Exercise 2: Linear Regression

This course consists of videos and programming exercises to teach you about machine learning. The exercises are designed to give you hands-on, practical experience for getting these algorithms to work. To get the most out of this course, you should watch the videos and complete the exercises in the order in which they are listed.

This first exercise will give you practice with linear regression. These exercises have been extensively tested with Matlab, but they should also work in Octave, which has been called a "free version of Matlab." If you are using Octave, be sure to install the **Image** package as well (available for Windows as an option in the installer, and available for Linux from Octave-Forge).

Data

Download ex2Data.zip, and extract the files from the zip file.

The files contain some example measurements of heights for various boys between the ages of two and eights. The y-values are the heights measured in meters, and the x-values are the ages of the boys corresponding to the heights.

Each height and age tuple constitutes one training example $(x^{(i)}, y^{(i)})$ in our dataset. There are $m = 50$ training examples, and you will use them to develop a linear regression model.

Supervised learning problem

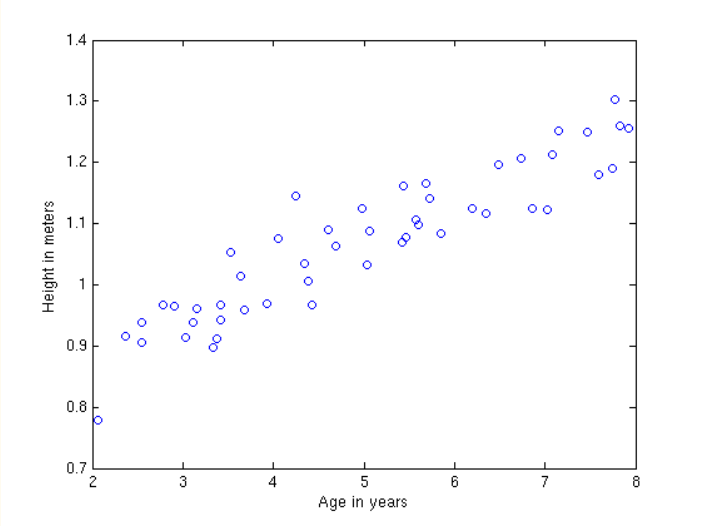
In this problem, you'll implement linear regression using gradient descent. In Matlab/Octave, you can load the training set using the commands

```
x = load('ex2x.dat');
y = load('ex2y.dat');
```

This will be our training set for a supervised learning problem with $n = 1$ features (in addition to the usual $x_0 = 1$, so $x \in R^2$). If you're using Matlab/Octave, run the following commands to plot your training set (and label the axes):

```
figure % open a new figure window
plot(x, y, 'o');
ylabel('Height in meters')
xlabel('Age in years')
```

You should see a series of data points similar to the figure below.



Before starting gradient descent, we need to add the $x_0 = 1$ intercept term to every example. To do this in Matlab/Octave, the command is

```
m = length(y); % store the number of training examples
x = [ones(m, 1), x]; % Add a column of ones to x
```

From this point on, you will need to remember that the age values from your training data are actually in the second column of `x`. This will be important when plotting your results later.

Linear regression

Now, we will implement linear regression for this problem. Recall that the linear regression model is

RESOURCES

Syllabus
FAQ
Credits/Acknowledgments

$$h_\theta(x) = \theta^T x = \sum_{i=0}^{n} \theta_i x_i,$$

and the batch gradient descent update rule is

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

1. Implement gradient descent using a learning rate of $\alpha = 0.07$. Since Matlab/Octave and Octave index vectors starting from 1 rather than 0, you'll probably use `theta(1)` and `theta(2)` in Matlab/Octave to represent θ_0 and θ_1. Initialize the parameters to $\theta = \vec{0}$ (i.e., $\theta_0 = \theta_1 = 0$), and run one iteration of gradient descent from this initial starting point. Record the value of of θ_0 and θ_1 that you get after this first iteration. (To verify that your implementation is correct, later we'll ask you to check your values of θ_0 and θ_1 against ours.)

2. Continue running gradient descent for more iterations until θ converges. (this will take a total of about 1500 iterations). After convergence, record the final values of θ_0 and θ_1 that you get.

When you have found θ, plot the straight line fit from your algorithm on the same graph as your training data. The plotting commands will look something like this:

```
hold on % Plot new data without clearing old plot
plot(x(:,2), x*theta, '-') % remember that x is now a matrix with 2 columns
                           % and the second column contains the time info
legend('Training data', 'Linear regression')
```

Note that for most machine learning problems, x is very high dimensional, so we don't be able to plot $h_\theta(x)$. But since in this example we have only one feature, being able to plot this gives a nice sanity-check on our result.

3. Finally, we'd like to make some predictions using the learned hypothesis. Use your model to predict the height for a two boys of age 3.5 and age 7.

Debugging If you are using Matlab/Octave and seeing many errors at runtime, try inspecting your matrix operations to check that you are multiplying and adding matrices in ways that their dimensions would allow. Remember that Matlab/Octave by default interprets an operation as a matrix operation. In cases where you don't intend to use the matrix definition of an operator but your expression is ambiguous to Matlab/Octave, you will have to use the 'dot' operator to specify your command. Additionally, you can try printing x, y, and theta to make sure their dimensions are correct.

Understanding $J(\theta)$

We'd like to understand better what gradient descent has done, and visualize the relationship between the parameters $\theta \in R^2$ and $J(\theta)$. In this problem, we'll plot $J(\theta)$ as a 3D surface plot. (When applying learning algorithms, we don't usually try to plot $J(\theta)$ since usually $\theta \in R^n$ is very high-dimensional so that we don't have any simple way to plot or visualize $J(\theta)$. But because the example here uses a very low dimensional $\theta \in R^2$, we'll plot $J(\theta)$ to gain more intuition about linear regression.) Recall that the formula for $J(\theta)$ is

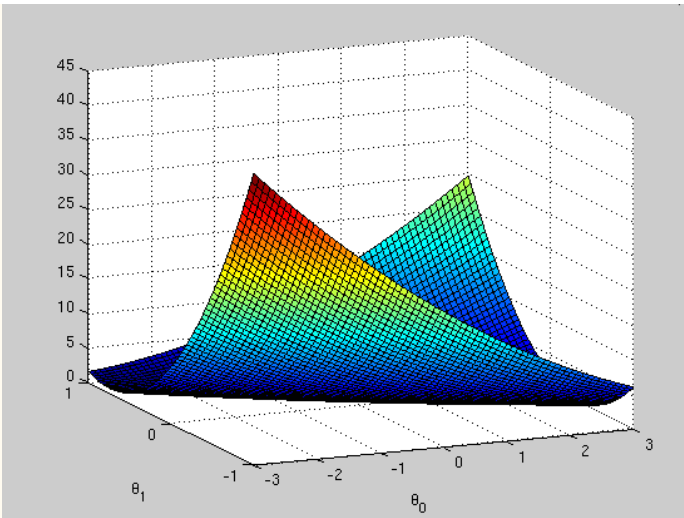
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

To get the best viewing results on your surface plot, use the range of theta values that we suggest in the code skeleton below.

```
J_vals = zeros(100, 100);   % initialize Jvals to 100x100 matrix of 0's
theta0_vals = linspace(-3, 3, 100);
theta1_vals = linspace(-1, 1, 100);
for i = 1:length(theta0_vals)
          for j = 1:length(theta1_vals)
          t = [theta0_vals(i); theta1_vals(j)];
          J_vals(i,j) = %% YOUR CODE HERE %%
    end
end

% Plot the surface plot
% Because of the way meshgrids work in the surf command, we need to
% transpose J_vals before calling surf, or else the axes will be flipped
J_vals = J_vals'
figure;
surf(theta0_vals, theta1_vals, J_vals)
xlabel('\theta_0'); ylabel('\theta_1')
```

You should get a figure similar to the following. If you are using Matlab/Octave, you can use the orbit tool to view this plot from different viewpoints.



What is the relationship between this 3D surface and the value of θ_0 and θ_1 that your implementation of gradient descent had found?

Hide Solution

Solutions

After you have completed the exercises above, please refer to the solutions below and check that your implementation and your answers are correct. In a case where your implementation does not result in the same parameters/phenomena as described below, debug your solution until you manage to replicate the same effect as our implementation.

A complete m-file implementation of the solutions can be found here. Run this m-file in Matlab/Octave to produce all the solutions and their corresponding graphs.

Linear Regression

1. After your first iteration of gradient descent, verify that you get

$$\begin{aligned}\theta_0 &= 0.0745\\ \theta_1 &= 0.3800\end{aligned}$$

If your answer does not exactly match this solution, you may have implemented something wrong. Did you get the correct $\theta_0 = 0.0745$, but the wrong answer for θ_1? (You might have gotten $\theta_1 = 0.4057$). If this happened, you probably updated the θ_j terms sequentially, that is, you first updated θ_0, plugged that value back into θ, and then updated θ_1.

Remember that you should not be basing your calculations on any intermediate values of θ that you would get this way.

2. After running gradient descent until convergence, verify that your parameters are approximately equal to the exact closed-form solution (which you will learn about in the next assignment):

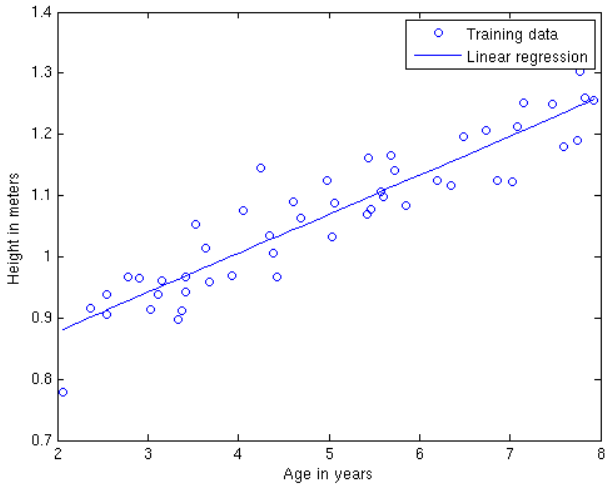
$$\begin{aligned}\theta_0 &= 0.7502\\ \theta_1 &= 0.0639\end{aligned}$$

If you run gradient descent in MATLAB for 1500 iterations at a learning rate of 0.07, you should see these exact numbers for theta. If used fewer iterations, your answer should not differ by more than 0.01, or you probably did not iterate enough. For example, running gradient descent in MATLAB for 500 iterations gives theta = [0.7318, 0.0672]. This is close to convergence, but theta can still get closer to the exact value if you run gradient descent some more.

If your answer differs drastically from the solutions above, there may be a bug in your implementation. Check that you used the correct learning rate of 0.07 and that you defined the gradient descent update correctly. Then, check that your x and y vectors are indeed what you expect them to be. Remember that x needs an extra column of ones.

3. The predicted height for age 3.5 is 0.9737 meters, and for age 7 is 1.1975 meters.

Plot A plot of the training data with the best fit from gradient descent should look like the following graph.



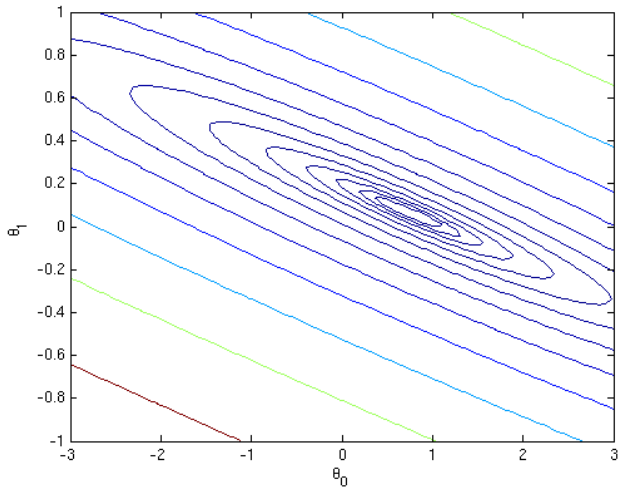
Understanding $J(\theta)$

In your surface plot, you should see that the cost function $J(\theta)$ approaches a minimum near the values of θ_0 and θ_1 that you found through gradient descent. In general, the cost function for a linear regression problem will be bowl-shaped with a global minimum and no local optima.

Depending on the viewing window of your surface plot, it may not be so apparent that the cost function is bowl-shaped. To see the approach to the global optimum better, it may be helpful to plot a contour plot. In Matlab/Octave, this can be done with

```
figure;
% Plot the cost function with 15 contours spaced logarithmically
% between 0.01 and 100
contour(theta0_vals, theta1_vals, J_vals, logspace(-2, 2, 15))
xlabel('\theta_0'); ylabel('\theta_1')
```

The result is a plot like the following.



Now the location of the minimum is more obvious.
