Damandeep Singh

Professor Liu

CS 549

11 March 2022

HA2 (Report)

GD.py:

I have completed the change variable in this file, consisting of the **alpha*gradient** part. For calculating the gradient part, I relied upon an equation mentioned in the lecture slides:

- Repeat until Convergence{

$$\theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_i (\theta_0 + \theta_1 x_i - y_i)$$

$$\theta_1 = \theta_1 - \alpha \frac{1}{m} \sum_i (\theta_0 + \theta_1 x_i - y_i) x_i$$

}

The **placeholder three** missed the gradient and (alpha/m) constant. I used the np.dot () function to get the dot product of the prediction part and subtract the true label (y array) from the result of the prediction part and lastly again call the dot product function on the transposed X array (feature) to get the gradient function. Here is the actual code I ended up with:

```
change =(alpha/m)*np.sum(np.dot(transposedX, np.dot(X, theta)-y))
```

The **placeholder four**, atmp variable is trying to calculate the cost of the current theta, and to complete this logic; I relied upon the following equation:

$$F(\theta_0, \theta_1) = \frac{1}{2m} \sum_i (y_i - f_\theta(x_i))^2$$

$$= \frac{1}{2m} \sum_i (\theta_0 + \theta_1 x_i - y_i)^2$$

The **cost function** calculation was straightforward with code:

```
atmp = (1/(2*m)) * sum((np.dot(X, theta)-y)**2)
```

main_ha2.py :

I have completed both placeholders in this file. **Placeholder two** is about normalizing the data, and I have tried mean normalization. I used the mean normalization due to its better outcomes for the convergence plots, which I will discuss later in this report. I relied upon the following equation:

$$x' = \frac{x - \mu}{\max(x) - \min(x)}$$

Mean Normalization Formula

Here is the code:

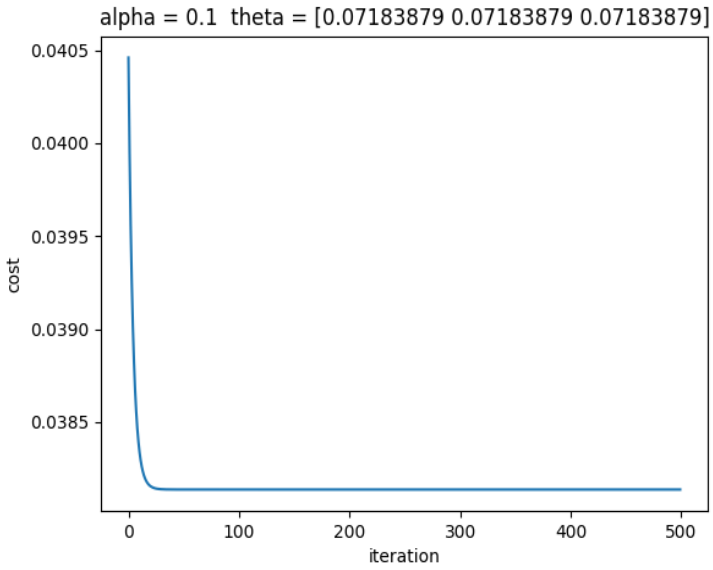
```
def meanNormalize(array):
    new_arr = np.zeros(array.shape)
    for i in range(0, 3):
        ma = max(array[:, i])
        mi = min(array[:, i])
        me = np.mean(array[:, i])
        colLen = len(array[:, i])
        for k in range(colLen):
            new_arr[k, i] = ((array[k, i]-me) / (ma-mi))
    return new_arr
```

The **placeholder one** mainly tried different learning rates (alpha) for the convergence plots.

Observances:

The different values of the **alpha** produced the following graph and data: (Mean Normalization method)

Graph 1:

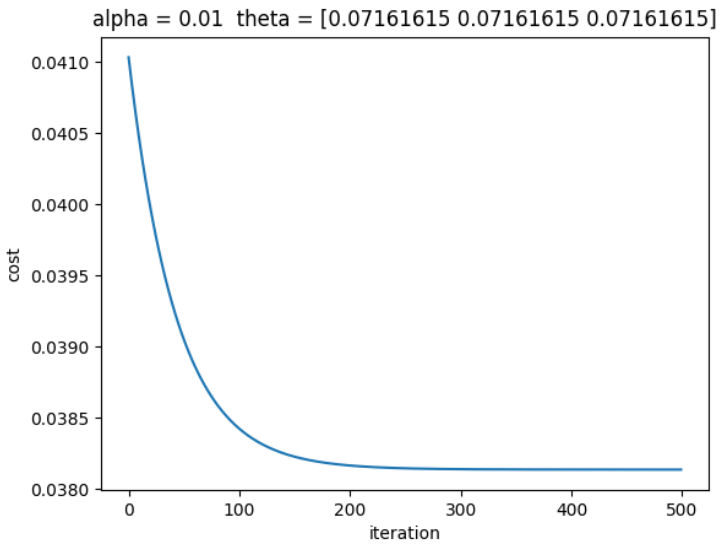


Cost of last 25 iterations produce a constant value: (for the graph above)

```
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
0.038137091407418014
```

Average error and standard deviation values: 0.1225 and 0.15355 (low error)

Graph 2:

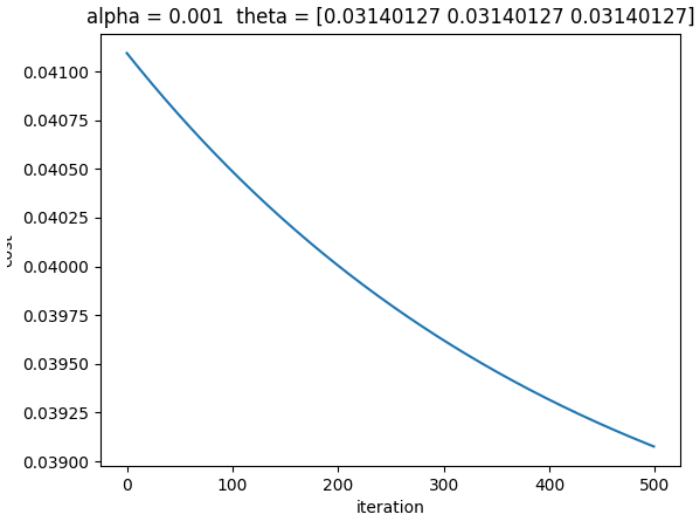


Cost of first 25 iterations: (Changing continuously, as we can also see from the graph above)

```
0.040459299516047 31
0.039956448498008765
0.03956248485548271
0.0392538301126966
0.039012011477775096
0.038822556257785745
0.03867412567718612
0.03855783625925195
0.038466728155592766
0.03839534860361092
0.038339425582095826
0.0382956121334464
0.038261286050367375
0.03823439293841504
0.038213323261784095
0.03819681601360401
0.03818388324547226
0.038173750939364946
```

Average error and standard deviation values: 0.1225 and 0.15355 (low error)

Graph 3:

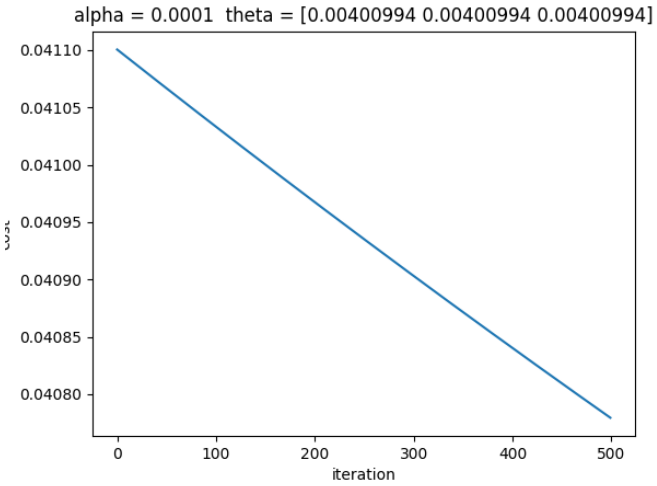


Cost of first ten iterations: (there is no convergence)

```
0.04109432770986369
0.0410875378191817 3
0.0410807635182635 4
0.04107400477131470 4
0.0410672615426228 7
0.0410605337965576 8
0.04105382149757065 6
0.0410471246101949 1
0.0410404430990449 9
0.04103377692881670 5
```

Average error and standard deviation: 0.137645 and 0.146354 (Slightly higher than the last two values)

Graph 4 :



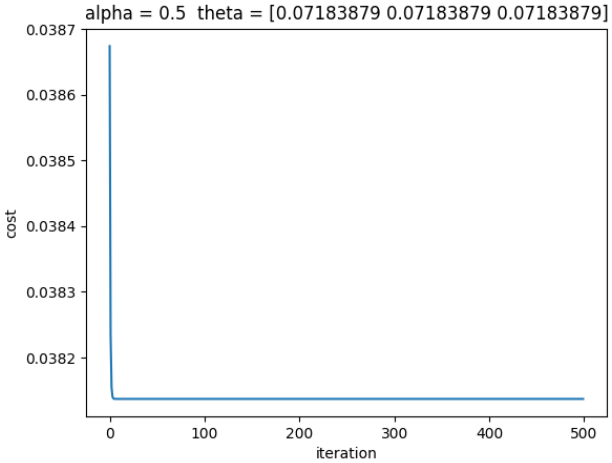
Similarly, there is no apparent convergence with an alpha value of 0.0001.

Average error and standard deviation: 0.15651 and 0.140028 (the error is highest than previous errors)

From the graphs, I observe that when we decrease the alpha value (learning rate value) by a factor of 10^n where n is a real number, we see that the rate of convergence slows down, and there is no direct convergence pattern that emerges as the number of iterations increase. However, the first two graphs with lower alpha rates of 0.1 and 0.01 produce a stable convergence pattern after 50 and 200 iterations.

Here is the pattern for an increasing value of learning rate (alpha):

Graph 5:

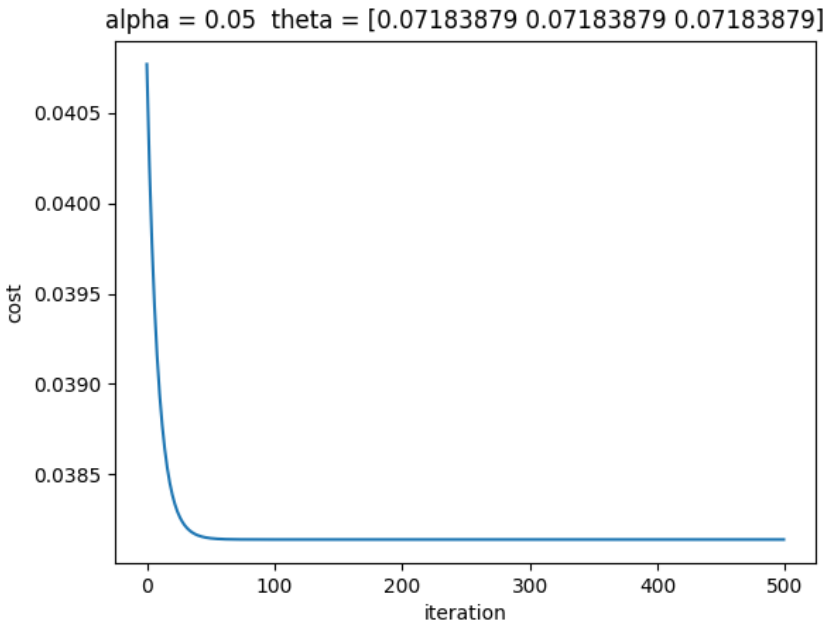


Cost after 16 iterations appears to stabilize, as we can notice from the graph above:

```
0.03867414492044489
0.03823439991164633
0.038154722695111824
0.03814028601315374
0.03813767023671192
0.03813719628525873
0.038137110410190654
0.038137094850522545
0.03813709203127268
0.03813709152045397
0.03813709142789895
0.03813709141112895
0.03813709140809039
0.03813709140753984
0.03813709140744008
```

average error and standard deviation: 0.1225 and 0.1535 (About the same value with an alpha value of 0.1)

Graph 6:



```
0.03813709140741799
0.038137091407418035
0.038137091407418
0.038137091407418
0.03813709140741799
0.038137091407418014
0.038137091407418
0.038137091407418014
0.03813709140741802
0.03813709140741803
0.038137091407418014
0.038137091407418
0.038137091407418
0.03813709140741802
0.03813709140741799
0.038137091407418014
0.038137091407418014
0.038137091407418
```

Average error and standard deviation: 0.12253 and 0.15355 (about the same)

These two graphs with lower values show that as we increase the learning rate, the cost value drops faster than having a small learning rate value. Therefore, the convergence appears to have been done a lot faster with iterations less than 100.