LINEAR REGRESSION

Machine Learning implemented in MATLAB - Report 1

Author: Alberto Ibarrondo Date: 10/01/2017 License: MIT Free software

1 INTRODUCTION

The objective of this first study is to implement a Linear Regression model in MATLAB and apply it on data.

1.1 Files included in this study

- *MachLearnInMATLAB_1_LinearRegression.m*- MATLAB script for the whole implementation of the study. The script sets up the dataset for the problems and makes calls to the rest of the functions. The first part of the study implements linear regression with one variable, while the second part covers linear regression with multiple variables.
- *data1.txt* - Dataset for linear regression with one variable
- *data2.txt* - Dataset for linear regression with multiple variables
- *computeCost.m* - Function to compute the cost of linear regression
- *gradientDescent.m* - Function to run gradient descent
- *featureNormalize.m* - Function to normalize features
- *normalEqn.m* - Function to compute the normal equations

2 LINEAR REGRESSION WITH ONE VARIABLE

In this section of the study, I implement linear regression with one variable to predict profit for a food truck. Assuming to be the CEO of a restaurant franchise that is 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. Which city should expand to next?

The file data1.txt contains the dataset for the 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.

2.1 Importing and Plotting the Data

Before starting on any task, it is often useful to understand the data by visualizing it. For a dataset with only two features a Scatter plot is suitable.

We import the data into the variables X and y:

```
data = load('data1.txt');                 % read comma separated data
X = data(:, 1); y = data(:, 2);
m = length(y);                            % number of training examples
```

Next, we create a scatter plot of the data:

```
plot(x, y, 'rx', 'MarkerSize', 10);           % Plot the data
ylabel('Profit in $10,000s');                 % Set the y-axis label
xlabel('Population of City in 10,000s');      % Set the x-axis label
```

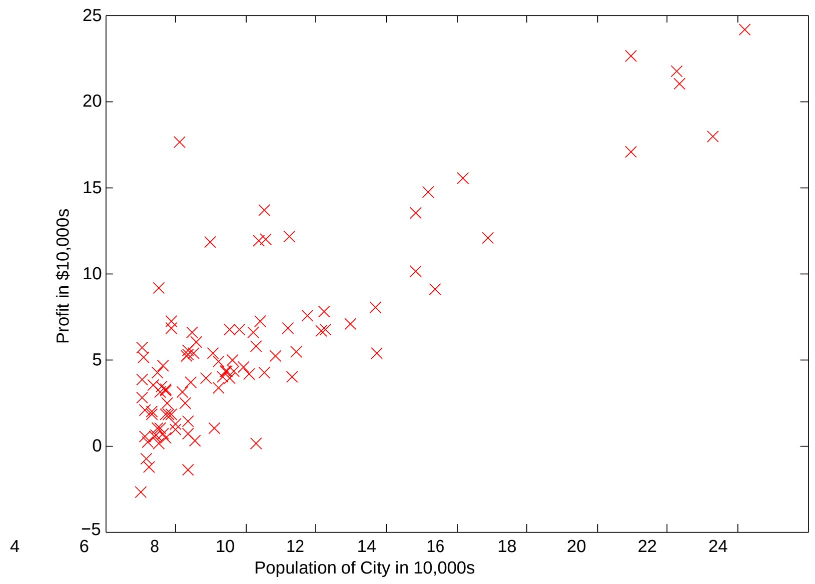


Figure 1: Scatter plot of training data

2.2 Gradient Descent

In this section we will fit the linear regression parameters θ to our dataset using gradient descent.

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 we 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:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

2.2.2 Implementation

In ex1.m, 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 alpha to 0.01.

```
X = [ones(m, 1), data(:,1)]; % Add a column of ones to x
theta = zeros(2, 1); % initialize fitting parameters

iterations = 1500; alpha = 0.01;
```

2.2.3 Computing the cost $J(\theta)$

As we perform gradient descent to learn minimize the cost function $J(\theta)$, it is helpful to monitor the convergence by computing the cost. I have implemented a function computeCost.m to calculate $J(\theta)$ in order to check the convergence of gradient descent.

2.2.4 Gradient descent

The gradient descent is coded in gradientDescent.m. We minimize the value of $J(\theta)$ by changing the values of the vector θ, not by changing X or y. Assuming a correct implementation of gradient descent and computeCost, the value of $J(\theta)$ should never increase, and should converge to a steady value by the end of the algorithm.

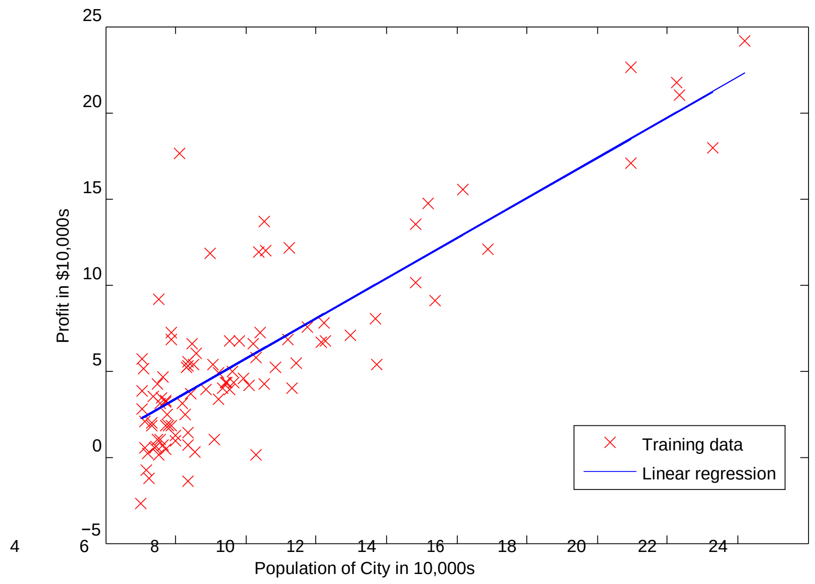


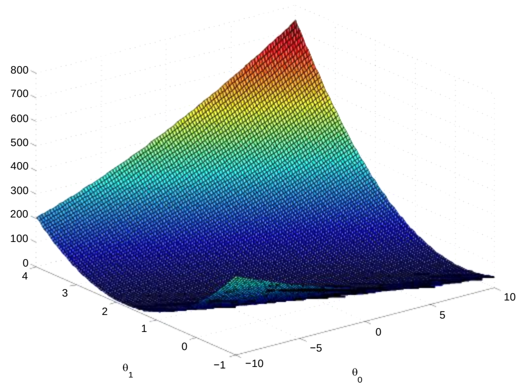
Figure 2: Training data with linear regression fit

2.3 Visualizing $J(\theta)$

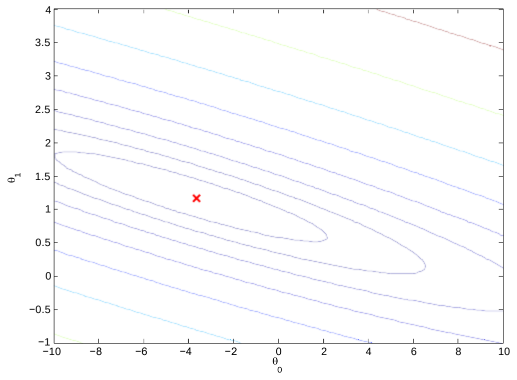
To understand the cost function $J(\theta)$ better, we plot the cost over a 2-dimensional grid of θ_0 and θ_1 values.

```
%-initialize J vals to a matrix of 0's
J vals = zeros(length(theta0 vals), length(theta1 vals));

% Fill out J vals
for i = 1:length(theta0 vals)
    for j = 1:length(theta1 vals)
       t = [theta0 vals(i); theta1 vals(j)];
       J vals(i,j) = computeCost(x, y, t);
    end
end
```



(a) Surface



(b) Contour, showing minimum

Figure 3: Cost function $J(\theta)$

The purpose of these graphs is to show 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.

3 LINEAR REGRESSION WITH MULTIPLE VARIABLES

In this section of the study I 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 data2.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.

3.1 Feature Normalization

The 3rd section of the script will start by loading and displaying some values from this dataset. By looking at the values we discover 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, which is what I implement in featureNormalize.m:

- 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).

3.2 Gradient Descent

Previously, I 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. The functions created for univariate already accept multivariate linear regression.

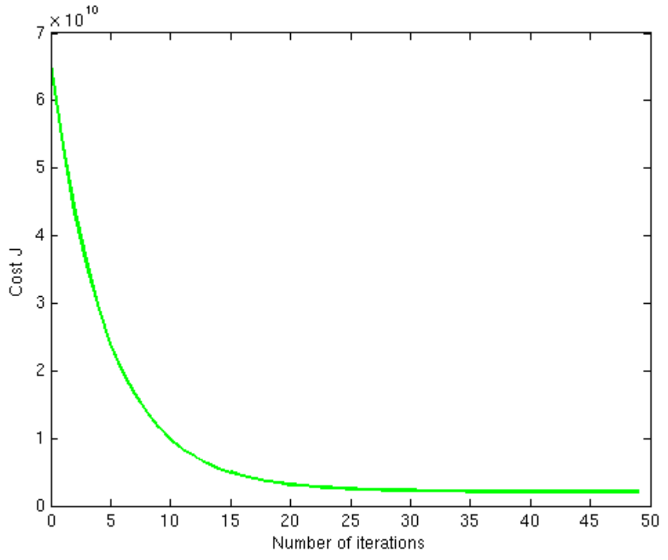


Figure 4: Convergence of gradient descent with an appropriate learning rate

3.3 Normal Equations

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

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

Using this formula does not require any feature scaling, and we get an exact solution in one calculation: there is no "loop until convergence" like in gradient descent. It's implement in normalEqn.m. To conclude, the different solutions are compared.