CENG 463 Spring 2017

Programming Exercise 2:
Linear Regression using Gradient Descent and Normal Equation

Introduction

In this exercise, you will implement univariate linear regression (i.e. linear regression with one variable). You will use both gradient descent method and normal equation method to reach the optimal parameters. The Python code to be prepared will be able to do predictions using new input.

Files included in this exercise

[**?**] HW2.py - Python script that will help you through the exercise.
population_vs_profit.data - Dataset for Exercise 2.
? indicates the files you will need to complete and submit

Throughout the exercise, you will be using the script HW2.py. This script will help you through implementation of the linear regression algorithm. For this homework, you will need to modify and submit HW2.py. Please also write your name and student number into HW2.py.

Submission

You will submit the homework via CMS. There is a homework submission link, clicking to which will lead you a file browser that you can choose your file. Please zip your files before submission and name the file as **yourstudentno-hw2.zip**. Submit the homework until **14 April 2017** 23:55. Late submissions will not be evaluated.

Problem Description

In this exercise, you will implement linear regression to predict profits for a restaurant chain. Suppose you are the director and considering different cities for opening a new restaurant. The chain already has restaurants in various cities and you have data for profits and populations from the cities. The population_vs_profit.data contains the dataset for our linear regression problem. The first column is the population of a city and it is the only variable/feature we have. The second column is the profit of a restaurant in that city. A negative value for profit indicates a loss.

Part 1: Loading data

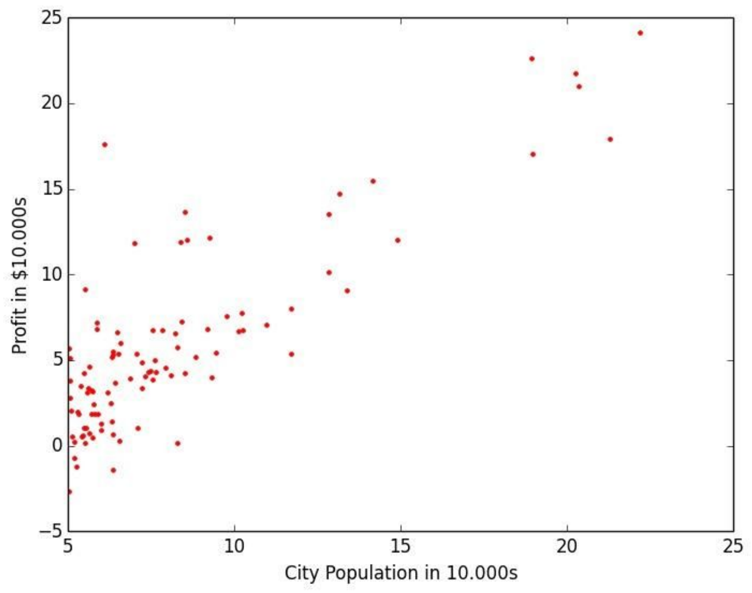
Please write your code to the place indicated in HW2.py to load the data in population_vs_profit.data. Data consists of 97 samples. Data matrix has two columns, the first column refers to our only variable/feature, and the second column contains the corresponding y value.

Part 2: Plotting data

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 2D plot.)

Please read your feature (city population) from the first column of the data matrix and use it for the x axis of your plot. Read the corresponding profit values from the second column and use it for the y axis of your plot. Write your code to the place indicated in HW2.py.

The output should be as follows:



Part 3: Gradient descent

In this part, you will calculate the gradient steps for the linear regression parameters *θ* (theta). The code to do this will be written in computeStep function which will be used during gradient descent iterations. You need to compute a gradient descent step for each parameter and in this example you have two parameters (θ_0 and θ_1).

Part 4: Computing the cost

The objective of linear regression is to minimize the cost function *J(θ)*:

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

where *m* is the number of training samples the hypothesis $h_\theta(x)$ is given by the linear model:

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

In this section, you will implement a function to calculate *J(θ)* so you can check the convergence of your gradient descent implementation. Your task is to complete the code in the computeCost function that computes *J(θ)* for a particular *θ* vector, when *X* and *y* are given. As you are doing this, remember that the variables *X* and *y* are not scalar values, but matrices/vectors whose rows represent the examples from the training set. You can use $X{\cdot}\theta$ or $\theta^T X$ to compute the hypothesis depending on the size of your matrices/vectors.

Part 5: Preparing the data

Before using the functions implemented above, we need to prepare our data. As we have seen in the class, to vectorize our computations, we need to add a new feature (x_0=1) to our feature matrix to accommodate θ_0. In this way, we can compute our hypothesis with a simple multiplication: $X{\cdot}\theta$. Note: if your feature vector was horizontal, same multiplication would be obtained by $\theta^T X$. Essentially, both computes the same thing, our hypothesis.

Currently, your feature values are in a vector like the $X = \begin{bmatrix} 6.11 \\ 5.52 \\ \vdots \end{bmatrix}$ following:

After adding the new feature (x_0=1) to all samples, it becomes: $X = \begin{bmatrix} 1 & 6.11 \\ 1 & 5.52 \\ \vdots & \vdots \end{bmatrix}$

So, add a column of ones to the left of your feature vector. Write your code to the place indicated in HW2.py.

Part 6: Linear regression using gradient descent

We first initialize the parameters (θ) to 0 before starting gradient descent. The number of iterations is set to 1500. A set of values is given for alpha to experiment with. These are already written in the code. Write the code to compute the initial cost to be printed.

Recall that the parameters of your model are the θ_j values. These are the values you will adjust to minimize cost $J(\theta)$. In gradient descent, each iteration performs an update on all θ_j values, so that with each step of gradient descent, your parameters θ_j come closer to the optimal values that will achieve the lowest cost $J(\theta)$.

At this step, you will implement the parameter update. The loop structure has been written for you, and you only need to supply the updates to θ f each iteration using the computeStep function implemented in Part 3.
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 above and to the lecture notes if you are uncertain.

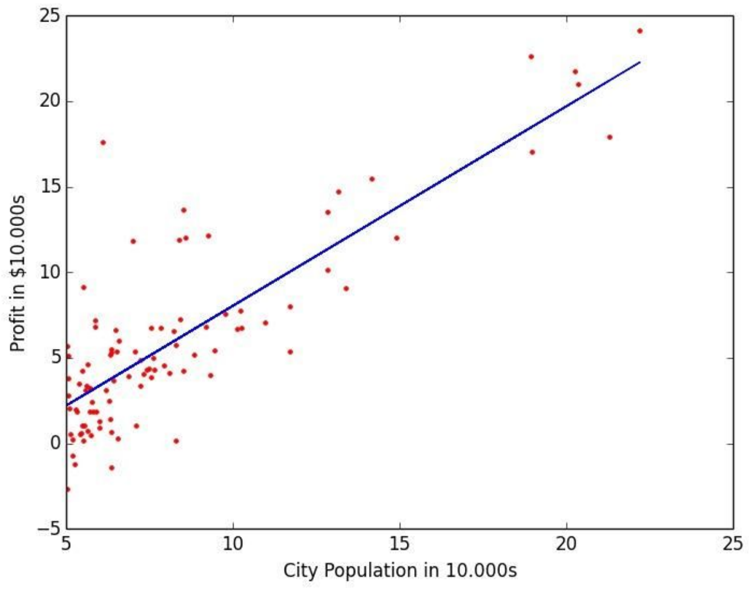
It is very important to update θ_j values simultaneously. That is, you first have to compute the gradients for both θ_0 and θ_1. Then you update θ_0 and θ_1 at the same time. If you happen to update θ_0 and use it to compute gradient for θ_1 it will not work!

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 given code calls computeCost on every iteration and saves them in an array called `J_history`.

Assuming you have implemented gradient descent and computeCost correctly, if the alpha is chosen properly, the cost, $J(\theta)$, should never increase, and should converge to a steady value by the end of the algorithm. The code in HW2.py also contains the lines to plot `J_history` for different alpha values and to print final cost values.

Part 7: Plotting the linear fit and making predictions for new data

Analyzing the printed final costs and the plotted cost history, select one of the final_theta values that performs best. Using this theta value, the linear fit will be plot. The plot should look like:

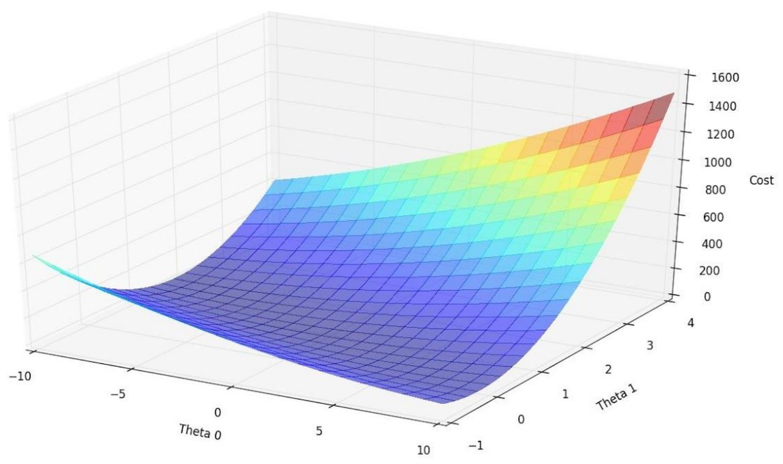


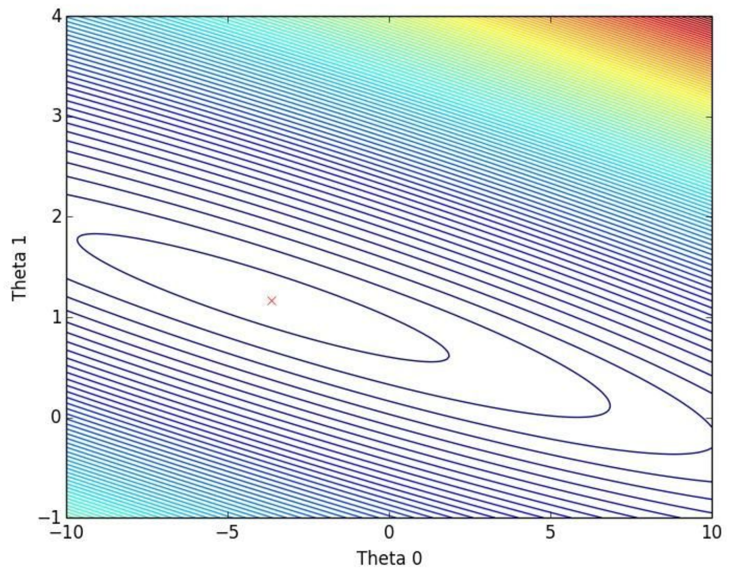
Your final values for θ will be used to make predictions on profits in new areas. Please use your optimized theta values and predict the profit in two new cities having 35,000 and 70,000 population respectively. Pay attention, we trained our hypothesis such that it uses population in 10000's and predict the profit in 10000 US dollars. You will need to convert the units.
Write your code to the place indicated in HW2.py.

Part 8: Visualizing J

To understand the cost function better, you will now plot the cost over a 2-dimensional grid of θ_0 and θ_1 values. You do not need to code anything for this part, but you should understand how the code you have written already is creating these images.

After these lines are executed, you will have a 2-D array of $J(\theta)$ values. The script in HW2.py will then use these values to produce surface and contour plots of $J(\theta)$. The plots should look something like:





The purpose of these graphs is to show you that how $J(\theta)$ varies with changes in θ_0 and θ_1. The cost function $J(\theta)$ is bowl-shaped and has a global minimum. (This is easier to see in the contour plot than in the 3D surface plot). This minimum is the optimal point for θ_0 and θ_1, and each step of gradient descent moves closer to this point.

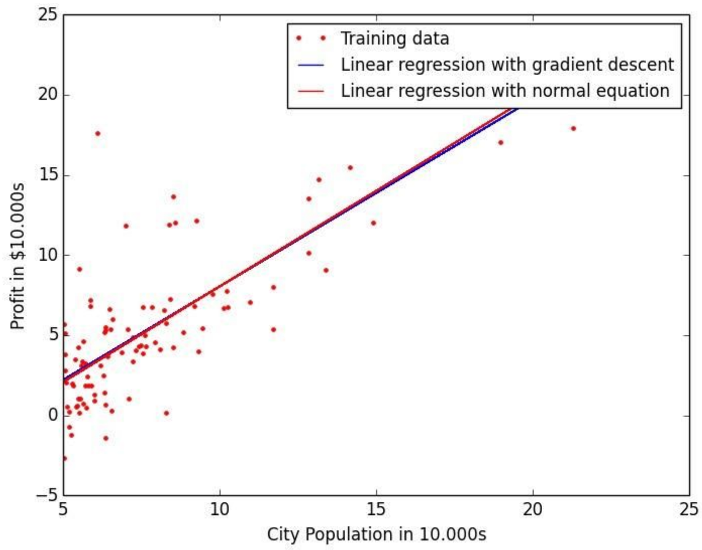
Part 9: Normal Equation Method

We learned that the closed-form solution to linear regression is:

$$\theta = (X^T X)^{-1} X^T y$$

Complete the code in HW2.py to use the formula above to calculate θ. In numpy, you can use numpy.linalg.pinv() command to compute $(X^TX)^{-1}X^T$.

After this, the script in HW2.py will plot the new hypothesis onto Figure 1. The new hypothesis (plotted in red) is quite close to the one obtained with gradient descent but not exactly the same. What you will see is something like this:



If you get this graph, congratulations! You have completed Homework 2!