Student-ID	201811552	Name	Watanabe Shingo	Filling Date	2020/11/30
Corse Name	Machine Learning and Information Retrieval System	Task	Relevance prediction as regression	Teacher	Hai-Tao, Yu

I defined two functions by myself. One is something to compute the loss function as the mean squared error and the other is something to train and optimize the parameter through gradient descent.

Source Code1

```
def mean_squared_error(m, prediction, t):
    return (1/m) * np.sum( ( prediction - t ) **2 )

```

The above source code 1 is the function which computes and returns mean squared error. The definition of mean squared error (MSE) is as follows

$MSE = \frac{1}{m} \sum_k (y_k - t_k)^2$ (y_k: prediction value, t_k: correct answer label).

Therefore, this function computes how far away from the prediction value and actual correct label value are from each other. And the smaller this result of calculation, the more accuracy will be.

Source Code2

```
def gradient_descent(f, init_x, lr=0.01, step_num=20):
    parameter = init_x
    cost_history = np.zeros(step_num)

    for h in range(step_num):
        for (qid, train_X, train_Y) in train_list_Qs:
            for i in range(len(train_Y)):
                x = train_X[i, :]
                y = train_Y[i]
                prediction = np.dot(x, parameter)
                grad = 2/float(len(train_Y)) * (x * (prediction - y)
)
```

```
                parameter -= lr * grad

        cost=0
        for (qid, train_X, train_Y) in train_list_Qs:
            predictions_per_query = train_X.dot(parameter)
            m = len(train_Y)
            cost_per_query = mean_squared_error(m,
predictions_per_query, train_Y)
            cost += cost_per_query
            # print(h, ":", cost)
        cost_history[h]  = cost # record the cost/loss per epoch

    return parameter, cost_history
```

In this function, I implemented the function to compute gradient and update the parameter so that the value of MSE is minimal. The linear regression model has the following form
$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \dots \theta_n x_n = \theta^T \cdot x.$
In addition, the result of partial differentiation of the MSE is as follows

$$\frac{\partial MSE}{\partial \theta} = \frac{2}{m} \sum_{i=1}^{m} (\theta^T \cdot x^i - y^i)^2 x^i$$

Therefore, the gradient is as follows

$$\nabla MSE = \frac{2}{m} X^T \cdot (X \cdot \theta - y)$$

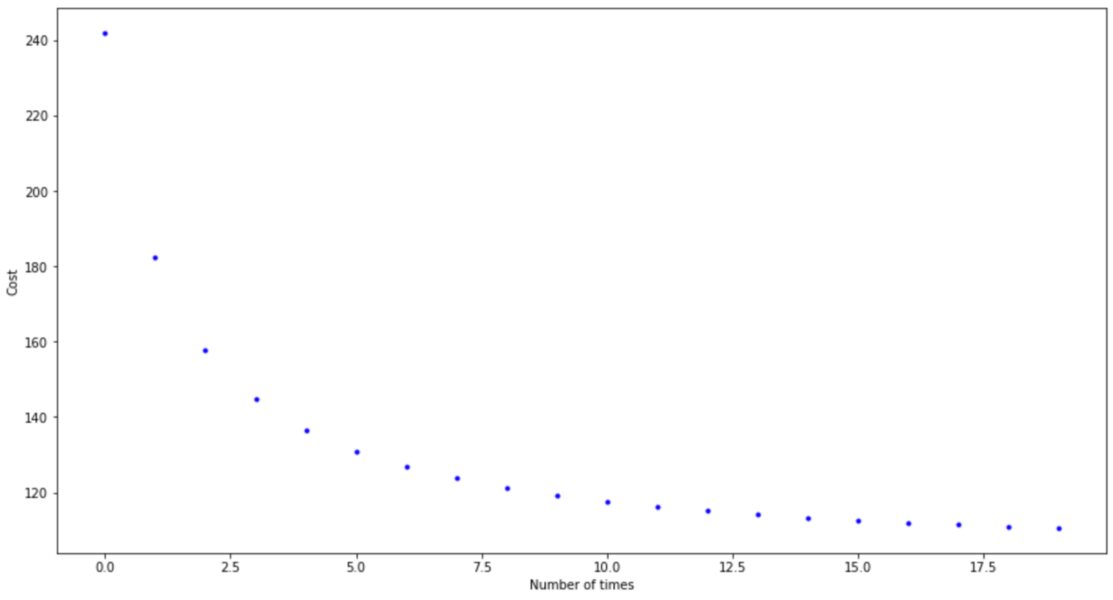
Also, the learning rate (lr) was set to 0.01 and the parameters were updated.
I implemented this content up to this point as follows

```
prediction = np.dot(x, parameter)
grad = 2/float(len(train_Y)) * (x * (prediction - y))
parameter -= lr * grad
```

Moreover, to see how much the loss was reduced, this function records the losses per epoch using mean squared error function. The results of 20 times studies are shown in the following graph1. As you can see, the losses were gradually decreasing.

Graph1

```
[<matplotlib.lines.Line2D at 0x7f9238315ca0>]
```



Cost
Number of times
240
220
200
180
160
140
120
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5