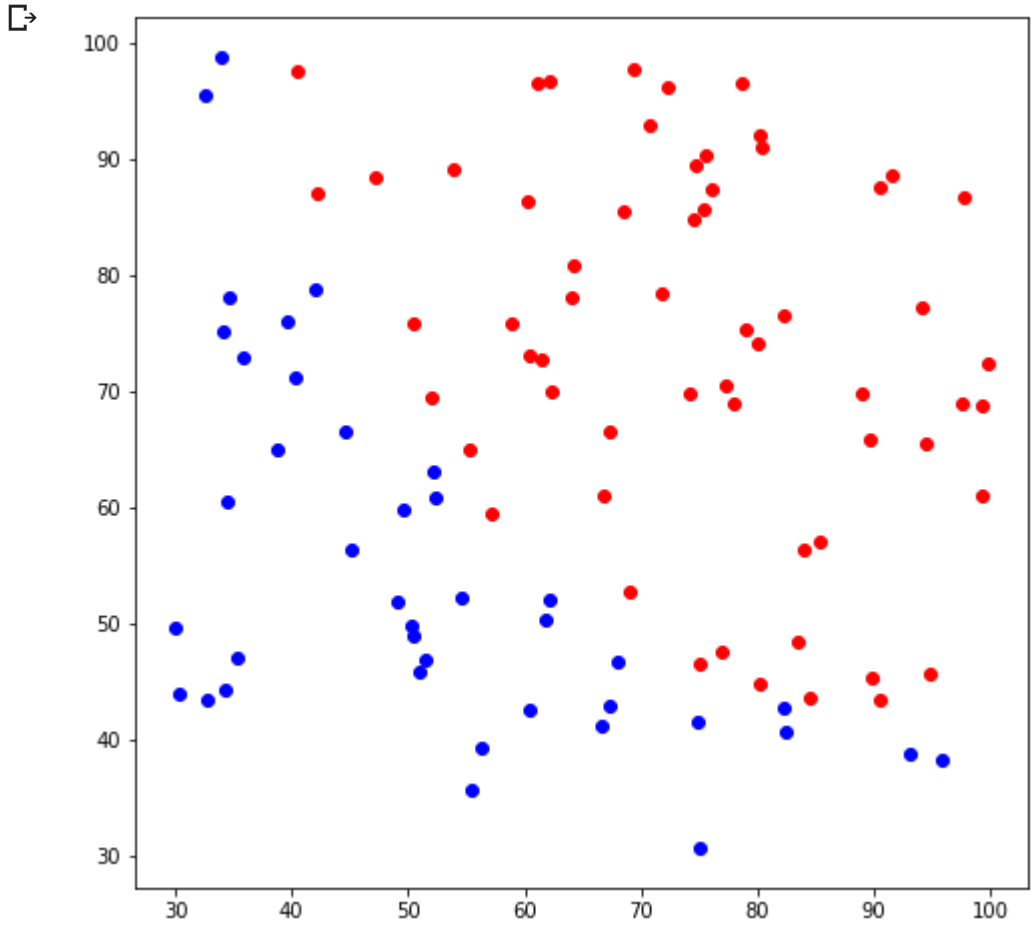
1. Load data

- data.txt 파일을 동일 디렉터리에 위치시키고 데이터를 읽었습니다.
- 읽은 데이터들을 visualize할때 label이 0인 data는 blue, 1인 data는 red로 시각화했습니다.

```
import csv
import numpy as np
import matplotlib.pyplot as plt
import math
'''
1. Data
 - Config
 - Load Data
'''

# Config
learning_rate=0.0002014
opt_threshold=0.00000002748
t0, t1, t2=-10., -5., 10
theta=np.array([[t0, t1, t2]])

# Load data
path = "drive/My Drive/Classroom/Machine Learning (2) 2020-1\
/class-MachineLearning/assignment05/"
train=[]
test=[]

data = np.genfromtxt(path+"data.txt", delimiter=',')

x = data[:, 0]
y = data[:, 1]
m = len(x)

label = data[:, 2]

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

y_label0 = y[label == 0]
y_label1 = y[label == 1]
temp=np.ones(m)

input_data=np.hstack((temp.reshape(-1, 1), x.reshape(-1, 1), y.reshape(-1, 1)))

plt.figure(figsize=(8, 8))
plt.scatter(x_label0, y_label0, alpha=1, c='b')
plt.scatter(x_label1, y_label1, alpha=1, c='r')
plt.show()
```



2. Optimize Logistic regression

- train data에 대한 hypothesis, objective function, gradient descent를 정의했습니다.
- 연산과정들을 matrix로 빠르게 계산하기 위해 np.dot함수를 사용했습니다.
- 각 theta(feature)들에 맞는 update함수를 정의함으로써 학습동안 각각의 theta들이 모두 update되도록
- 수렴하는 기준을 0.00000002748로 설정하고 iteration에 대한 loss변화량이 이 기준값보다 작으면 수렴했다 판단했습니다.
- Objective function에서 log0으로 인한 -inf값 방지를 위해 동일한 notation이지만 inf를 생성하지 않도록 식을 수정했습니다.

```
'''
2. Hypothesis
 - Define hypothesis and sigmoid hypothesis.

3. Objective function
 - Define objective function, modify notation for J.

4. Gradient Descent
 - Define derivations.
 - Update theta0, theta1, theta2.
```

```
'''

# Define hypothesis
def sigmoid_h() :
    return 1/(1+np.exp(-z))

def hypothesis() :
    return input_data.dot(theta.T).flatten()

z=hypothesis()
h=sigmoid_h()

# Define objective function
def objective_function() :
    positive_z=np.array(list(map(lambda x : max(x, 0), z)))
    J = np.sum(positive_z - z*label + np.log(1+np.exp(-np.abs(z)))) / m
    return J

# Define derivations
def dev() :
    dev0=np.sum(h-label) / m
    dev1=np.sum((h-label)*x) / m
    dev2=np.sum((h-label)*y) / m
    dev=[dev0, dev1, dev2]
    return dev

# Update theta0, theta1, theta2, theta3
def gradient_descent() :
    return (theta[0][0]-(learning_rate*dev0), theta[0][1]-(learning_rate*dev1),\
            theta[0][2]-(learning_rate*dev2))

J_list=[]
theta0_list=[]
theta1_list=[]
theta2_list=[]
h_list=[]
# Train
count=0
for i in range(20000) :
    h_list.append(h)
    count+=1
    J=objective_function()

    J_list.append(J)

    theta0_list.append(theta[0][0])
    theta1_list.append(theta[0][1])
    theta2_list.append(theta[0][2])


    dev0, dev1, dev2 = dev()
    temp0, temp1, temp2 = theta[0]
    theta[0][0], theta[0][1], theta[0][2] = gradient_descent()
    z=hypothesis()
    h=sigmoid_h()
```

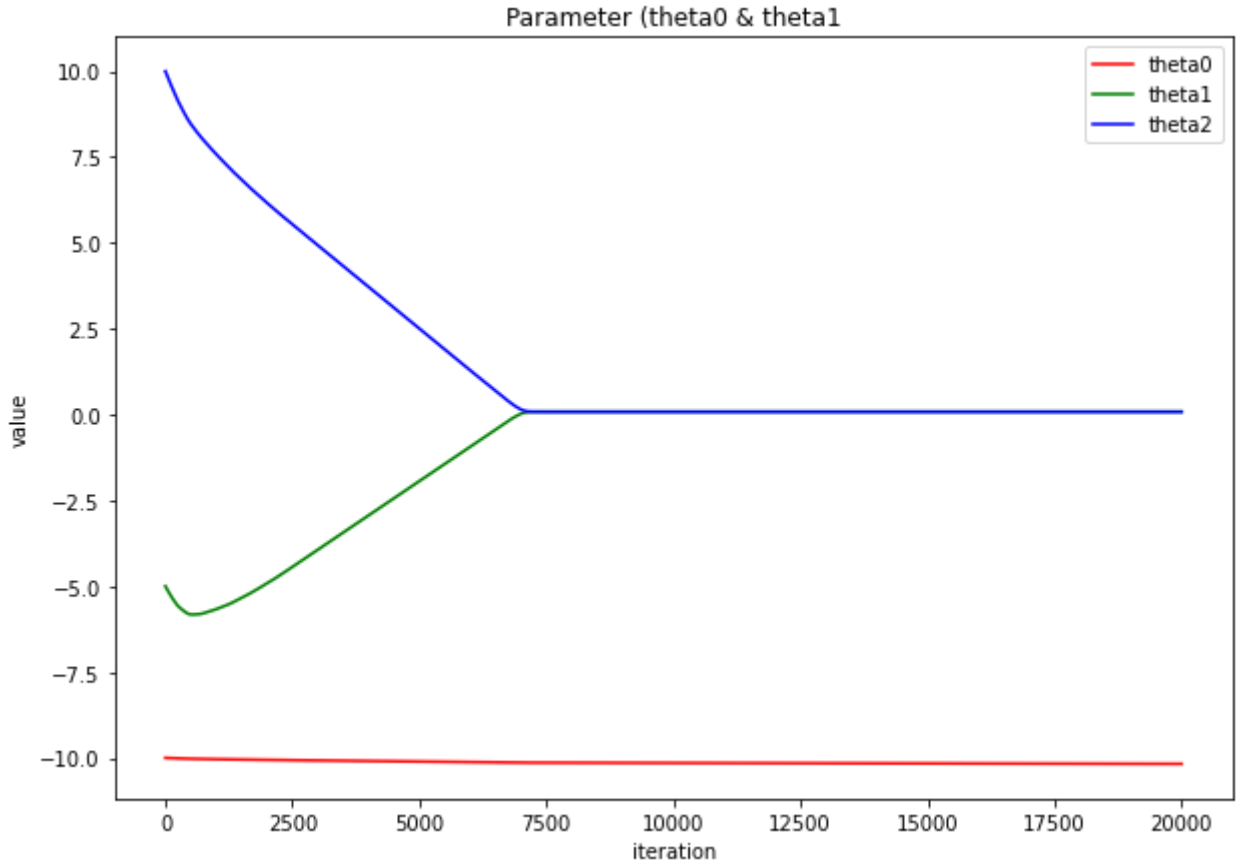
```
    if len(J_list)>=2 and \
    abs(J_list[-2] - J_list[-1]) <opt_threshold :
        break
print("theta0 : ", theta[0][0])
print("theta1 : ", theta[0][1])
print("theta2 : ", theta[0][2])
print("loss : ", J)
```

```
theta0 :  -10.173494868641017
theta1 :  0.0868318479067422
theta2 :  0.08059519104611516
loss :  0.27049902948840765
```

3. Model parameters

- update를 할때마다 변하는 theta0, theta1, theta2, theta3의 값들을 따로 리스트에 저장해뒀었습니다.
- 각 iteration마다 위 theta들이 각각 어떻게 변하는지 시각화했습니다.

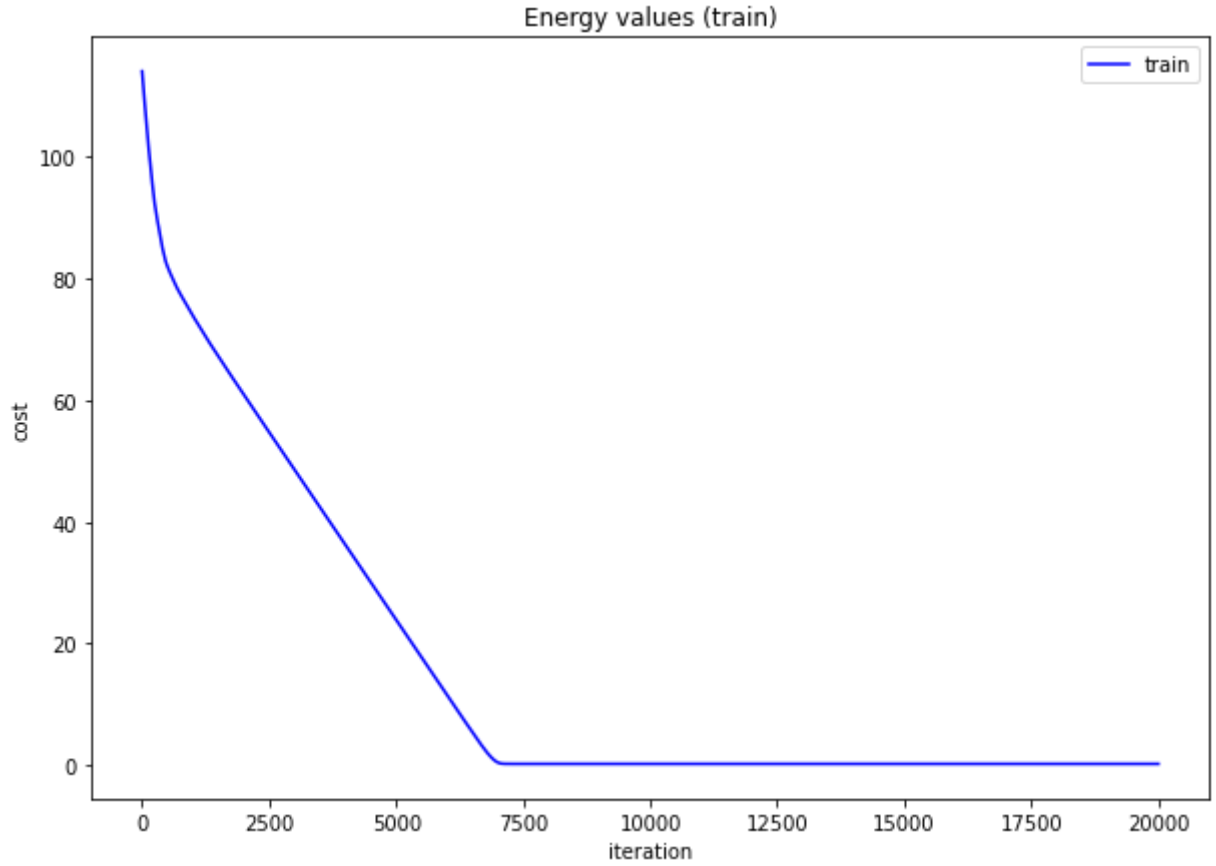
```
'''
Visualize theta
'''
plt.figure(figsize=(10, 7))
plt.plot(theta0_list, color='red', label='theta0')
plt.plot(theta1_list, color='green', label='theta1')
plt.plot(theta2_list, color='blue', label='theta2')
plt.legend(loc='upper right')
plt.xlabel("iteration")
#plt.xlim(left=-1000)
plt.ylabel("value")
plt.title("Parameter (theta0 & theta1")
plt.show()
```



4. Training error

- training data에 대해서 update를 할때마다 train_J_list에 그동안의 loss 값들을 넣었습니다.
- 각 iteration마다 loss값이 어떻게 변하는지 시각화했습니다.

```
'''
Visualize Loss
'''
plt.figure(figsize=(10, 7))
plt.plot(J_list, color='blue', label='train')
plt.xlabel("iteration")
plt.ylabel("cost")
plt.legend(loc='upper right')
plt.title("Energy values (train)")
plt.show()
```



5. Visualize classifier

- 그래프에서 x와 y에 대한 범위는 30~100으로 설정하고
  해당 범위 내에서 0.5씩 증가하면서 각각 141개의 값을 갖게됩니다.
- meshgrid를 통해서 x와 y값들을 좌표평면에 대응시킬 수 있는 grid matrix로 만듭니다.
- 각 x, y 좌표에 따른 hypothesis값을 구하는 함수를 정의하고
  0에 가까울수록 blue, 1에 가까울수록 red로 plot했습니다.

```
'''
Visualize classifier
'''
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
def H(x, y) :
    H=theta[0][0] + theta[0][1] * x + theta[0][2] * y
    return 1/(1+np.exp(-H))

grid_x=np.linspace(start=30, stop=100, num=141)
grid_y=np.linspace(start=30, stop=100, num=141)
# Construct meshgrid between x and y
grid_x, grid_y=np.meshgrid(grid_x, grid_y)

```

```
16
17 fig = plt.figure(figsize=(12, 8))
18
19 grid_H=np.array([H(x, y) for (x, y) in \
20                  zip(np.ravel(grid_x), np.ravel(grid_y))])
21 grid_H = grid_H.reshape(grid_x.shape)
22
23 plt.scatter(grid_x, grid_y, c=grid_H, cmap='jet', s=3)
24 plt.scatter(x_label0, y_label0, alpha=1, c='b')
25 plt.scatter(x_label1, y_label1, alpha=1, c='r')
26 plt.show()
```