Khoi Duong

Prof. Yang

CS483

5/28/2022

ASSIGNMENT #1

1.

Two features data:

Hypothesis: $h_0 = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

Cost function: $J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}[(\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}) - y^{(i)}]^2$

Gradient decent algorithm:

$$\theta_0 = \theta_0 - \alpha\frac{\partial J(\theta)}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha\frac{\partial J(\theta)}{\partial \theta_1}$$

$$\theta_2 = \theta_2 - \alpha\frac{\partial J(\theta)}{\partial \theta_2}$$

We have:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}[(\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}) - y^{(i)}]$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}[(\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}) - y^{(i)}] * x_1^{(i)}$$

$$\frac{\partial J(\theta)}{\partial \theta_2} = \frac{1}{m}\sum_{i=1}^{m}[(\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}) - y^{(i)}] * x_2^{(i)}$$

Let $\theta_0 = \theta_1 = \theta_2 = 0$, $m = 14$, and learning rate $\alpha = 0.01$

We have the source below:

```
import numpy as np

age = [60,61,74,57,63,68,66,77,63,54,63,76,60,61]
weight = [58,90,96,72,62,79,69,96,96,54,67,99,74,73]
SBP = [117,120,145,129,132,130,110,163,136,115,118,132,111,112]


theta_0 = theta_1 = theta_2 = 0
alpha = 0.0001
i = 0

while i <= 500000:
  diff_theta_0 = diff_theta_1 = diff_theta_2 = 0
  for m in range (14):
    diff_theta_0 += theta_0 + theta_1*age[m] + theta_2*weight[m] - SBP[m]
    diff_theta_1 += (theta_0 + theta_1*age[m] + theta_2*weight[m] - SBP[m])
* age[m]
    diff_theta_2 += (theta_0 + theta_1*age[m] + theta_2*weight[m] - SBP[m])
* weight[m]
  diff_theta_0 = diff_theta_0 * (1/14)
  diff_theta_1 = diff_theta_1 * (1/14)
  diff_theta_2 = diff_theta_2 * (1/14)


  theta_0 = theta_0 - alpha * diff_theta_0
  theta_1 = theta_1 - alpha * diff_theta_1
  theta_2 = theta_2 - alpha * diff_theta_2
  i += 1

print("diff_theta_0  =  " + str(diff_theta_0) + ",  " + "Theta 0 = ",
str(theta_0))
print("diff_theta_1  =  " + str(diff_theta_1) + ",  " + "Theta 1 = ",
str(theta_1))
```

```
print("diff_theta_2 = " + str(diff_theta_2) + ", " + "Theta 2 = ",
str(theta_2))
print("Predict 1: " + str(theta_0 + theta_1 * 65 + theta_2 * 85))
print("Predict 2: " + str(theta_0 + theta_1 * 79 + theta_2 * 80))
```

Run program and we have the result:

```
diff_theta_0 = -0.2153836356849662, Theta 0 =  13.989151966126204

diff_theta_1 = 0.0039921016325844706, Theta 1 =  1.480154089230989

diff_theta_2 = -0.000570996057904592, Theta 2 =  0.21618190046308347

Predict 1: 128.57462930550258

Predict 2: 148.21587705242104
```

So we can see that the hypothesis function is training (runtime: 10s). And all diff_theta_0, diff_theta_1, and diff_theta_2 are close to 0. And we have the hypothesis function:

$$SBP = 13.98915 + 1.48015 * age + 0.21618 * weight$$

And we have the prediction for the two patient as below:

Patient's ID 14 (Age: 65, weight: 85) => SBP = 128.57463 mmHg

Patient's ID 15 (Age 79, weight: 80) => SBP = 148.21588 mmHg

2.

Hypothesis: $h_0 = \theta_0 + \theta_1 x + \theta_2 x^2$

Loss function: $L = [(\theta_0 + \theta_1 x^{(i)} + \theta_2 x^{(i)^2}) - y^{(i)}]^2$

Cost function: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}[(\theta_0 + \theta_1 x^{(i)} + \theta_2 x^{(i)^2}) - y^{(i)}]^2$

Gradient decent algorithm:

$$\theta_0 = \theta_0 - \alpha\frac{\partial J(\theta)}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha\frac{\partial J(\theta)}{\partial \theta_1}$$

$$\theta_2 = \theta_2 - \alpha\frac{\partial J(\theta)}{\partial \theta_2}$$

We have:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}[(\theta_0 + \theta_1 x^{(i)} + \theta_2 x^{(i)^2}) - y^{(i)}]$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}[(\theta_0 + \theta_1 x^{(i)} + \theta_2 x^{(i)^2}) - y^{(i)}] * x^{(i)}$$

$$\frac{\partial J(\theta)}{\partial \theta_2} = \frac{1}{m}\sum_{i=1}^{m}[(\theta_0 + \theta_1 x^{(i)} + \theta_2 x^{(i)^2}) - y^{(i)}] * x^{(i)2}$$

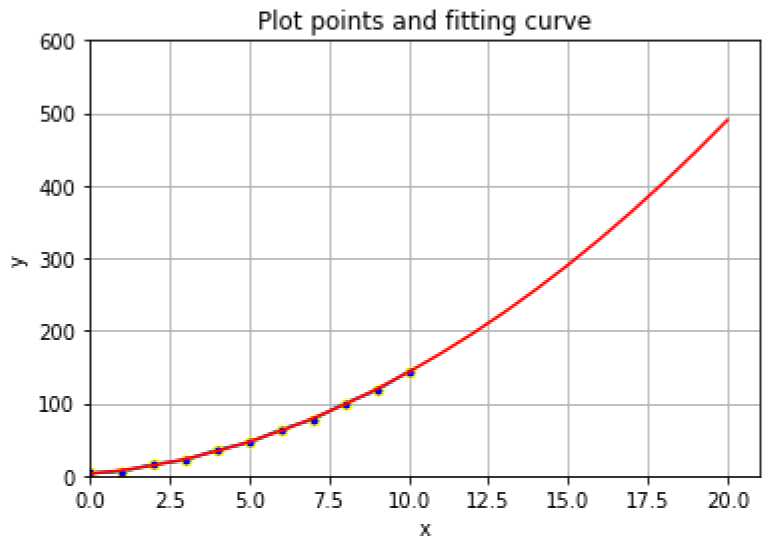
Source code:

```
import numpy as np

y = [4,5,16,21,36,45,64,77,100,117,144]


theta_0 = theta_1 = theta_2 = 0
alpha = 0.00001
i = 0

while i <= 1000000:
  diff_theta_0 = diff_theta_1 = diff_theta_2 = 0
  for m in range (11):
    diff_theta_0 += theta_0 + theta_1*m + theta_2*(m**2) - y[m]
    diff_theta_1 += (theta_0 + theta_1*m + theta_2*(m**2) - y[m]) * m
    diff_theta_2 += (theta_0 + theta_1*m + theta_2*(m**2) - y[m]) * m**2
  diff_theta_0 = diff_theta_0 * (1/11)
  diff_theta_1 = diff_theta_1 * (1/11)
  diff_theta_2 = diff_theta_2 * (1/11)
```

```
  theta_0 = theta_0 - alpha * diff_theta_0
  theta_1 = theta_1 - alpha * diff_theta_1
  theta_2 = theta_2 - alpha * diff_theta_2
  i += 1

print("diff_theta_0  =  "  +  str(diff_theta_0)  +  ",  "  +  "Theta  0  =  ",
str(theta_0))
print("diff_theta_1  =  "  +  str(diff_theta_1)  +  ",  "  +  "Theta  1  =  ",
str(theta_1))
print("diff_theta_2  =  "  +  str(diff_theta_2)  +  ",  "  +  "Theta  2  =  ",
str(theta_2))

import matplotlib.pyplot as plt

x = [i for i in range (11)]
plt.xlim(0, 21)
plt.ylim(0, 600)
plt.grid()
plt.plot(x,   y,   marker="o",   markersize=5,   markeredgecolor="yellow",
markerfacecolor="blue")

x_hypothesis = [i for i in range (21)]
y_hypothesis   =   [theta_2*(x**2)   +   theta_1*(x)   +   theta_0   for   x   in
x_hypothesis]
plt.plot(x_hypothesis, y_hypothesis, color="red", label="fitting curve")
plt.title('Plot points and fitting curve')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Run program and result:

```
diff_theta_0 = -0.04449695799932542, Theta 0 =  2.553052926474782

diff_theta_1 = 0.01802933002667925, Theta 1 =  3.6667256865770312

diff_theta_2 = -0.0014767140773634744, Theta 2 =  1.0357325692803407
```



We can see on the graph above that the blue point is the given data on the x-value and y-value.

After training regression module, we have the hypothesis function of y to x:

$$y = 1.03573x^2 + 3.66673x + 2.55305$$

3.

Using Cramer's rule, we have the source below:

```
import numpy as np

x=np.array([[1, 0, 0],
            [1, 1, 1],
            [1, 2, 4],
            [1, 3, 9],
            [1, 4, 16],
            [1, 5, 25],
            [1, 6, 36],
            [1, 7, 49],
            [1, 8, 64],
            [1, 9, 81],
            [1, 10, 100]])

x_transpose=np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                      [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
```

```
                      [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]])

y=np.array([[4],
            [5],
            [16],
            [21],
            [36],
            [45],
            [64],
            [77],
            [100],
            [117],
            [144]])

a = np.matmul(x_transpose, x)
b = np.matmul(np.linalg.inv(a), x_transpose)
c = np.matmul(b, y)
print(c)
print("Theta 0: ", float(c[0]))
print("Theta 1: ", float(c[1]))
print("Theta 2: ", float(c[2]))
```

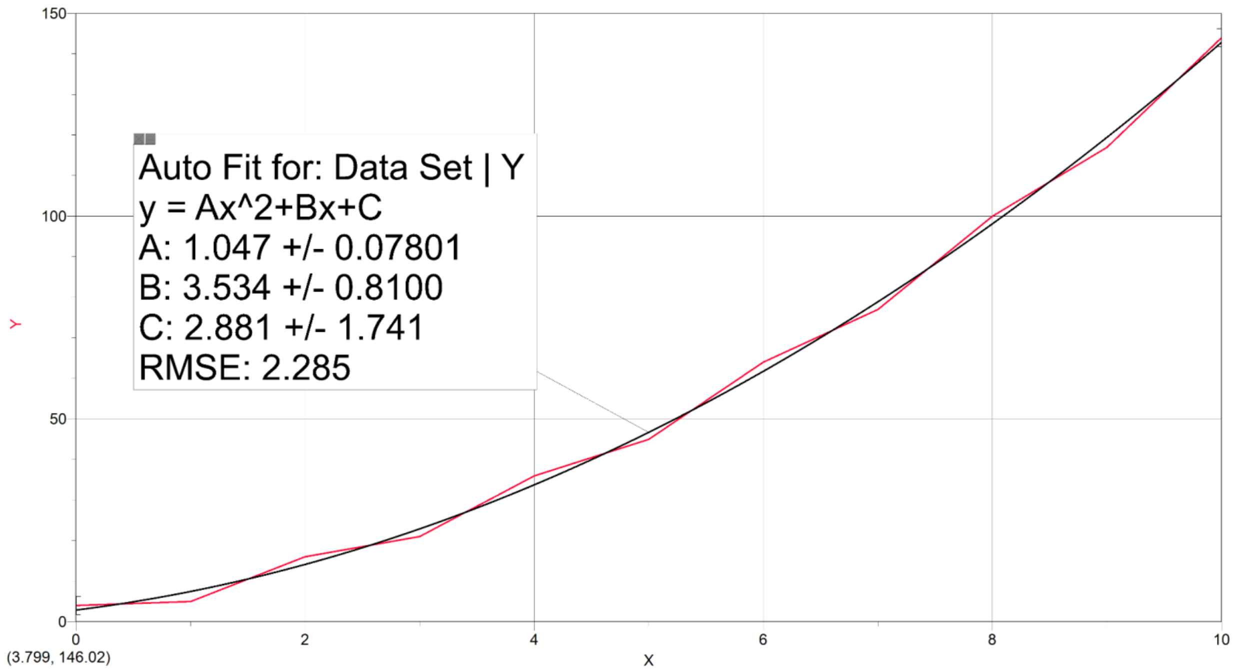
Run program and result:

```
[[2.88111888]

 [3.53379953]

 [1.04662005]]

Theta 0:  2.8811188811184643

Theta 1:  3.5337995337997015

Theta 2:  1.0466200466200402
```

The regression equation is:

$$y = 1.04662x^2 + 3.53380x + 2.88112$$

Thus, we can see that the result of the 3 thetas calculated with Cramer's rule is very close to the result calculated with gradient descent algorithm in question 2. Furthermore, we will put the data in LoggerPro and we can see the regression below:



Thus, we can see that the result calculated with Cramer's rule is more accurate than the result calculated with gradient descent algorithm in question 2.