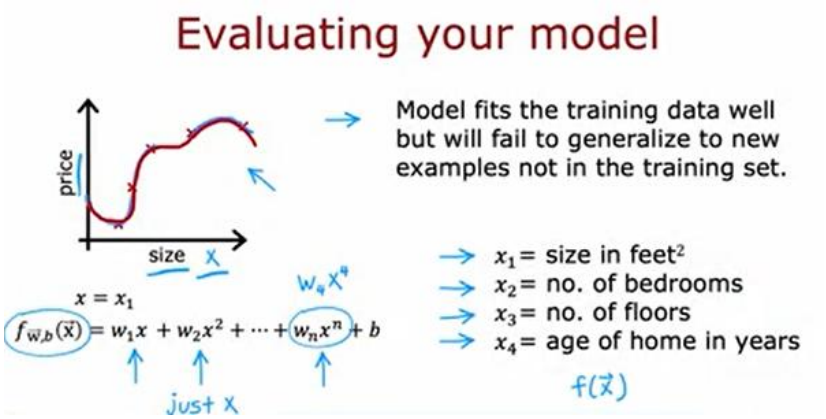
Model evaluation

Let's see, you've trained a machine learning model. How do you evaluate that model's performance? You find that having a systematic way to evaluate performance will also hope paint a clearer path for how to improve its performance.

So let's take a look at how to evaluate the model. Let's take the example of learning to predict housing prices as a function of the size. Let's say you've trained the model to predict housing prices as a function of the size x. And for the model that is a fourth order polynomial.

Let's see, you've trained a machine learning model. How do you evaluate that model's performance? You find that having a systematic way to evaluate performance will also hope paint a clearer path for how to improve its performance. So let's take a look at how to evaluate the model. Let's take the example of learning to predict housing prices as a function of the size. Let's say you've trained the model to predict housing prices as a function of the size x. And for the model that is a fourth order polynomial.

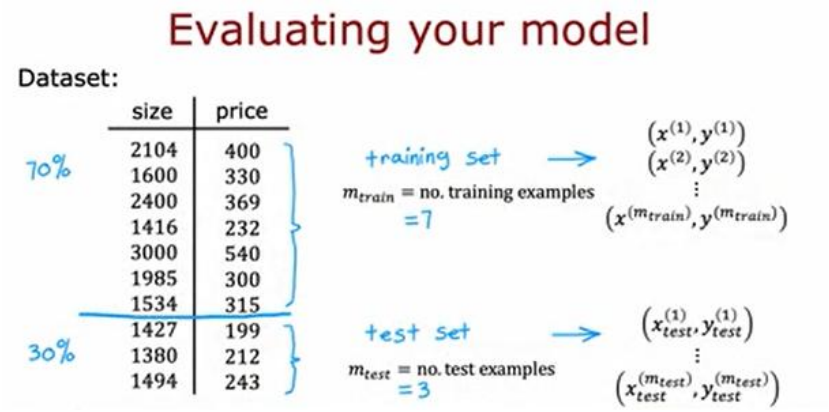


We need some more systematic way to evaluate how well your model is doing. Here's a technique that you can use. If you have a training set and this is a small training set with just 10 examples listed here, rather than taking all your data to train the parameters w and p of the model, you can instead split the training set into two subsets.

I'm going to draw a line here, and let's put 70% of the data into the first part and I'm going to call that the training set. And the second part of the data, let's say 30% of the data, I'm going to put into it a test set. And what we're going to do is train the models, parameters on the training set on this first 70% or so of the data, and then we'll test its performance on this test set. In notation, I'm going to use x1, y1?

Same as before, to denote the training examples through xm, ym, except that now to make explicit. So in this little example we would have seven training example. And to introduce one new piece of notation, I'm going to use m subscript train. M train is a number of training examples which in this small dataset is 7. So the subscript train just emphasizes if we're looking at the training set portion of the data.

And for the test set, I'm going to use the notation x1 subscript test comma y1, subscript test to denote the first test example, and this goes all the way to x mtest subscript tests, y mtest subscript tests and m tests is the number of test examples, which in this case is 3. And it's not uncommon to split your dataset according to maybe a 70, 30 split or 80, 20 split with most of your data going into the training set, and then a smaller fraction going into the test set.

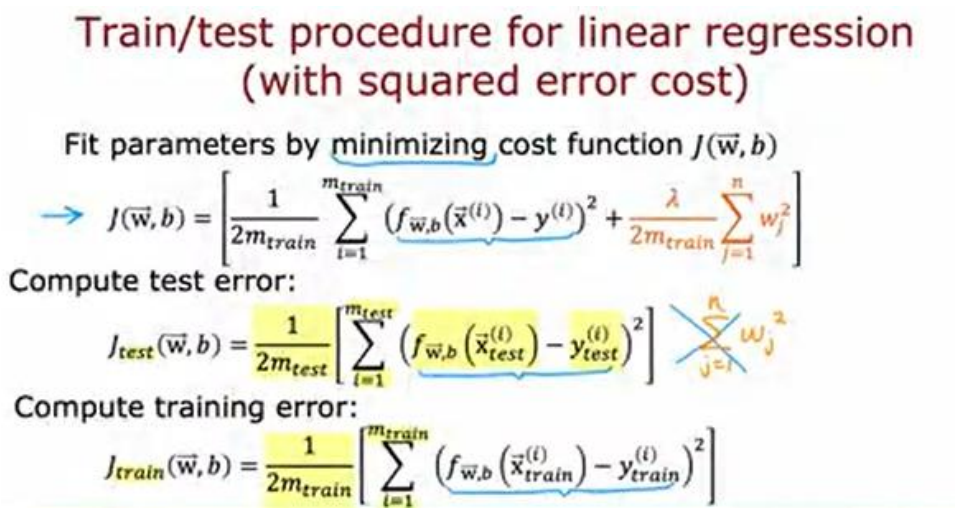


So, in order to train a model and evaluated it, this is what it would look like if you're using linear regression with a squared error cost. Start off by fitting the parameters by minimizing the cost function j of w,b. So this is the usual cost function minimize over w,b of this square error cost, plus regularization term lambda over 2m times some of the w,j squared.

And then to tell how well this model is doing, you would compute J test of w,b, which is equal to the average error on the test set, and that's just equal to 1/2 times m test. That's the number of test examples. And then of some overall the examples from r equals 1, to the number of test examples of the squared era on each of the test examples like so. So it's a prediction on the i'th test example input minus the actual price of the house on the test example squared.

And notice that the test error formula J test, it does not include that regularization term. And this will give you a sense of how well your learning algorithm is doing. One of the quantity that's often useful to computer as well as the training error, which is a measure of how well you're learning algorithm is doing on the training set.

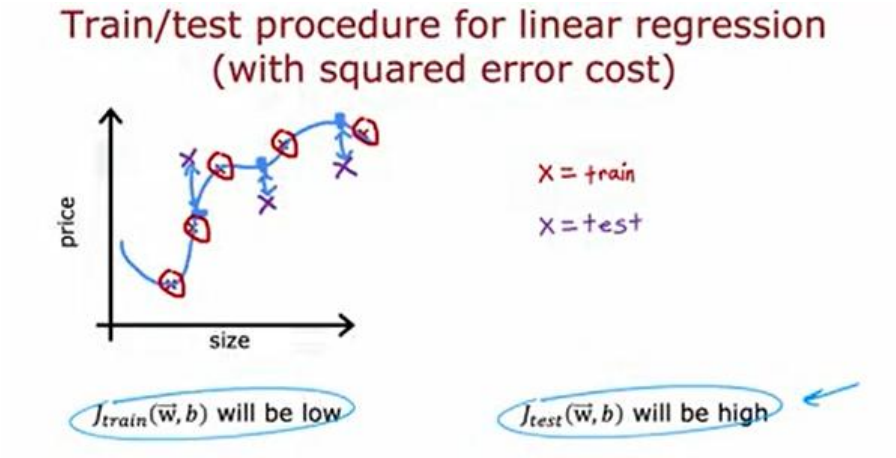
So let me define J train of w,b to be equal to the average over the training set. 1 over to 2m, or 1/2 m subscript train of some over your training set of this squared error term. And once again, this does not include the regularization term unlike the cost function that you are minimizing to fit the parameters. So, in the model like what we saw earlier in this video, J train of w,b will be low because the average era on your training examples will be zero or very close to zero. So J train will be very close to zero.



But if you have a few additional examples in your test set that the album had not trained on, then those test examples, might look like these. And there's a large gap between what the album is

predicting as the estimated housing price, and the actual value of those housing prices. And so, J tests will be high. So seeing that J test is high on this model, gives you a way to realize that even though it does great on the training set, is actually not so good at generalizing to new examples to new data points that were not in the training set.

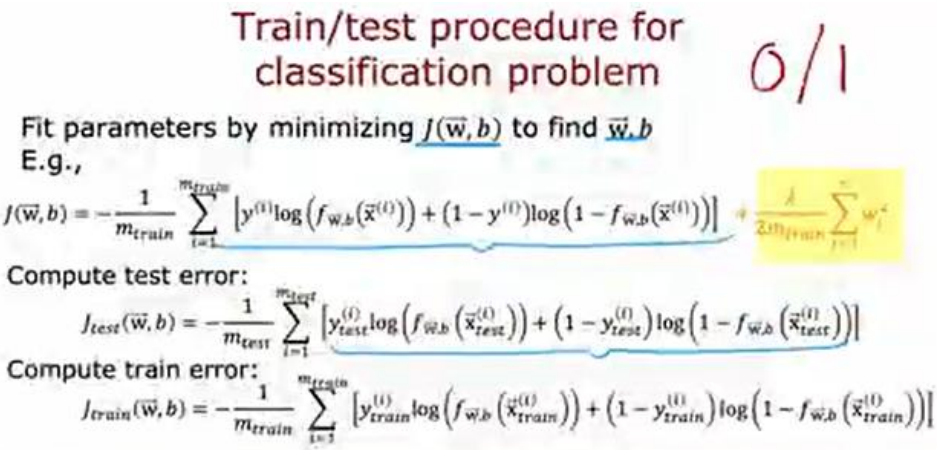
So, that was regression with squared error cost. Now, let's take a look at how you apply this procedure to a classification problem. For example, if you are classifying between handwritten digits that are either 0 or 1,, so same as before, you fit the parameters by minimizing the cost function to find the parameters w,b.



For example, if you were training logistic regression, then this would be the cost function J of w,b, where this is the usual logistic loss function, and then plus also the regularization term. And to compute the test error, J test is then the average over your test examples, that's that 30% of your data that wasn't in the training set of the logistic loss on your test set.

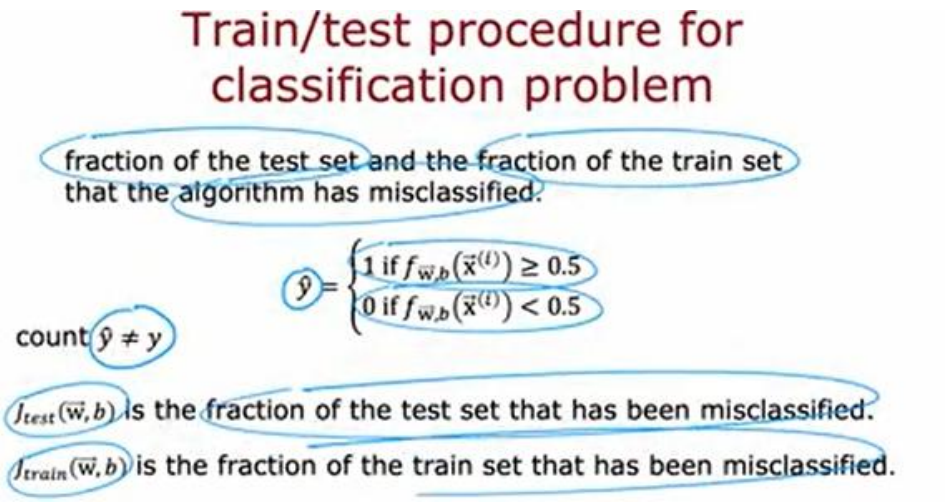
And the training error you can also compute using this formula, is the average logistic loss on your training data that the album was using to minimize the cost function J of w, b. Well, when I described here will work, okay, for figuring out if your learning algorithm is doing well, by seeing how I was doing in terms of test error.

When applying machine learning to classification problems, there's actually one other definition of J tests and J train that is maybe even more commonly used. Which is instead of using the logistic loss to compute the test error and the training error to instead measure what the fraction of the test set, and the fraction of the training set that the algorithm has misclassified. So specifically on the test set, you can have the algorithm make a prediction 1 or 0 on every test example.



So, recall y_hat we would predict us 1 if f of x is greater than equal 4.5, and zero if it's less than 0.5. And you can then count up in the test set the fraction of examples where y_hat is not equal to the actual ground truth label while in the test set. So concretely, if you are classifying handwritten digits 0, 1 binary classification loss, then J tests would be the fraction of that test set, where 0 was classified as 1 of 1, classified as 0.

And similarly, J train is a fraction of the training set that has been misclassified. Taking a dataset and splitting it into a training set and a separate test set gives you a way to systematically evaluate how well your learning algorithm is doing. By computing both J tests and J train, you can now measure how was doing on the test set and on the training set. This procedure is one step to what you'll be able to automatically choose what model to use for a given machine learning application.



For example, if you're trying to predict housing prices, should you fit a straight line to your data, or fit a second order polynomial, or third order fourth order polynomial? It turns out that with one further refinement to the idea you saw in this video, you'll be able to have an algorithm help you to automatically make that type of decision well.