ex5 tutorial linearRegCostFunction

Tom MosherMentorWeek 6 · 2 years ago · Edited

Here is a brief tutorial for the linearRegCostFunction().

We last did a linear regression exercise back in ex1, so start with these two tutorials for computeCost() and gradientDescent(). Since they're vectorized, they work equally well for any multiple-variable linear regression.

computeCost tutorial

gradientDescent tutorial

You only need the first three steps of the gradientDescent() tutorial, plus scaling by 1/m (ignore the 'alpha' variable, it is not used in this exercise). That's gives us the gradient. Since we let fmincg() perform gradient descent for us, we just have to compute the cost and gradient. We don't use a for-loop over the number of iterations, or use any learning rate. The fmincg() function does that for us.

So now you've got unregularized cost J, and unregularized gradient 'grad'.

For the cost regularization:

- Set theta(1) to 0.
- Compute the sum of all of the theta values squared. One handy way to do this is sum(theta.^2). Since theta(1) has been forced to zero, it doesn't add to the regularization term.
- Now scale this value by lambda / (2*m), and add it to the unregularized cost.

For the gradient regularization:

- The regularized gradient term is theta scaled by (lambda / m). Again, since theta(1) has been set to zero, it does not contribute to the regularization term.
- Add this vector to the unregularized portion.

That's it. Here is a test case for this function:

https://www.coursera.org/learn/machine-learning/discussions/O25D0QykEeWZSyIAC5bWOg

Tutorial for polyFeatures()

Tom MosherMentorWeek 6 · 10 months ago

There are a couple of different methods that work for the polyFeatures() function.

One is to use the bsxfun() function, with the @power operator, like this:

1

X_poly = bsxfun(@power, vector1, vector2)

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

... where vector1 is a column vector of the feature values 'X', and vector2 is a

row vector of exponents from 1 to 'p'.
Other options involve using the element-wise exponent operator '.^', and converting both X and the vector of exponent values into equal-sized matrices by multiplying each by a vectors of all-ones.

ex5: tips for learningCurve()

Tom MosherMentorWeek 6 · 2 years ago · Edited

Tips for learningCurve()

- you may get some "divide by zero" warnings. These are expected and normal. fmincg() generates "divide by zero" warnings whenever the training set has only one or two examples. Do not worry about it.
- you should train a new theta vector for every possible size of training set (1:m). Use a for-loop from 1 to m.
- use the lambda that is in the learningCurve() parameter list every time you train to obtain theta.
- **do not** set lambda = 0 inside the learningCurve() function.
- use your cost function for computing the training error Jtrain and validation error Jcv.
- when you compute Jtrain and Jcv, use your cost function with a zero for the lambda parameter. We do not include any penalties for regularization in Jtrain or Jcv.
- use the same subset of X for both training theta and measuring Jtrain.
- use all of the validation set (Xval and yval) to measure Jcv. Do not use any subsets of Xval or yval.

Validation curve question

Mark AmottAssignment: Regularized Linear Regression and Bias/Variance · 2 years ago · Edited by moderator

for i = 1:length(lambda_vec) %already provided
lambda = lambda_vec(i); % already provided
{Mentor edit: pseudo-code-that-was-really-code-except-for-punctuation removed to protect the Honor Code}
end loop
My graph identifies that the optimum lambda is around 0.5, not 3 as the assignment suggests. I am training the model with the entire X and y and using the ith value of lambda on each occasion.