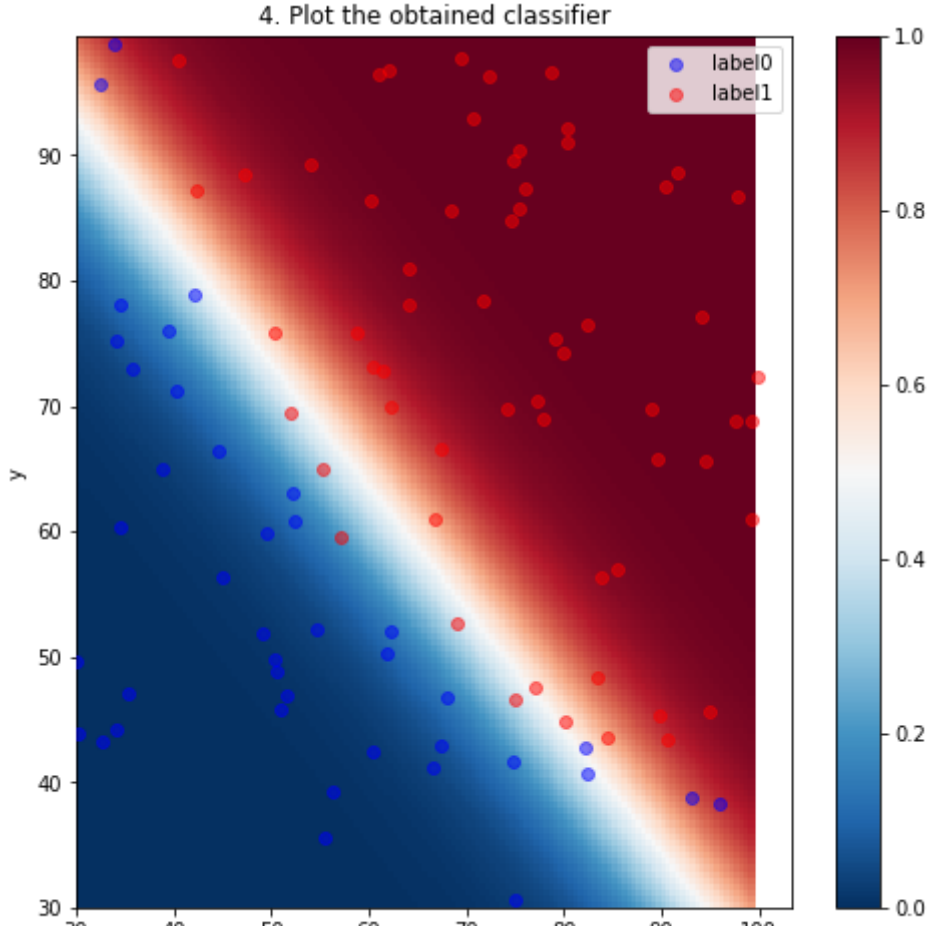
```
Iteration Finished with each value
iteration:  5838389
theta_0:  -23.611853697554782
theta_1:  0.19384101656252112
theta_2:  0.1889317562034806
training error:  0.20387463445005125
```





3. Plot the training error


train
0.4
0.3
0.2
0
1
2
3
4
5
6
iteration
1e6


4. Plot the obtained classifier
label0
label1
y
90
80
70
60
50
40
30
1.0
0.8
0.6
0.4
0.2
0.0


30
40
50
60
70
80
90
100
x