

Laboratory #4

Linear regression with one variable

In this part of this exercise, you will implement linear regression with one variable to predict profits for a food truck. Suppose you are the CEO of a restaurant franchise and are considering different cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from the cities.

You would like to use this data to help you select which city to expand to next.

The file *ex1data1.txt* contains the dataset for our linear regression problem. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss.

The *ex1.py* script has already been set up to load this data for you.

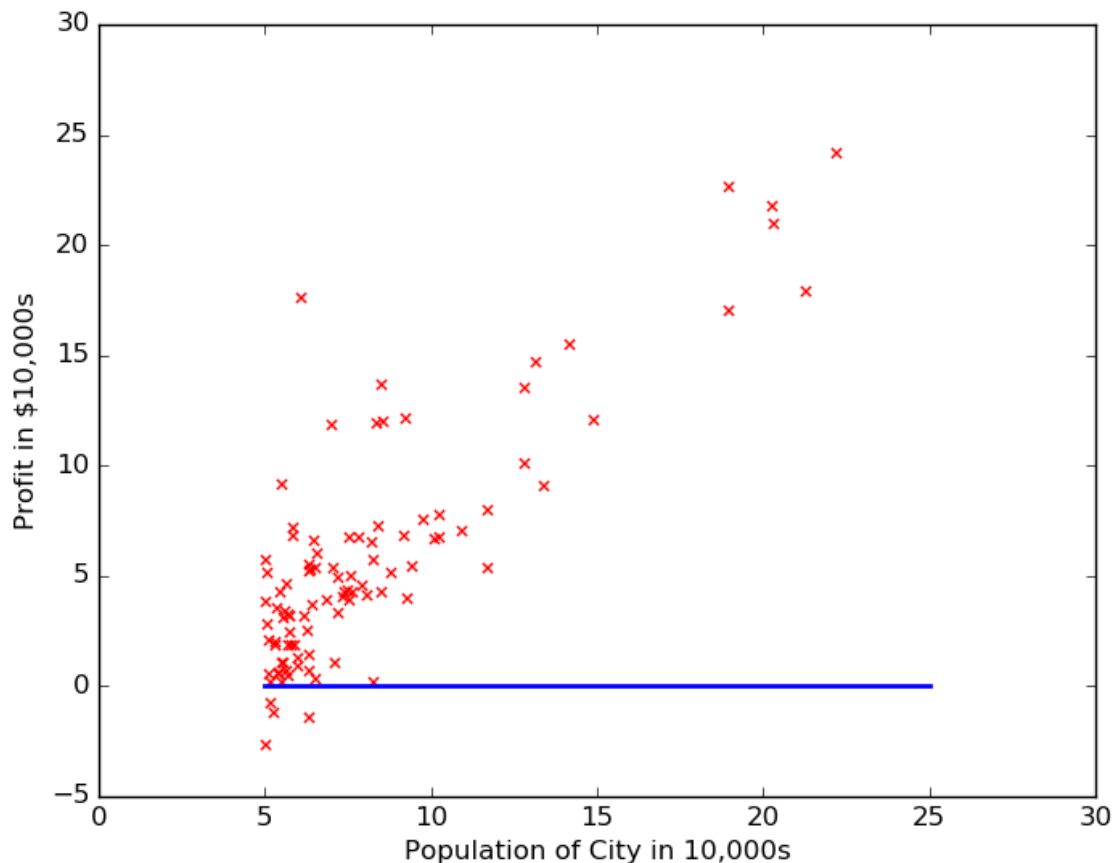


Figure 1

1. Gradient Descent

In this part, you will fit the linear regression parameters θ to our dataset using gradient descent.

1.1 Update Equations

The objective of linear regression is 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

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

Recall that the parameters of your model are the θ_j values. These are the values you will adjust to minimize cost $J(\theta)$. One way to do this is to use the batch gradient descent algorithm. In batch gradient descent, each iteration performs the update

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (\text{simultaneously update } \theta_j \text{ for all } j).$$

With each step of gradient descent, your parameters θ_j come closer to the optimal values that will achieve the lowest cost $J(\theta)$.

1.2 Implementation

In `ex1.py`, we have already set up the data for linear regression. In the following lines, we add another dimension to our data to accommodate the θ_0 intercept term. We also initialize the initial parameters to 0 and the learning rate α to 0.01.

1.3 Computing the cost $J(\theta)$

As you perform gradient descent to learn minimize the cost function $J(\theta)$, it is helpful to monitor the convergence by computing the cost. In this section, you will implement a function to calculate $J(\theta)$ so you can check the convergence of your gradient descent implementation.

Your next task is to complete the code in the function `computeCost()`, which computes $J(\theta)$. As you are doing this, remember that the variables X and y are not scalar values, but matrices whose rows represent the examples from the training set.

Once you have completed the function, the next step in `ex1.py` will run `computeCost` once using θ initialized to zeros, and you will see the cost printed to the screen.

You should expect to see a cost of 32.07.

1.4 Gradient descent

Next, you will implement gradient descent in the function `gradientDescent()`. The loop structure has been written for you, and you only need to supply the updates to θ within each iteration.

As you program, make sure you understand what you are trying to optimize and what is being updated. Keep in mind that the cost $J(\theta)$ is parameterized by the vector θ , not X and y . That is, we minimize the value of $J(\theta)$ by changing the values of the vector θ , not by changing X or y . Refer to the equations in this handout and to the video lectures if you are uncertain.

A good way to verify that gradient descent is working correctly is to look at the value of $J(\theta)$ and check that it is decreasing with each step. The starter code for `gradientDescent()` calls `computeCost` on every iteration and prints the cost. Assuming you have implemented gradient descent and `computeCost` correctly, your value of $J(\theta)$ should never increase, and should converge to a steady value by the end of the algorithm.

After you are finished, `ex1.py` will use your final parameters to plot the linear fit. The result should look something like Figure 2:

Your final values for θ will also be used to make predictions on profits in areas of 50,000 and 250,000 people. Note the way that the following lines in `ex1.py` uses matrix multiplication, rather than explicit summation or looping, to calculate the predictions.

```
predict1 = np.dot([1, 5], theta)
```

```
predict2 = np.dot([1, 25], theta)
```

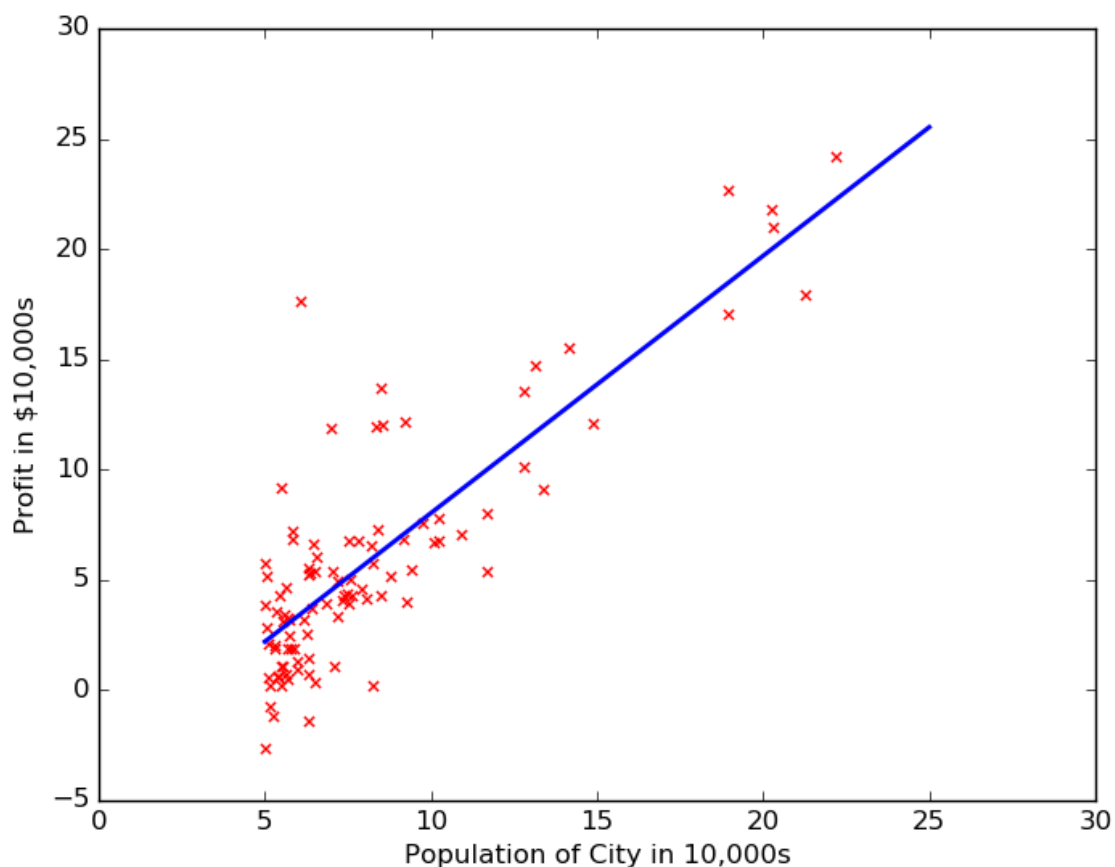


Figure 2