Data Mining
Assignment 1: Linear Regression

Tauseef Iftikhar

October 22, 2017

1 Univariate Linear Regression

This assignment consists of following 4 files:

1. univariate_linear_regression_data.txt
2. univariate_linear_regression.py
3. costFunction.py
4. computeGradient.py
5. plotData.py

There is a text file univariate_linear_regression_data.txt which contains comma separated values. You need to run univariate_linear_regression.py at your command prompt it will load the data from the text file. You have to provide code in costFunction.py and computeGradient.py to run the program successfully.

1.1 Loading the Data

In univarite_linear_regression.py, the dataset is loaded from the data file into the variables x and y:

```
# moving first column  lableled 'x' in variable x as a
    Series
x = df['x']
# moving first column lableled 'y' in variable y as a
    Series
y = df['y']
```

Now $\mathbf{x} \in \mathcal{R}^m$ and $\mathbf{y} \in \mathcal{R}^m$

1.2 Plotting the Data

Before starting on any task, it is often useful to understand the data by visualizing it. For this dataset, you can use a scatter plot to visualize the data, since it has only two properties to plot. (Many other problems that you will encounter in real life are multi-dimensional and can't be plotted on a 2-d plot.)

Next, the script calls the plotData function to create a scatter plot of the data. Your job is to complete plotData.py to draw the plot; modify the file and fill in the following code:

mp.plot(x,y,'o')

Now, when you continue to run univarite_linear_regression.py, our end result should look like

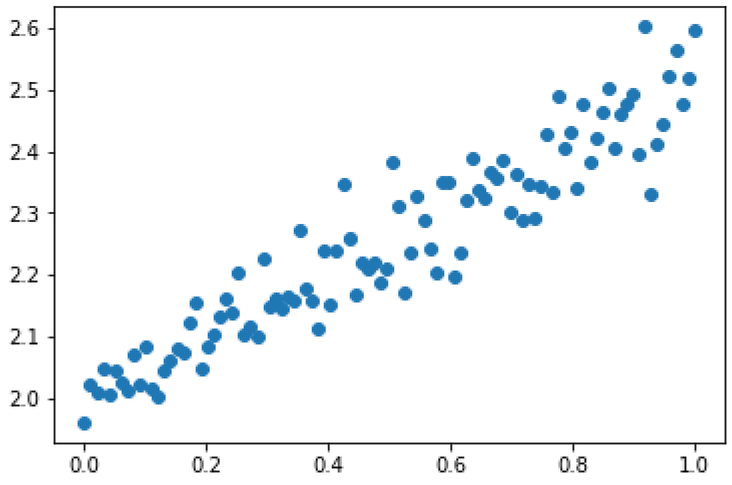

Figure 1: Data Visualization

1.3 Implementing Cost Function

Before computing cost it is necessary to compute $h_\theta(x^{(i)})$. We know that

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1(x^{(i)}) \tag{1}$$

Computing eq(1) using vectorized form is more efficient in MATLAB as well as in Python. To vectorise eq(1) we introduce x_o in eq(**??**) where $x_0 = 1$.

$$h_\theta(x^{(i)}) = \theta_0 x_0^{(i)} + \theta_1(x_1^{(i)})$$

Now $\mathbf{x}^{(i)} \in \mathcal{R}^2$ and $\boldsymbol{\theta} \in \mathcal{R}^2$. Hence $h_\theta(x^{(i)})$ can be computed as

$$h_\theta(x^{(i)}) = \boldsymbol{\theta}^T \mathbf{x} \tag{2}$$

Now if there are m training examples then there should be m values for each $h_\theta(x^{(1)}), h_\theta(x^{(2)}), ..., h_\theta(x^{(m)})$. It can be written as a vector $h_\theta(\mathbf{x}) \in \mathcal{R}^m$.

Practically in our code we have $\mathbf{x}$ which holds the input data without 1. We can introduce a column of 1's at the beginning of $\mathbf{x}$ In the following code, in line one we have made a vector named *ones* where *ones* $\in \mathcal{R}^m$. Line 5 in this figure adds this column of 1's in front of $\mathbf{x}$ yielding **new_x** $\in \mathcal{R}^{m\times 2}$.

```
1 ones = np.ones(M)
2 # adding column of 1's in front of x (Note: each 1 is
    representing
3 # x_o in an example of input training data.)
4 new_x = np.column_stack((ones,x))
5 #initialize theta with 0's (Size of theta should be equal
    to
6 #number of attributes in x in training examples.)
7 N = new_x.shape[1]
8 theta = np.zeros(N)
```

In line 7 we get the number of columns in our input data x. In line 8 we have set a vector **theta** $\in \mathcal{R}^2$ initialized with 0's. It is to be noted, each row vector $\mathbf{x}^{(i)}$ in matrix **new_x** is a single training example. Now it can be seen that a simple vector multiplication of **new_x** by **theta** will yield $h_\theta(\mathbf{x})$ We

	x_0	x_1
$\mathbf{x}^{(1)}$	1	0.0
$\mathbf{x}^{(2)}$	1	0.10
$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}^{(m)}$	1	1

Table 1: **new_x**

have to implement the sum of squared error function in costFunction.py file. The function we designed is

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{3}$$

You should implement eq(**??**) in vector form for efficiency reasons. $\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ part of eq(3) can be vectorized as follows. We know vector $\mathbf{y} \in \mathcal{R}^m$ and $h_\theta(\mathbf{x}) \in \mathcal{R}^m$. Error can be computed simply by subtracting vector from vector. Now square and sum can be computed be dot product of Error with Error vector. In case of correct implementation your cost should be 2.550155155868016.

1.4 Implementing Gradient Descent

Last objective is to implement gradient descent algorithm. You should implement the following equations after taking gradient.

$$\theta_0 = \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0$$

$$\theta_1 = \theta_1 - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1$$

Where $\theta_0, and \theta_1$ should be updated simultaneously.

In general you can implement following single equation with proper vectorization to compute $\boldsymbol{\theta}^*$ where $\boldsymbol{\theta}^* \in \mathcal{R}^2$

repeat until converge {

$$\theta_j = \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j \tag{4}$$

} If you implement it correctly the final result will look like.

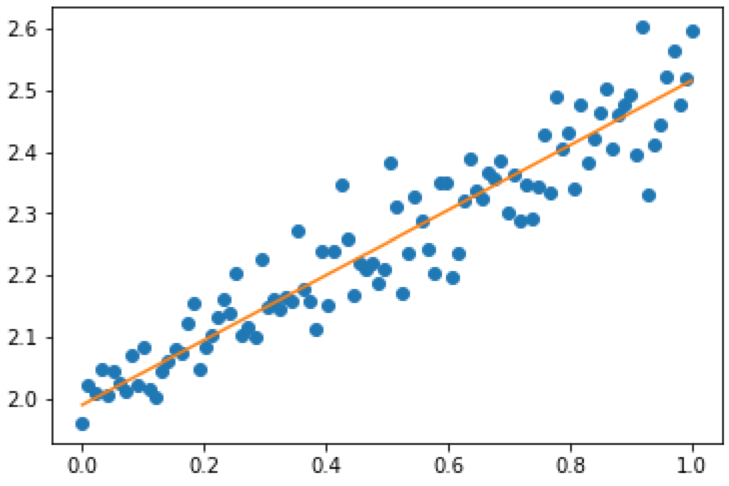

Figure 2: Univariate Linear Regression

1.5 Debugging

For debugging purposes there is a graph between number of iterations and cost as shown in Figure 3. You may perform different experiments by changing the numbers of *iteration* and *alpha* in univariate_linear_regression.py and you must try it to understand the effect of alpha on learning.

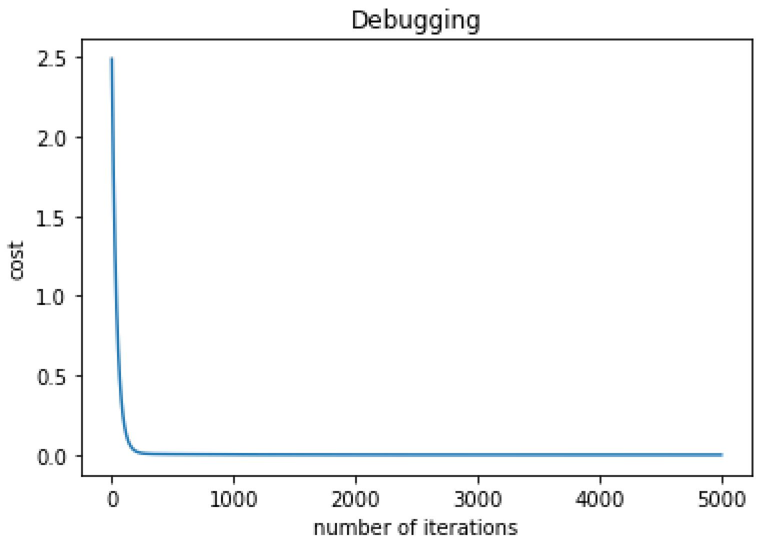

Figure 3: debugging

2 Submission

Submit a zip folder named according to your roll number at the link given for submission on course page. Folder must contain only three files: plotData.py, costFunction.py, computeGradient.py.