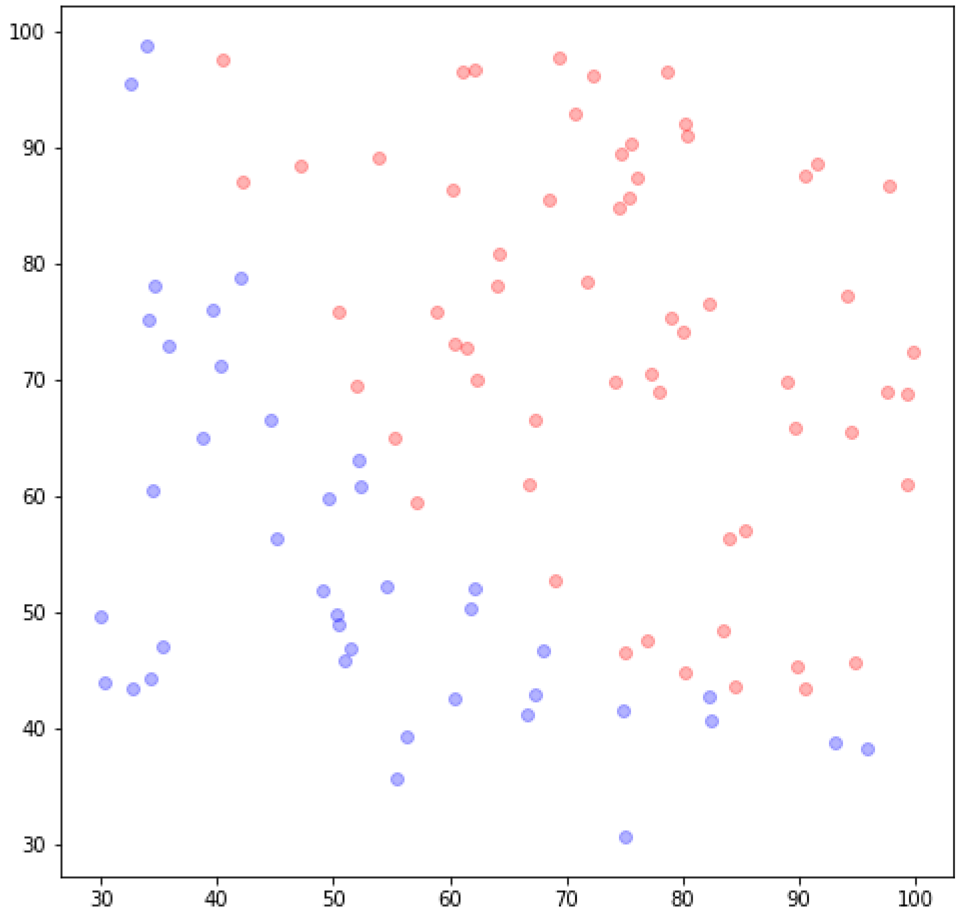
```
import csv
import numpy as np
import matplotlib.pyplot as plt
import math
'''
1. Data
 - Config
 - Load Data
 - Split loaded data
'''

minimum = -(1e-5)
# Config
learning_rate=0.0001
opt_threshold=1e-15
t0, t1, t2=0., 0., 0
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=0.3, c='b')
plt.scatter(x_label1, y_label1, alpha=0.3, c='r')
plt.show()

```

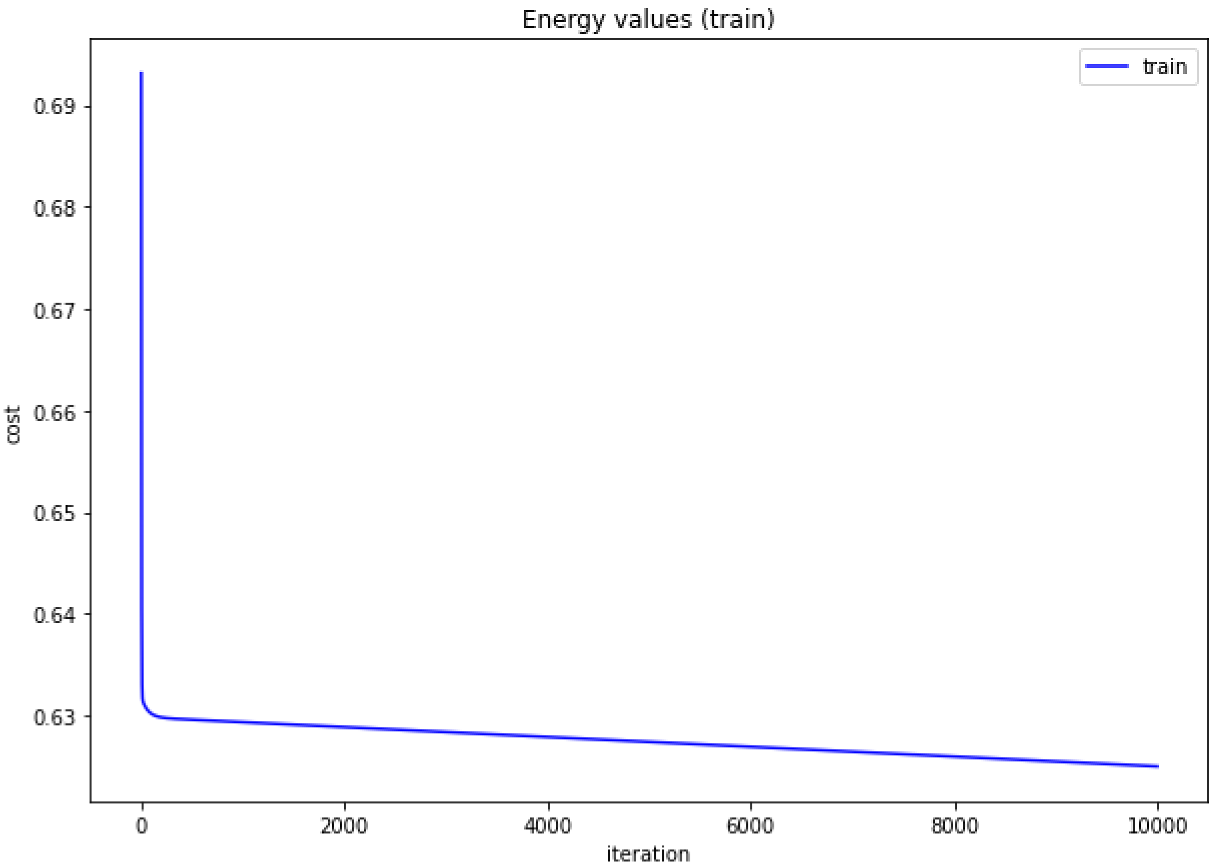



```
'''
2. Hypothesis
 - Define hypothesis for each train and test

3. Objective function
 - Define objective function for each train and test

4. Gradient Descent
 - Define derivations
 - Update theta0, theta1, theta2, theta3
'''

# Define hypothesis
def sigmoid(z) :

    return 1/ (1.+np.exp(-z))

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

h=hypothesis()

# Define objective function
def objective_function() :
    #print(np.log(h))
    #print('-'*20)
    #print(-label*np.log(h))
    #print('*'*20)
```

```
    #print(np.log(1-h))
    log_h=np.array(list(map(lambda x : minimum if np.isinf(np.log(x)) \
    and np.log(x) < 0 else np.log(x), h)))
    log_minus_h=np.array(list(map(lambda x : minimum if np.isinf(np.log(1-x)) \
    and np.log(1-x) < 0 else np.log(1-x), h)))
    #print(log_h, log_minus_h)
    #print(-(label*np.log(h)) - ((1-label)*np.log(1-h)))
    J = np.sum(-(label*log_h) - ((1-label)*log_minus_h))/ (m)
    #print(J)
    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]
    #print("dev : ", dev)
    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(10000) :
    h_list.append(h)
    count+=1
    J=objective_function()

    #print(J)
    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()

    h=hypothesis()


    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)
print(J_list)
plt.figure(figsize=(10, 7))
plt.plot(J_list, color='blue', label='train')
plt.xlabel("iteration")
#plt.ylim(top=1000, bottom=0)
#plt.xlim(right=10000, left=-1000)
plt.ylabel("cost")
plt.legend(loc='upper right')
plt.title("Energy values (train)")
plt.show()
```

```
(100,)
theta0 :  -0.06945358450750927
theta1 :  0.010907277343489178
theta2 :  0.0009912972636381611
loss :  0.6249867499098797
[0.6931471805599453, 0.6690967609080577, 0.6546428926941688, 0.645887606531179
```




```
'''
Visualize
'''
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()
```