江南大学

JIANGNAN UNIVERSITY

Machine Learning

homework3

Linear regression with one variable

Name: Abdulkadir Duran Adan

Student ID: 5035190144

Class: BCS1901

Major: computer science and technology

School: school of artificial intelligence and computer science

Problem

Please read section 2, "2 Linear regression with one variable" in the file "ex1.pdf", and finish the following tasks:

(1) Plotting the Data;

(2) Implement the Gradient Descent algorithm and find the value of $\boldsymbol{\theta}$;

(3) make predictions on profits in areas of 35,000 and 70,000 people.

Solution

We will solve this problem using python programming language as follows.

Importing all necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Opening the file (ex1.pdf) and Read the data into a pandas dataframe.

```
data_frame = pd.read_csv('/Users/boom/Desktop/homework3-
linearRegression/ex1data1.txt', names=['Population', 'Profit'])
data_frame
```

Output

Out[88]:

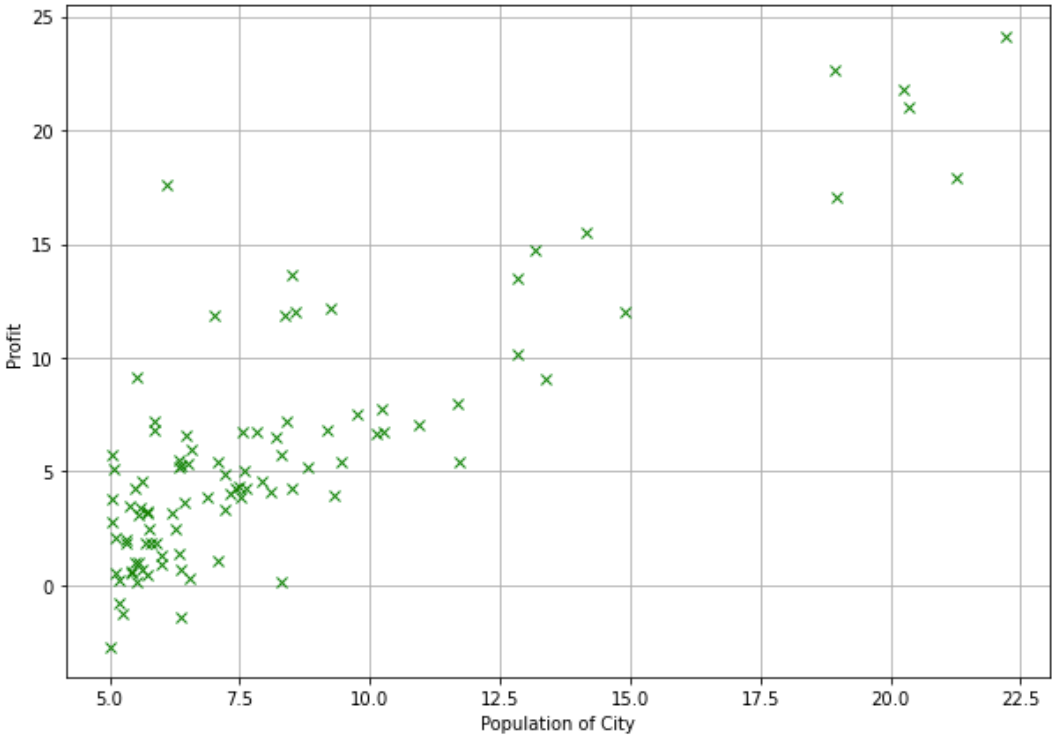
	Population	Profit
0	6.1101	17.59200
1	5.5277	9.13020
2	8.5186	13.66200
3	7.0032	11.85400
4	5.8598	6.82330
...	...	...
92	5.8707	7.20290
93	5.3054	1.98690
94	8.2934	0.14454
95	13.3940	9.05510
96	5.4369	0.61705

(1) Plotting the Data

```
%matplotlib inline

plt.figure(figsize=(10, 7))

plt.xlabel('Population of City')
plt.ylabel('Profit')
plt.grid()

plt.plot(data_frame.Population, data_frame.Profit, 'rx',
color="g")
```

Output

```
Out[107]: [<matplotlib.lines.Line2D at 0x7feae1b0eca0>]
```



(2) Implement the Gradient Descent algorithm and find the value of θ;

First, we need to Fit the linear regression parameters θ to the dataset using gradient descent and then we update the equations using linear regression to minimize the cost function.

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(h\theta(x^{(i)})-y^{(i)})^2$$

where the hypothesis $h\theta(x)$ is given by the linear model

$$h\theta(x)=\theta^T x=\theta_0+\theta_1 x_1$$

The parameters of the model are the `θj` values. These values will be adjusted to minimize cost `J(θ)`. One way to do this is to use the batch gradient descent algorithm. In batch gradient descent, each iteration performs the update

$$\theta j:=\theta j-\alpha\frac{1}{2m}\sum_{i=1}^{m}(h\theta(x^{(i)})-y^{(i)})x^{(i)}{}_{(J)}$$

Implementing the Gradient Descent algorithm and find the value of θ using python

```
# Get the number of features.
n = len(data_frame.columns)-1

# Create a function to pepare the data.
def prepareData(data, n):

    # Add a column with 1s in the data set.
    data.insert(0, 'Ones', 1)

    # Define X and y, separating the data set.
    x = data.iloc[:, 0:n+1]
    y = data.iloc[:, n+1:n+2]


    x = np.matrix(x.values)
    y = np.matrix(y.values)
    theta = np.matrix(np.zeros((n+1, 1)))
    return x, y, theta

x, y, theta = prepareData(data_frame, n)
iterations = 1500
alpha = 0.01

# Check the dimensions of the matrices.
x.shape, y.shape, theta.shape
def computeCost(x, y, theta):

    m = len(x)
    cost = np.sum(np.square((x * theta) - y)) / (2 * m)
    return cost

computeCost(x, y, theta)
```

Output

```
Out[82]: ((97, 2), (97, 1), (2, 1))

Out[83]: 32.072733877455676
```

(3) make predictions on profits in areas of 35,000 and 70,000 people.

After the correct implementation of gradient descent and compute Cost, the value of $J(\theta)$ should never increase, and should converge to a steady value by the end of the algorithm. The final parameters will be used to plot the linear fit and make predictions on profits in areas of 35,000 and 70,000 people. We minimize the value of $J(\theta)$ by changing the values of the vector θ, not by changing x or y.

Using python program

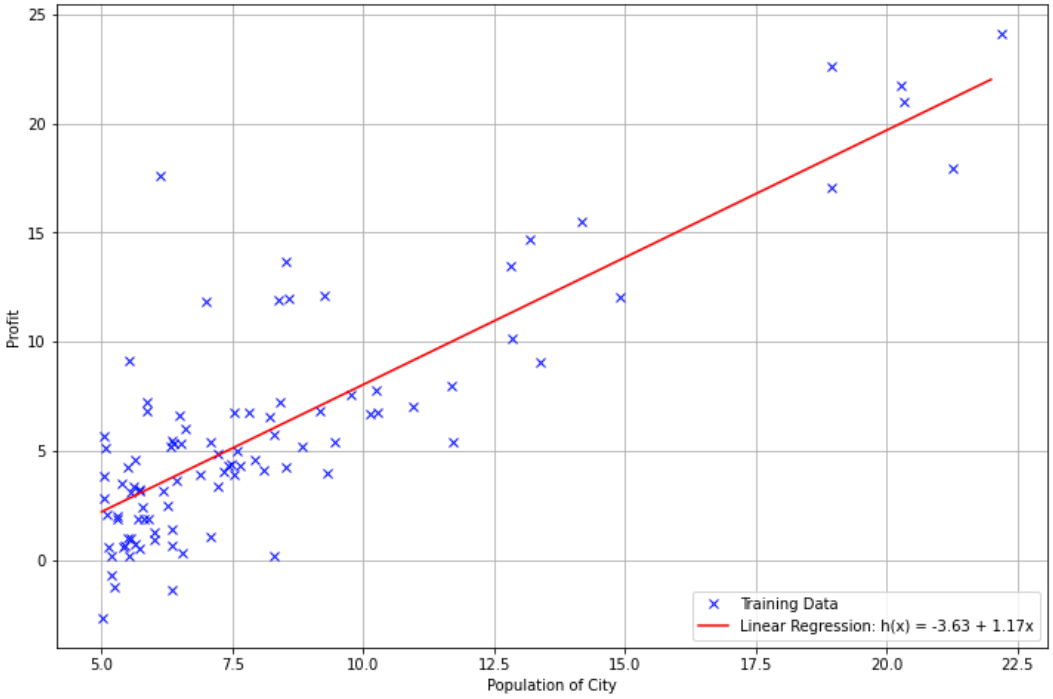
```
def gradientDescent(x, theta, iterations):

    m = len(x)
    J_vals = []

    for i in range(iterations):
        error = (x * theta) - y
        for j in range(len(theta.flat)):
            theta.T[0, j] = theta.T[0, j] - (alpha/m) *
np.sum(np.multiply(error, x[:, j]))
        J_vals.append(computeCost(x, y, theta))
    return (theta, J_vals)

theta, J_vals = gradientDescent(x, theta, iterations)
theta_f = list(theta.flat)
xs = np.arange(5, 23)
ys = theta_f[0] + theta_f[1] * xs

plt.figure(figsize=(12, 8))
plt.xlabel('Population of City')
plt.ylabel('Profit')
```

```
plt.grid()
plt.plot(data1.Population, data1.Profit, 'rx', label='Training
Data', color='blue' )
plt.plot(xs, ys, 'b-', color='r', label='Linear Regression: h(x) =
%0.2f + %0.2fx'%(theta[0], theta[1]))
plt.legend(loc=4)

print((theta_f[0] + theta_f[1] * 3.5) * 10000)
print((theta_f[0] + theta_f[1] * 7) * 10000)
```

Outputs

```
4519.767867701772
45342.45012944714
```

Out[108]: <matplotlib.legend.Legend at 0x7feae1255d00>



Visualizing $J(\theta)$

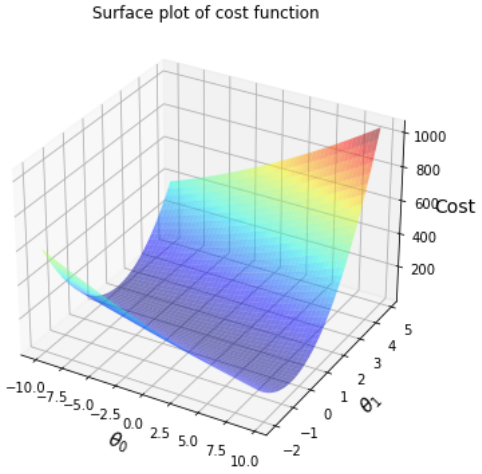
To understand the cost function J(θ) better, we will now plot the cost over a 2-dimensional grid of θ_0 and θ_1 values.

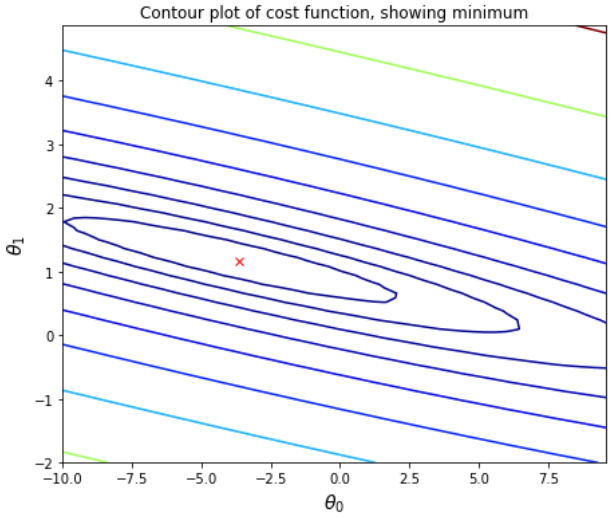
Here is the python code

```
from mpl_toolkits.mplot3d import axes3d

# Create meshgrid.
xs = np.arange(-10, 10, 0.4)
ys = np.arange(-2, 5, 0.14)
xx, yy = np.meshgrid(xs, ys)

# Initialize J values to a matrix of 0's.
J_vals = np.zeros((xs.size, ys.size))

# Fill out J values.
for index, v in np.ndenumerate(J_vals):
    J_vals[index] = computeCost(x, y, [[xx[index]], [yy[index]]])

# Create a set of subplots.
fig = plt.figure(figsize=(16, 6))
ax1 = fig.add_subplot(121, projection='3d')
ax2 = fig.add_subplot(122)

# Create surface plot.
ax1.plot_surface(xx, yy, J_vals, alpha=0.5, cmap='jet')
ax1.set_zlabel('Cost', fontsize=14)
ax1.set_title('Surface plot of cost function')

# Create contour plot.
ax2.contour(xx, yy, J_vals, np.logspace(-2, 3, 20), cmap='jet')
ax2.plot(theta_f[0], theta_f[1], 'rx')
ax2.set_title('Contour plot of cost function, showing minimum')
# Create labels for both plots.
for ax in fig.axes:
    ax.set_xlabel(r'$\theta_0$', fontsize=14)
    ax.set_ylabel(r'$\theta_1$', fontsize=14)
```

Output

After program is executed, we have a 2-D array of J(θ) values and then we use these values to produce surface and contour plots of J(θ) using the surf and contour commands and we can see the output below.





The purpose of these graphs is to show you that how J(θ) varies with changes in θ_0 and θ_1. The cost function J(θ) is bowl-shaped and has a global minimum.