##### <https://www.coursera.org/learn/machine-learning/discussions/all/threads/m0ZdvjSrEeWddiIAC9pDDA> o

##### Programming Exercise Tutorials (list)

Tom MosherMentor[General Discussion](https://www.coursera.org/learn/machine-learning/discussions/forums/vgyr_ygeEea7jBLLHPwd0w) · [2 years ago](https://www.coursera.org/learn/machine-learning/discussions/all/threads/m0ZdvjSrEeWddiIAC9pDDA) · Edited

This post contains links to all of the programming exercise tutorials.

After clicking on a link, you may need to scroll down to find the highlighted post.

--- Note: Additional test cases can be found [(here)](https://www.coursera.org/learn/machine-learning/discussions/0SxufTSrEeWPACIACw4G5w) ---

-------------

**ex1**

[computeCost()](https://www.coursera.org/learn/machine-learning/discussions/t35D1xn3EeWA7CIAC5WDNQ) tutorial - also applies to computeCostMulti().

[gradientDescent()](https://www.coursera.org/learn/machine-learning/discussions/-m2ng_KQEeSUBCIAC9QURQ) - also applies to gradientDescentMulti() - includes test cases.

[featureNormalize()](https://www.coursera.org/learn/machine-learning/module/vW94N/discussions/c7VBzJ9lEeWILRIOm1V0SQ) tutorial

Note: if you use OS X and the contour plot doesn't display correctly, see the Course Wiki for additional tips.

-------------

**ex2**

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: [<link>](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.

------------------------

[sigmoid()](https://www.coursera.org/learn/machine-learning/discussions/-v5KABxxEea_TAo4ODIo0w) tutorial

[costFunction()](https://www.coursera.org/learn/machine-learning/module/mgpv7/discussions/0DKoqvTgEeS16yIACyoj1Q) cost tutorial - also good for costFunctionReg()

[costFunction()](https://www.coursera.org/learn/machine-learning/discussions/GVdQ9vTdEeSUBCIAC9QURQ) gradient tutorial - also good for costFunctionReg()

[predict()](https://www.coursera.org/learn/machine-learning/discussions/weeks/3/threads/j2Vn07HqEeaYcRJ-aKpq1A) - tutorial for logistic regression prediction

Discussion of plotDecisionBoundary() [<link>](https://www.coursera.org/learn/machine-learning/module/mgpv7/discussions/HAEss7C7EeWoGg6ulZMPEw)

-------------

**ex3**

Note: a change to displayData.m for MacOS users: [(link)](https://www.coursera.org/learn/machine-learning/discussions/YlOmkiWsEeWeUyIAC44Ejw/replies/0A7DZi_BEeWOkCIAC4UG7w)

Note: if your images are upside-down, use flipud() to reverse the data. This is due to a change in gnuplot()'s defaults.

lrCostFunction() - This function is identical to your costFunctionReg() from ex2. Do not remove the line "grad = grad(:)" from the end of the lrCostFunction.m script template. This line guarantees that the grad value is returned as a column vector.

[oneVsAll()](https://www.coursera.org/learn/machine-learning/discussions/weeks/4/threads/sLIsSJU1EeW70BJZtLVfGQ) tutorial

[predictOneVsAll()](https://www.coursera.org/learn/machine-learning/module/mZYiz/discussions/Hfo82qxTEeWjcBKYJq1ZMQ) tutorial (updated)

[predict()](https://www.coursera.org/learn/machine-learning/module/mZYiz/discussions/miam5q2IEeWhLRIkesxXNw) tutorial (for the NN forward propagation - updated)

-------------

**ex4**

[nnCostFunction()](https://www.coursera.org/learn/machine-learning/programming/AiHgN/neural-network-learning/discussions/QFnrpQckEeWv5yIAC00Eog) - forward propagation and cost w/ regularization

[nnCostFunction()](https://www.coursera.org/learn/machine-learning/discussions/a8Kce_WxEeS16yIACyoj1Q) - tutorial for backpropagation

[Tutorial on using matrix multiplication to compute the cost value 'J'](https://www.coursera.org/learn/machine-learning/discussions/weeks/5/threads/AzIrrO7wEeaV3gonaJwAFA)

-------------

**ex5**

[linearRegCostFunction()](https://www.coursera.org/learn/machine-learning/discussions/UAv1DB62EeWd3iIAC7VAtA) tutorial

[polyFeatures()](https://www.coursera.org/learn/machine-learning/discussions/weeks/6/threads/YbO2RaVGEeaCbg44JUM1Vg) - tutorial

[learningCurve()](https://www.coursera.org/learn/machine-learning/module/xAUWb/discussions/Y_DZmpkgEeWNbBIwwhtGwQ) tutorial (really just a set of tips)

[validationCurve()](https://www.coursera.org/learn/machine-learning/discussions/AdGhzAX1EeWyEyIAC7PmUA/replies/7XjBAQ-MEeWUtiIAC9TNkg) tips­­­

-------------

**ex6**

Note: Update to ex6.m: At line 69/70, change "sigma = 0.5" to "sigma = %0.5f"

and change the list of output variables from "sim" to "sigma, sim".

(note: As of Jan 2017, this issue is already included in the zip file)

Note: Error in visualizeBoundary.m. Change the call to contour() like this:

contour(X1, X2, vals, [1 1], 'b');

(This change removes the attribute 'Color', and changes the contour interval. Note that [0.5 0.5] also works and is more logical, since "vals" has range [0..1])

This issue can cause either the "hggroup" error message, or the decision boundaries to not be displayed, or possibly cause Octave 3.8.x to crash when running ex6.m.

[All ex6 tutorials](https://www.coursera.org/learn/machine-learning/discussions/g2VB7po6EeWKNwpBrKr_Fw) (link)

-------------

**ex7**

[findClosestCentroids()](https://www.coursera.org/learn/machine-learning/module/kxH2P/discussions/ncYc-ddQEeWaURKFEvfOjQ)tutorial

[computeCentroids()](https://www.coursera.org/learn/machine-learning/discussions/weeks/8/threads/WzfDM7LjEeatew7zqUaXxg) tutorial

[Tutorials for ex7\_pca functions](https://www.coursera.org/learn/machine-learning/programming/ZZkM2/k-means-clustering-and-pca/discussions/wp_NfU55EeWxHxIGetKceQ) - pca(), projectData(), recoverData()

-------------

**ex8**

selectThreshold() - use the tips in the function script template, and the bulleted list on page 6 of ex8.pdf, to compute each of the tp, fp, and fn values.

Note: error in ex8\_cofi.m [(click this link)](https://www.coursera.org/learn/machine-learning/discussions/YD0v9TL_EeWj5iIACwIAYw)

Tip for estimateGaussian(): Compute the mean using "mean()". You can compute sigma2 using the equation in ex8.pdf, or you can use "var()" if you set the OPT parameter so it normalizes over the entire sample size.

[cofiCostFunc()](https://www.coursera.org/learn/machine-learning/module/HjnB4/discussions/92NKXCLBEeWM2iIAC0KUpw) tutorial

-------------

##### ex1 Tutorial for computeCost

Tom MosherMentor[Week 2](https://www.coursera.org/learn/machine-learning/discussions/weeks/2) · [2 years ago](https://www.coursera.org/learn/machine-learning/discussions/all/threads/t35D1xn3EeWA7CIAC5WDNQ) · Edited

This is a step-by-step tutorial for how to complete the computeCost() function portion of ex1. You will still have to do some thinking, because I'll describe the implementation, but you have to turn it into Octave script commands.

All the programming exercises in this course follow the same procedure; you are provided a starter code template for a function that you need to complete. You never have to start a new script file from scratch.

This is a vectorized implementation. You're only going to write a few simple lines of code.

With a text editor (NOT a word processor), open up the computeCost.m file. Scroll down until you find the "====== YOUR CODE HERE =====" section. Below this section is where you're going to add your lines of code. Just skip over the lines that start with the '%' sign - those are instructive comments.

We'll write these three lines of code by inspecting the equation on Page 5 of ex1.pdf

The first line of code will compute a vector 'h' containing all of the hypothesis values - one for each training example (i.e. for each row of X).

The hypothesis (also called the prediction) is simply the product of X and theta. So your first line of code is...

h = {multiply X and theta, in the proper order that the inner

    dimensions match}

Since X is size (m x n) and theta is size (n x 1), you arrange the order of operators so the result is size (m x 1).

The second line of code will compute the difference between the hypothesis and y - that's the error for each training example. Difference means subtract.

error = {the difference between h and y}

The third line of code will compute the square of each of those error terms (using element-wise exponentiation),

An example of using element-wise exponentiation - try this in your workspace command line so you see how it works

v = [-2 3]

v\_sqr = v.^2

So, now you should compute the squares of the error terms:

error\_sqr = {use what you have learned}

Next, here's an example of how the sum function works (try this from your command line)

q = sum([1 2 3])

Now, we'll finish the last two steps all in one line of code. You need to compute the sum of the error\_sqr vector, and scale the result (multiply) by 1/(2\*m). That completed sum is the cost value J.

J = {multiply 1/(2\*m) times the sum of the error\_sqr vector}

That's it. If you run the ex1.m script (by entering the command "ex1" in the console), you should have the correct value for J. Then you should test further by running the additional Test Cases (available via the Resources menu).

**Important Note:** You cannot test your computeCost() function by simply entering "computeCost" or "computeCost()" in the console. The function requires that you pass it three data parameters (X, y, and theta). The "ex1" script does this for you.

Then you can run the "submit" script, and hopefully it will pass.

Note: Be sure that every line of code ends with a semicolon. That will suppress the output of any values to the workspace. Leaving out the semicolons will surely make the grader unhappy.

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

This thread is closed to new comments. If you have a question, please start a new thread in the Week 2 discussion forum area.

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

keywords: tutorial computeCost

##### ex1 Tutorial and test case for gradientDescent()

Tom MosherMentor[Week 2](https://www.coursera.org/learn/machine-learning/discussions/weeks/2) · [2 years ago](https://www.coursera.org/learn/machine-learning/discussions/all/threads/-m2ng_KQEeSUBCIAC9QURQ) · Edited

Here is a tutorial on implementing gradientDescent() and gradientDescentMulti().

I use the vectorized method, hopefully you're comfortable with vector math. Using this method means you don't have to fuss with array indices, and your solution will automatically work for any number of features or training examples.

What follows is a vectorized implementation of the gradient descent equation on the bottom of Page 5 in ex1.pdf.

Reminder that 'm' is the number of training examples (the rows of X), and 'n' is the number of features (the columns of X). 'n' is also the size of the theta vector (n x 1).

Perform all of these steps within the provided for-loop from 1 to the number of iterations. Note that the code template provides you this for-loop - you only have to complete the body of the for-loop. The steps below go immediately below where the script template says "======= YOUR CODE HERE ======".

1 - The hypothesis is a vector, formed by multiplying the X matrix and the theta vector. X has size (m x n), and theta is (n x 1), so the product is (m x 1). That's good, because it's the same size as 'y'. Call this hypothesis vector 'h'.

2 - The "errors vector" is the difference between the 'h' vector and the 'y' vector.

3 - The change in theta (the "gradient") is the sum of the product of X and the "errors vector", scaled by alpha and 1/m. Since X is (m x n), and the error vector is (m x 1), and the result you want is the same size as theta (which is (n x 1), you need to transpose X before you can multiply it by the error vector.

The vector multiplication automatically includes calculating the sum of the products.

When you're scaling by alpha and 1/m, be sure you use enough sets of parenthesis to get the factors correct.

4 - Subtract this "change in theta" from the original value of theta. A line of code like this will do it:

theta = theta - theta\_change;

That's it. Since you're never indexing by m or n, this solution works identically for both gradientDescent() and gradientDescentMulti().

There is a test case below (or use this link):

<https://www.coursera.org/learn/machine-learning/discussions/-m2ng_KQEeSUBCIAC9QURQ/replies/jCkbzfQsEeSkXCIAC4tJTg>

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

Note: Replies to this thread tend to get lost due to a glitch in the forum. Please use the link below to post new questions.

<https://www.coursera.org/learn/machine-learning/discussions/uCXYyH6wEeWU3RJpSD4VQQ>

The thread you are reading is closed to new comments.

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

Keywords: ex1 tutorial gradientdescent gradientdescentmulti gradient

ex1 tutorial for featureNormalize()

Tom MosherMentor[Week 2](https://www.coursera.org/learn/machine-learning/discussions/weeks/2) · [2 years ago](https://www.coursera.org/learn/machine-learning/discussions/weeks/2/threads/c7VBzJ9lEeWILRIOm1V0SQ) · Edited

This tutorial discusses how to implement the featureNormalize() function using vectorization.

There are a couple of methods to accomplish this. The method here is one I use that doesn't rely on automatic broadcasting or the bsxfun() or repmat() functions.

* You can use the mean() and sigma() functions to get the mean and std deviation for each column of X. These are returned as row vectors (1 x n)
* Now you want to apply those values to each element in every row of the X matrix. One way to do this is to duplicate these vectors for each row in X, so they're the same size.

One method to do this is to create a column vector of all-ones - size (m x 1) - and multiply it by the mu or sigma row vector (1 x n). Dimensionally, (m x 1) \* (1 x n) gives you a (m x n) matrix, and every row of the resulting matrix will be identical.

* Now that X, mu, and sigma are all the same size, you can use element-wise operators to compute X\_normalized.

Try these commands in your workspace:

X = [1 2 3; 4 5 6] % creates a test matrix

mu = mean(X) % returns a row vector

sigma = std(X) % returns a row vector

m = size(X, 1) % returns the number of rows in X

mu\_matrix = ones(m, 1) \* mu

sigma\_matrix = ones(m, 1) \* sigma

Now you can subtract the mu matrix from X, and divide element-wise by the sigma matrix, and arrive at X\_normalized.

You can do this even easier if you're using a Matlab or Octave version that supports automatic broadcasting - then you can skip the "multiply by a column of 1's" part.

You can also use the bsxfun() or repmat() functions. Be advised the bsxfun() has a non-obvious syntax that I can never remember,

[Tom Mosher](https://www.coursera.org/learn/machine-learning/profiles/f9ede1d5c36042f1920fbd9b1ba3d7bb)

[M](https://www.coursera.org/learn/machine-learning/profiles/f9ede1d5c36042f1920fbd9b1ba3d7bb)

##### ex2: Tutorial for sigmoid()

Tom MosherMentor[Week 3](https://www.coursera.org/learn/machine-learning/discussions/weeks/3) · [a year ago](https://www.coursera.org/learn/machine-learning/discussions/all/threads/-v5KABxxEea_TAo4ODIo0w) · Edited

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.

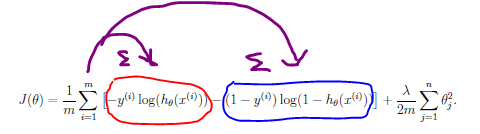
============

keywords: tutorial sigmoid

##### Ex2 Tutorial: vectorizing the Cost function

Tom MosherMentor[Week 3](https://www.coursera.org/learn/machine-learning/discussions/weeks/3) · [2 years ago](https://www.coursera.org/learn/machine-learning/discussions/weeks/3/threads/0DKoqvTgEeS16yIACyoj1Q) · Edited

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



1. The hypothesis is a vector, formed from the sigmoid() of the products of X and *θ*. 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

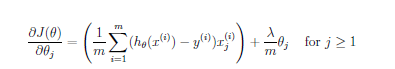
##### Ex2 Tutorial: vectorizing the gradient calculation

Tom MosherMentor[Week 3](https://www.coursera.org/learn/machine-learning/discussions/weeks/3) · [2 years ago](https://www.coursera.org/learn/machine-learning/discussions/all/threads/GVdQ9vTdEeSUBCIAC9QURQ) · Edited

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

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

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/HgdQp0tUEeWeURJeKTVPbw_d05d6b07453fd07c378958e0f118c09d_ex2_grad.PNG?expiry=1510099200000&hmac=1wUIqC3-09M_L0ju6wXnNI-8qhiEZd1elQrawTQ2Gqs



Note that if we set *θ*0 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 *θ* (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 MosherMentor[Week 3](https://www.coursera.org/learn/machine-learning/discussions/weeks/3) · [a year ago](https://www.coursera.org/learn/machine-learning/discussions/weeks/3/threads/j2Vn07HqEeaYcRJ-aKpq1A) · 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 MosherMentor[Week 3](https://www.coursera.org/learn/machine-learning/discussions/weeks/3) · [2 years ago](https://www.coursera.org/learn/machine-learning/discussions/weeks/3/threads/HAEss7C7EeWoGg6ulZMPEw)

This post derives 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 know h - 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 [*x*0*x*1*x*2] and theta as [*θ*0*θ*1*θ*2]

0=*x*0*θ*0+*x*1*θ*1+*x*2*θ*2

*x*0 is the bias unit, it is hard-coded to 1.

0=*θ*0+*x*1*θ*1+*x*2*θ*2

Solve for *x*2

*x*2=−(*θ*0+*x*1*θ*1)/*θ*2

Now, to draw a line, you need two points. So pick two values for *x*1 - anything near the minimum and maximum of the training set will serve. Compute the corresponding values for *x*2, and plot the (*x*1*x*2) 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. *x*2 is the variable "plot\_y", and *x*1 is the variable "plot\_x".

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

keywords: tutorial plotDecisionBoundary()