Note: If you are using MATLAB version R2015a or later, the fminunc() function has been changed in this version. The function works better, but does not give the expected result for Figure 5 in ex2.pdf, and it throws some warning messages (about a local minimum) when you run ex2_reg.m. This is normal, and you should still be able to submit your work to the grader.

Note: If your installation has trouble with the GradObj option, see this thread: <https://www.coursera.org/learn/machine-learning/discussions/s6tSSB9CEeWd3iIAC7VAtA>

Note: If you are using a linux-derived operating system, you may need to remove the attribute "MarkerFaceColor" from the plot() function call in plotData.m.

ex2: Tutorial for sigmoid()

You can get a one-line function for sigmoid(z) if you use only element-wise operators.

- The exp() function is element-wise.
- The addition operator is element-wise.
- Use the element-wise division operator ./

Combine these elements with a few parenthesis, and operate only on the parameter 'z'. The return value 'g' will then be the same size as 'z', regardless of what data 'z' contains.

============

keywords: tutorial sigmoid

Tom Mosher

MentorWeek 3 · 2 years ago · Edited

Test Cases for sigmoid() and predict()

Vicc Alexander

Week 3 · 3 years ago · Edited by moderator

In previous versions of this course we were lucky enough to be provided with unit tests to test our functions for mistakes / errors before submitting them (Maybe Tom might want to bring some of his older ones back?).
For those of you who don't know, unit testing is essentially the practice of testing functions in your code to make sure they work as expected. You can determine if your functions are working properly by cross-referencing your answers with the answers below. This helps identify any failures in your algorithms / logic.

I've provided a few of my own below. Hope this helps some of you out!

(Mentor comment: This thread is closed to comments. If you have a question, please post it in a new thread. Include a copy of the test case you used and your results when you post your question.)

```
>> sigmoid(-5)
```

```
ans =  0.0066929

>> sigmoid(0)
ans =  0.50000

>> sigmoid(5)
ans =  0.99331

>> sigmoid([4 5 6])
ans =

   0.98201   0.99331   0.99753

>> sigmoid([-1;0;1])
ans =

   0.26894
   0.50000
   0.73106

>> V = reshape(-1:.1:.9, 4, 5);
>> sigmoid(V)
ans =

   0.26894   0.35434   0.45017   0.54983   0.64566
   0.28905   0.37754   0.47502   0.57444   0.66819
   0.31003   0.40131   0.50000   0.59869   0.68997
   0.33181   0.42556   0.52498   0.62246   0.71095

>> X = [1 1 ; 1 2.5 ; 1 3 ; 1 4];
>> theta = [-3.5 ; 1.3];

% test case for predict()
>> predict(theta, X)
ans =

   0
   0
   1
   1
```

Note: If you do not get these result, check that you are including the sigmoid() function, and that the decision threshold is >= 0.5

Ex2 Tutorial: vectorizing the Cost function

Tom Mosher

MentorWeek 3 · 3 years ago · Edited

The regularized cost calculation can be vectorized easily. Here is the cost equation from ex2.pdf, page 9.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[\overset{\Sigma}{\boxed{-y^{(i)}\log(h_\theta(x^{(i)}))}} - \overset{\Sigma}{\boxed{(1-y^{(i)})\log(1-h_\theta(x^{(i)}))}}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2.$$

1. The hypothesis is a vector, formed from the sigmoid() of the products of X and . See the equation on ex2.pdf - Page 4. Be sure your sigmoid() function passes the submit grader before going any further.
2. First focus on the circled portions of the cost equation. Each of these is a vector of size (m x 1). In the steps below we'll distribute the summation operation, as shown in purple, so we end up with two scalars (for the 'red' and 'blue' calculations).
3. The red-circled term is the sum of -y multiplied by the natural log of h. Note that the natural log function is log(). Don't use log10(). Since we want the sum of the products, we can use a vector multiplication. The size of each argument is (m x 1), and we want the vector product to be a scalar, so use a transposition so that (1 x m) times (m x 1) gives a result of (1 x 1), a scalar.
4. The blue-circled term uses the same method, except that the two vectors are (1 - y) and the natural log of (1 - h).
5. Subtract the right-side term from the left-side term
6. Scale the result by 1/m. This is the unregularized cost.
7. Now we have only the regularization term remaining. We want the regularization to exclude the bias feature, so we can set theta(1) to zero. Since we already calculated h, and theta is a local variable, we can modify theta(1) without causing any problems.
8. Now we need to calculate the sum of the squares of theta. Since we've set theta(1) to zero, we can square the entire theta vector. If we vector-multiply theta by itself, we will calculate the sum automatically. So use the same method we used in Steps 3 and 4 to multiply theta by itself with a transposition.
9. Now scale the cost regularization term by (lambda / (2 * m)). Be sure you use enough sets of parenthesis to get the correct result. **Special Note for those whose cost value is too high:** 1/(2*m) and (1/2*m) give drastically different results.
10. Now add your unregularized and regularized cost terms together.

===============

keywords: ex2 tutorial costfunction costfunctionreg

ex2 test cases for costFunction() and costFunctionReg()

Tom Mosher

MentorWeek 3 · 2 years ago · Edited

(contributed by mentor Paul T Mielke)

Here is another test case that is a robust because the X matrix is not square and theta has no zero values.

The thread is closed to comments. If you have a question, please post in a new thread, and include a copy of this test case and your results.

(note: updated 1/24/2017 changing lambda from 3 to 4)

```
X = [ones(3,1) magic(3)];
y = [1 0 1]';
theta = [-2 -1 1 2]';

% un-regularized
[j g] = costFunction(theta, X, y)
% or...
[j g] = costFunctionReg(theta, X, y, 0)

% results
j = 4.6832

g =
  0.31722
  0.87232
  1.64812
  2.23787

% regularized
[j g] = costFunctionReg(theta, X, y, 4)
% note: also works for ex3 lrCostFunction(theta, X, y, 4)

% results
j =  8.6832
g =

   0.31722
  -0.46102
   2.98146
   4.90454
```

Ex2 Tutorial: vectorizing the gradient calculation

Tom Mosher

MentorWeek 3 · 3 years ago · Edited

The gradient calculation can be easily vectorized. See this two formulas from ex2.pdf pages 9 and 10.

Note: ignore the term in the 2nd equation if you are working on costFunction() - just do Step 1 and Step 2.

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad \text{for } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m}\theta_j \quad \text{for } j \geq 1$$

Note that if we set to zero (in Step 3 below), the second equation is exactly equal to the first equation. So we can ignore the "j = 0" condition entirely, and just use the second equation.

1. Recall that the hypothesis vector h is the sigmoid() of the product of X and (see ex2.pdf - Page 4). You probably already calculated h for the cost J calculation.
2. The left-side term is the vector product of X and (h - y), scaled by 1/m. You'll need to transpose and swap the product terms so the result is (m x n)' times (m x 1) giving you a (n x 1) result. This is the unregularized gradient. Note that the vector product also includes the required summation.
3. Then set theta(1) to 0 (if you haven't already).
4. Then calculate the regularized gradient term as theta scaled by (lambda / m).
5. The grad value is the sum of the Step 2 and Step 4 results. Since you forced theta(1) to be zero, the grad(1) term will only be the unregularized value.

============

keywords: ex2 tutorial costfunction tutorial costfunctionreg gradient

Tutorial for ex2: predict()

Tom Mosher

MentorWeek 3 · a year ago · Edited

This is logistic regression, so the hypothesis is the sigmoid of the product of X and theta.

Logistic prediction when there are only two classes uses a threshold of >= 0.5 to represent 1's and < 0.5 to represent a 0.

Here's an example of how to make this conversion in a vectorized manner. Try these commands in your workspace console, and study how they work:

```
v = rand(10,1)      % creates some random values between 0 and 1
v >= 0.5            % performs a logical comparison on each value
```

Inside your predict.m script, you will need to assign the results of this sort of logical comparison to the 'p' variable. You can use "p = " followed by a logical comparison inside a set of parenthesis.

This thread is closed to replies. Please post any questions about this tutorial on the Week 3 Discussion Forum.

Regarding how plotDecisionBoundary() works

Tom Mosher

MentorWeek 3 · 2 years ago · Edited

This post explains how the plotDecisionBoundary() function works.

For logistic regression, h = sigmoid(X * theta). This describes the relationship between X, theta, and h.

We know theta (from gradient descent). We are given X. So we can compute 'h'.

Now, by definition, the decision boundary is the locus of points where h = 0.5, or equivalently (X * theta) = 0, since the sigmoid(0) is 0.5.

Now we can write out the equation for the case where we have two features and a bias unit, and we write X as [] and theta as []

is the bias unit, it is hard-coded to 1.

Solve for

Now, to draw a line, you need two points. So pick two values for - anything near the minimum and maximum of the training set will serve. Compute the corresponding values for , and plot the pairs on the horizontal and vertical axes, then draw a line through them.

This line represents the decision boundary.

This is exactly what the plotDecisionBoundary() function does. is the variable "plot_y", and is the variable "plot_x".

=============

keywords: tutorial plotDecisionBoundary()

M

Making plotDecisionBoundary.m a little more useful

Tom Mosher

MentorWeek 3 · 3 months ago · Edited

The plotDecisionBoundary.m function is hard-coded for the range of X values that are used in the ex2 programming exercise.

You can modify the function so it works for more general purposes by:

```
% insert this code before line 17
dx = range(X(:,2)) * 0.02;

% comment out this line and replace
% allows the line to scale with the data set
```

```
%plot_x = [min(X(:,2))-2,  max(X(:,2))+2];
plot_x = [min(X(:,2))-dx,  max(X(:,2))+dx];

% comment-out this line
%axis([30, 100, 30, 100])

% replace these lines as follows

% comment-out the original code and replace as shown
%u = linspace(-1, 1.5, 50);
%v = linspace(-1, 1.5, 50);

% this lets the contour plot auto-range to the X data set
u = linspace(min(X(:,2)), max(X(:,2)), 50);
v = linspace(min(X(:,3)), max(X(:,2)), 50);
```

This isn't a graded function, so you can modify it without impacting your course grade.