Programming Exercise 2: Linear Regression with Multiple Variables

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. If needed, use the cd command in Octave/MATLAB to change to this directory before starting this exercise.

You can log into your CougarNet and download MATLAB from this website: https://uh.edu/software-downloads/index.php.

Files included in this exercise

ex2data.txt - Dataset for linear regression with multiple variable
computeCostMulti.m - Cost function for multiple variables
[y] ex2.m - Octave/MATLAB script that steps you through the exercise
[y] featureNormalize.m - Function to normalize features
[y] gradientDescentMulti.m - Gradient descent for multiple variables
[y] normalEqn.m - Function to compute the normal equations
y indicates files you will need to complete

Files needed to be submit

[1] ML_ex2 – Include all the code (You need to complete ex2.m, featureNormalize.m, gradientDescentMulti.m, normalEqn.m by yourself)

[2] ex2_report – Directly give the answers of first three questions:

(1) The mean value and the standard deviation for ex2data

(2) Gradient descent's result after 400 iterations with α=0.1
Predict the price of a 1500 sq-ft, 5 br house

(3) Normal equation's result
Predict the price of a 1500 sq-ft, 5 br house

(4) Find the best learning rate(optional)

Throughout the exercise, you will be using the scripts ex2.m. This script set up the dataset for the problems and make calls to functions that you will write. You are required to modify it and some other functions, by following the instructions in this assignment.

Where to get help

The exercises in this course use Octave[1] or MATLAB, a high-level programming language well-suited for numerical computations.

At the Octave/MATLAB command line, typing help followed by a function name displays documentation for a built-in function. For example, help plot will bring up help information for plotting. Further documentation for Octave functions can be found at the Octave documentation pages. MAT- LAB documenttation can be found at the MATLAB documentation pages.

Do not look at any source code written by others or share your source code with others.

---

1 Linear regression with multiple variables

In this part, 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 ex2data.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.

1.1 Feature Normalization

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

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

- 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 (most data points will lie within ±2 standard deviations of the mean); 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 x1 (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 x0 = 1 has not yet been added to X (see `ex2.m` for details).

You will do this for all the features and your code should work with datasets of all sizes (any number of features / examples). 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.

1.2 Gradient Descent

Previously, 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 batch gradient descent update rule remain unchanged.

You should complete the code in `gradientDescentMulti.m` to implement the cost function and gradient descent for linear regression with multiple variables. If your code in the previous part (single variable) already supports multiple variables, you can use it here too.

Make sure your code supports any number of features and is well-vectorized. You can use 'size(X, 2)' to find out how many features are present in the dataset.

Implementation Note: In the multivariate case, the cost function can also be written in the following vectorized form:

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y})$$

Where

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

The vectorized version is efficient when you're working with numerical computing tools like Octave/MATLAB. If you are an expert with matrix

Set the learning rate to 0.1, num_iters to 400, 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 1500 square feet and 5 bedrooms. You will use value later to check your implementation of the normal equations. Don't forget to normalize your features when you make this prediction!

1.2.1 Optional (ungraded) exercise: Selecting learning rates

In this part of the exercise, you will get to try out different learning rates for the dataset and find a learning rate that converges quickly. You can change the learning rate by modifying ex2.m and changing the part of the code that sets the learning rate.

The next phase in ex2.m will call your gradientDescentMulti. m function and run gradient descent for about 50 iterations at the chosen learning rate. The function should also return the history of $J(\theta)$ values in a vector J. After the last iteration, the ex2.m script plots the J values against the number of the iterations.

If you picked a learning rate within a good range, your plot look similar following Figure. If your graph looks very different, especially if your value of $J(\theta)$ increases or even blows up, adjust your learning rate and try again. We recommend trying values of the learning rate α on a log-scale, at multiplicative steps of about 3 times the previous value (i.e., 0.3, 0.1, 0.03, 0.01 and so on). You may also want to adjust the number of iterations you are running if that will help you see the overall trend in the curve.

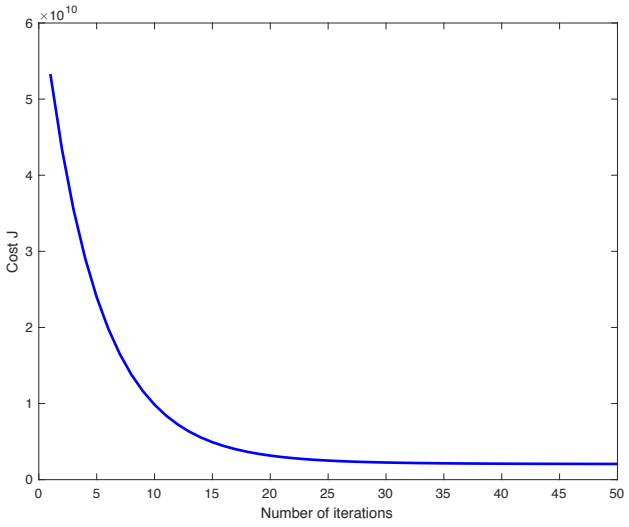

Figure 1: Convergence of gradient descent with an appropriate learning rate

Implementation Note: If your learning rate is too large, $J(\theta)$ can diverge and 'blow up', resulting in values which are too large for computer calculations. In these situations, Octave/MATLAB will tend to return NaNs. NaN stands for 'not a number' and is often caused by undefined operations that involve $-\infty$ and $+\infty$.

Notice the changes in the convergence curves as the learning rate changes. With a small learning rate, you should find that gradient descent takes a very long time to converge to the optimal value. Conversely, with a large learning rate, gradient descent might not converge or might even diverge!

1.3 Normal Equations

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

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

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 `ex1.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 1500-square-foot house with 5 bedrooms. You should find that gives the same

predicted price as the value you obtained using the model fit with gradient descent (in Section 1.2.1).

Submission and Grading

After completing various parts of the assignment, be sure to submit all the files needed to blackboard. The following is a breakdown of how each part of this exercise is scored.

Part	Related code file	Points
The mean value and the standard deviation for ex2data	featureNormalize.m	20 points
The value of theta after 400 iterations with α=0.1 Predict the price of a 1500 sq-ft, 5 br house	gradientDescentMulti.m ex2.m	40 points
The value of theta Predict the price of a 1500 sq-ft, 5 br house	normalEqn.m ex2.m	40 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.