Programming Exercise 1: Linear Regression

Machine Learning

Introduction

In this exercise, you will implement linear regression and get to see it work on data. To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise.

Files included in this exercise

main.py – Main python script that steps you through the exercise
ex1data1.txt - Dataset for linear regression with one variable
ex1data2.txt - Dataset for linear regression with multiple variables

[?] *def computeCost(X, y, theta)* - Function to compute the cost of linear regression
[?] *def gradientDescent(X, y, theta, alpha, iters)* - Function to run gradient descent

? – indicates the methods you should to complete.

1. Where to get help

The exercises in this course use Python, a high-level programming language well-suited for numerical computations.
Python documentation can be found at https://www.python.org/ . We also strongly encourage using the online Discussions to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

2. 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.m script has already been set up to load this data for you.

2.1 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 (profit and population). (Many other problems that you will encounter in real life are multi-dimensional and can't be plotted on a 2-d plot.) In *main()*, the dataset is loaded from the data file:

```
path = 'ex1data1.txt'
data = pd.read_csv(path, header=None, names=['Population', 'Profit'])
```

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

```
data.plot(kind='scatter', x='Population', y='Profit', figsize=(12,8))
plt.show()
```

Now, when you continue to run main.py, our end result should look like Figure 1, with the same red "x" markers and axis labels.

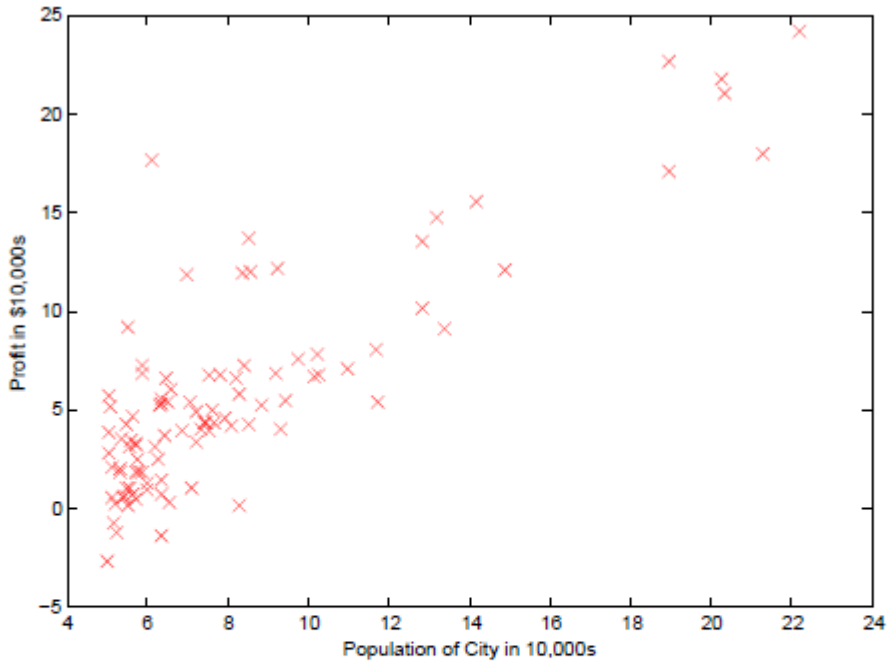


Figure 1: Scatter plot of training data

2.2 Gradient Descent

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

First, we split main features and label:

```
cols = data.shape[1]
X = data.iloc[:,0:cols-1]
y = data.iloc[:,cols-1:cols]
```

and convert it to numpy matrix and vector:

```
# convert pandas dataframe to numpy matrix
X = np.matrix(X.values)
y = np.matrix(y.values)
```

Then we should define initial values of Theta:

```
theta = np.matrix(np.array([0,0]))
```

Now we ready to begin main part:

2.2.1 Update Equations

The objective of linear regression is to minimize the cost function

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

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 $J(\theta)$.

a) Implement *computeCost method*. **Your cost at zero initialization must be equal to 32.07**
b) Implement *gradientDescent method.* **Your Final cost** (after gradient descent) **must be equal to 4.52**

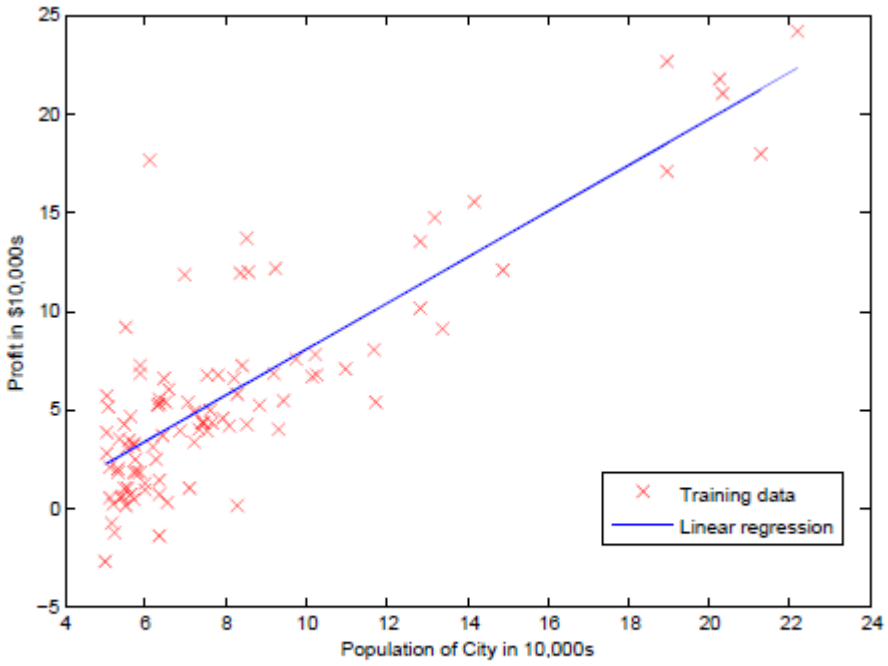
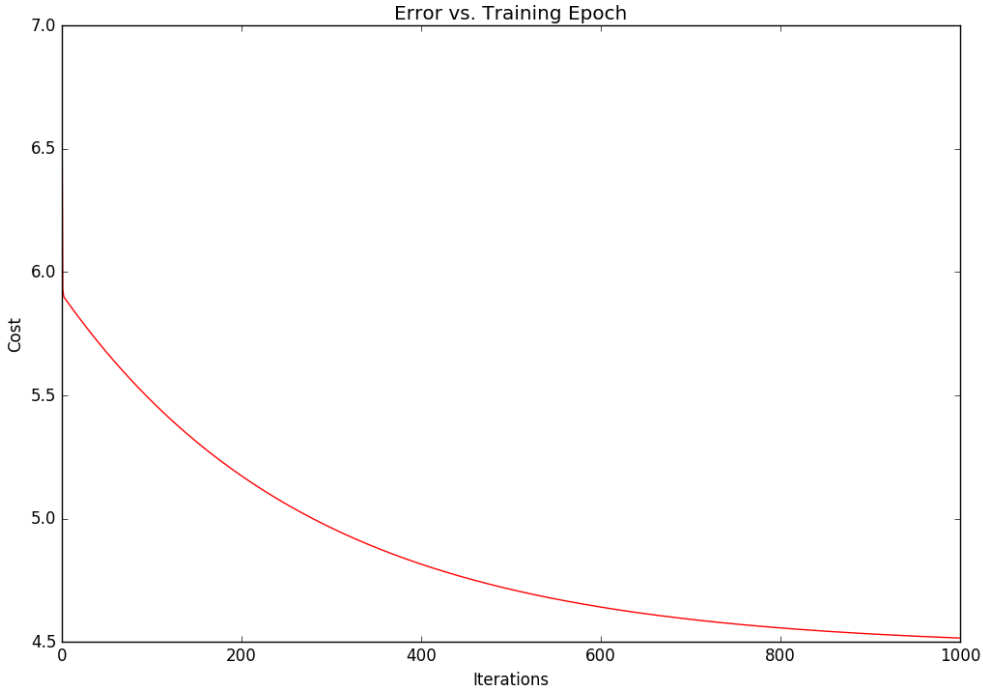


Figure 2: Training data with linear regression fit

Cost function value progress:



2. Linear regression with multiple variables

Final cost function value must be equal to (after gradient descent) **0.13**

Cost function value progress: