Programming Exercise 2: Linear Regression (multivariate)

Introduction

In this exercise, you will implement linear regression with multiple variables 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. If needed, use the cd command in Octave/MATLAB to change to this directory before starting this exercise.
For this assignment, you will implement linear regression with multiple variables to predict the prices of houses. Suppose you are selling your house and you want to know what a good market price would be. One way to do this is to first collect information on recent houses sold and make a model of housing prices.
The file `ex2data1.txt` contains a training set of housing prices in Portland, Oregon. The first column is the size of the house (in square feet), the second column is the number of bedrooms, and the third column is the price of the house. The `ex2.m` script has been set up to help you step through this exercise.

Files included in this exercise

`ex2data1.txt` - Dataset for linear regression with multiple variables

[*] `ex2.m` - Octave/MATLAB script that steps you through the exercise

[*] `featureNormalize.m` – Function to normalize features

[*] `computeCostMulti.m` - Function to compute the cost of multiple variable linear regression

[*] `gradientDescentMulti.m` - Function to run gradient descent for multiple variables

[*] `normalEqn.m` - Function to compute the normal equations

Throughout the exercise, you will be using the script `ex2.m`. This script sets up the dataset for the problems and makes calls to functions that you will write. You do not need to modify `ex2.m`. You are only required to modify functions in other files, by following the instructions in this assignment.

---

[*] indicates files you will need to complete

[1] Octave is a free alternative to MATLAB. For the programming exercises, you are free to use either Octave or MATLAB.

1 Simple Octave/MATLAB function

The `ex2.m` script will start by loading and displaying some values from this dataset. By looking at the values, note that house sizes are about 1000 times the number of bedrooms. When features differ by orders of magnitude, first performing feature scaling can make gradient descent converge much more quickly.
Your task here is to complete the code in `featureNormalize.m` to:

- Subtract the mean value of each feature from the dataset.
- After subtracting the mean, additionally scale (divide) the feature values by their respective “standard deviations."

The standard deviation is a way of measuring how much variation there is in the range of values of a particular feature; this is an alternative to taking the range of values (max-min). In Octave/MATLAB, you can use the “`std`" function to compute the standard deviation. For example, inside `featureNormalize.m`, the quantity `X(:,1)` contains all the values of x_1 (house sizes) in the training set, so `std(X(:,1))` computes the standard deviation of the house sizes. At the time that `featureNormalize.m` is called, the extra column of 1's corresponding to $x_0 = 1$ has not yet been added to `X` (see `ex2.m` for details).
You will do this for all the features. Note that each column of the matrix `X` corresponds to one feature.

Implementation Note: When normalizing the features, it is important to store the values used for normalization - the *mean value* and the *standard deviation* used for the computations. After learning the parameters from the model, we often want to predict the prices of houses we have not seen before. Given a new `x` value (living room area and number of bedrooms), we must first normalize `x` using the mean and standard deviation that we had previously computed from the training set.

Note that:

- Expected mean value for "house size" feature is approximately 2000
- Expected mean value for "number of bedrooms" feature is approximately 3.17
- Expected standard deviation for "house size" feature is approximately 794.7
- Expected standard deviation for "number of bedrooms" feature is approximately 0.76

2 Gradient Descent

In programming exercise 1, you implemented gradient descent on a univariate regression problem. The only difference now is that there is one more feature in the matrix X. The hypothesis function and the gradient descent update rules remain unchanged.
You should complete the code in `computeCostMulti.m` and `gradientDescentMulti.m` to implement the cost function and gradient descent for linear regression with multiple variables. If your code from previous programming exercise (single variable) already supports multiple variables, you can use it here too.
Please be advised that you are required to implement the vectorized version of these two functions in this programming assignment.
After implementing `computeCostMulti.m` and `gradientDescentMulti.m`, run the `ex2.m` script to run gradient descent until convergence to find the final values of θ. Next, use this value of θ to predict the price of a house with 1650 square feet and 3 bedrooms. You will use this value later to check your implementation of the normal equations. Don't forget to normalize your features when you make this prediction!

3 Normal Equations

In the lecture videos, you learned that the closed-form solution to linear regression is

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

Using this formula does not require any feature scaling, and you will get an exact solution in one calculation: there is no “loop until convergence" like in gradient descent.
Complete the code in `normalEqn.m` to use the formula above to calculate θ. Remember that while you don't need to scale your features, we still need to add a column of 1's to the X matrix to have an intercept term (θ_0). The code in `ex2.m` will add the column of 1's to X for you.
Now, once you have found θ using this method, use it to make a price prediction for a 1650-square-foot house with 3 bedrooms. You should find that gives the same predicted price as the value you obtained using the model fit with gradient descent (in Section 2).