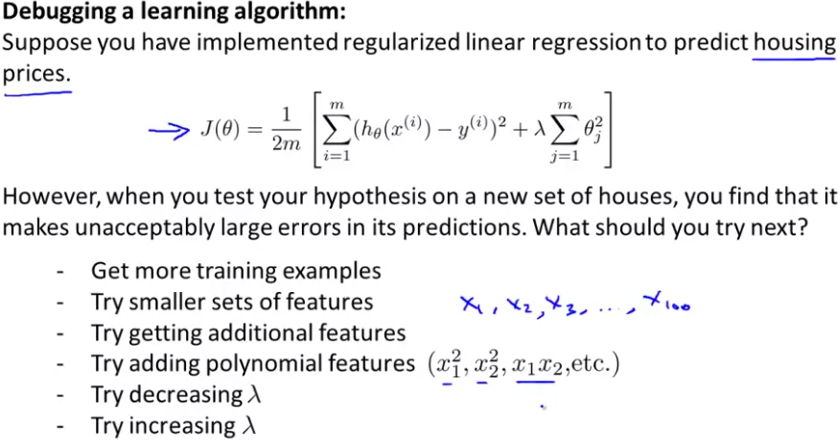
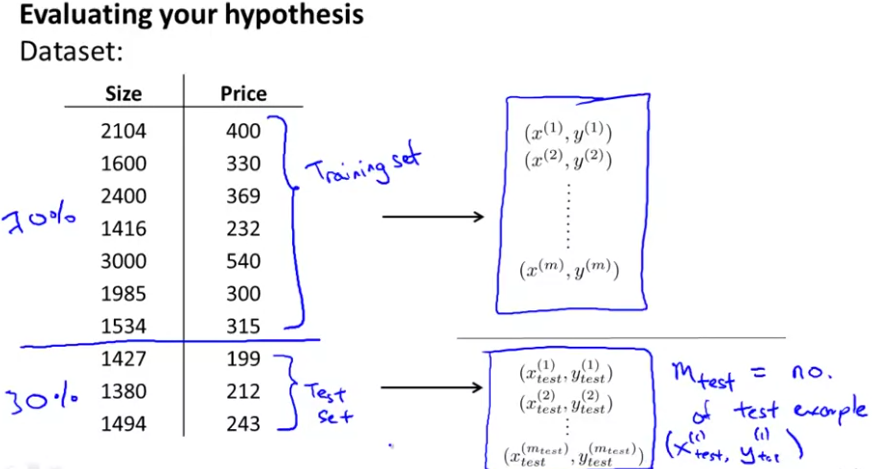
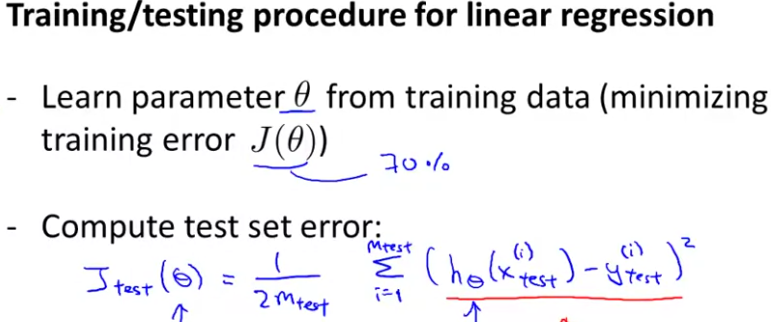
**Evaluating a Learning Algorithm**

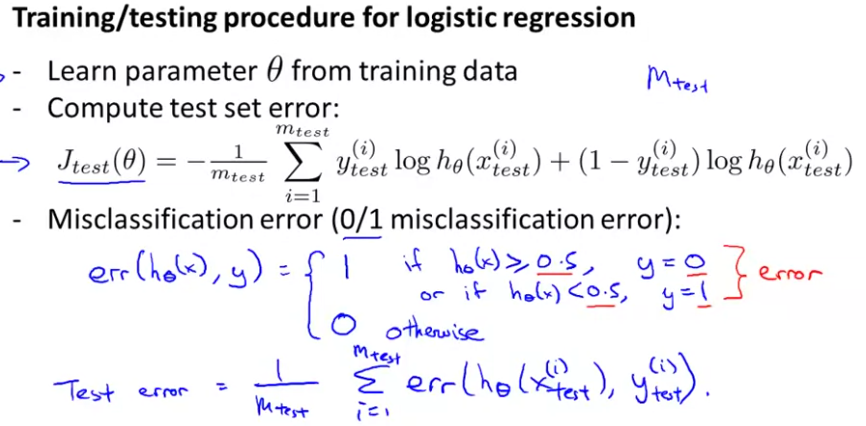


**Evaluating a hypothesis**

To evaluate a hypothesis, a usual way is to split the data we have and divide it into a training set and a test set. Then we train our algorithm with the training set and then we calculate the error that our hypothesis has for the test set.

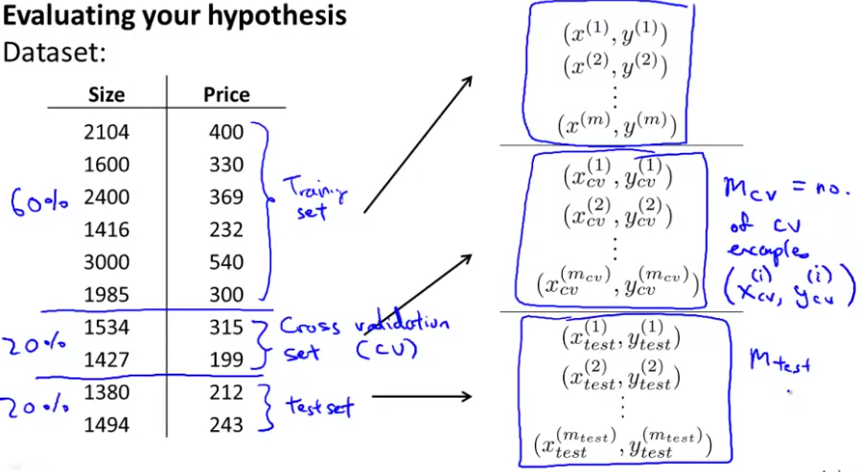




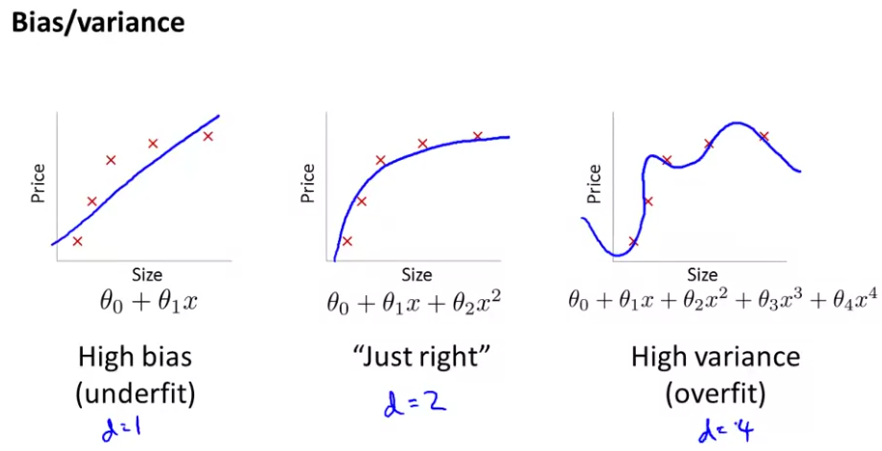


**Model selection and Train/Validation/Test Sets**

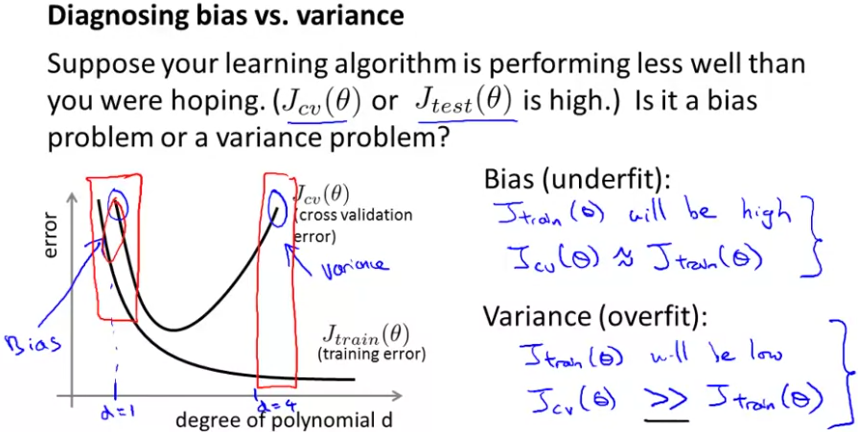
If we want to try out different models and see which one is better, we can calculate the theta parameters for each model using the training set, calculate the error for this set and then compare it to the error for the cross validation set for each model. We choose the model with a lower error for the cross validation set. Finally, we can estimate the generalization error form the test set.



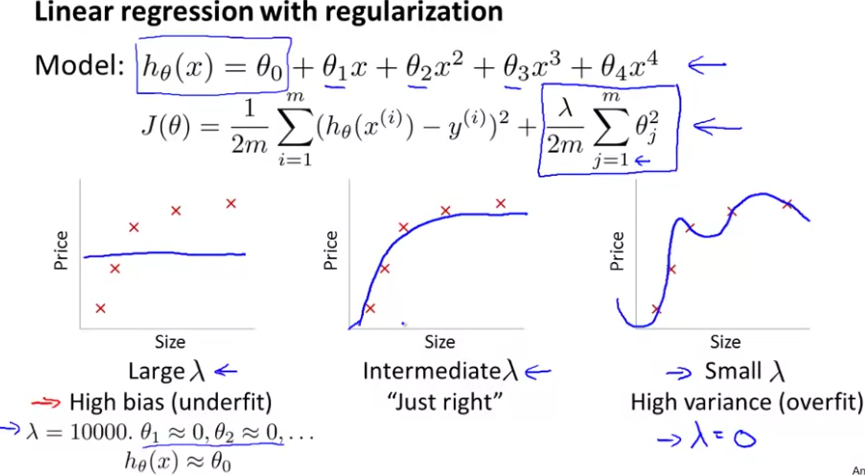
**Diagnosing Bias vs Variance (underfitting vs overfitting)**



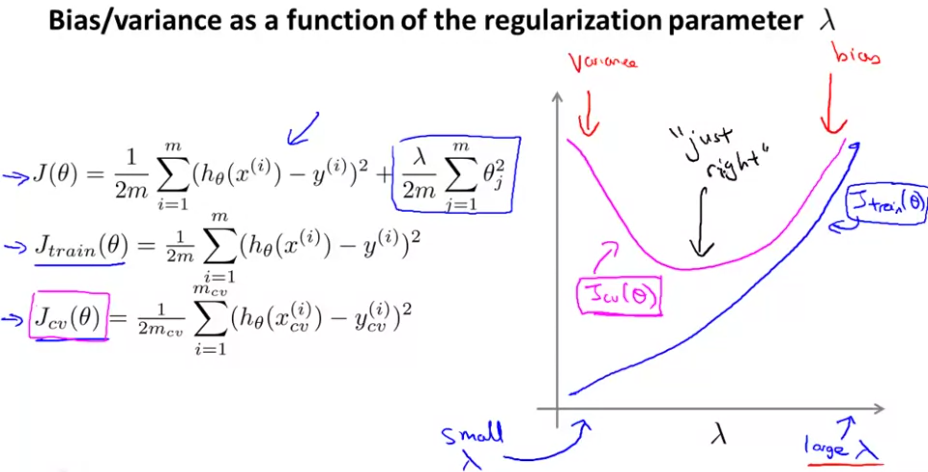
How to see if you are suffering from underfitting or form overfitting:



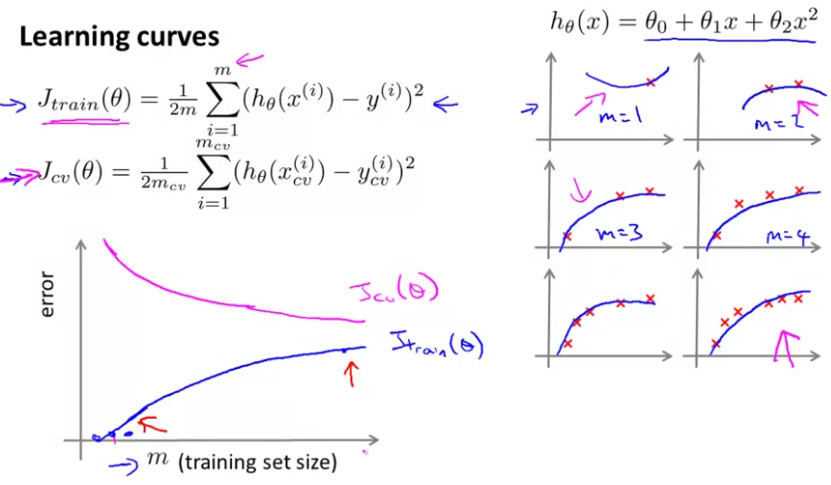
**Regularization and Bias/Variance**

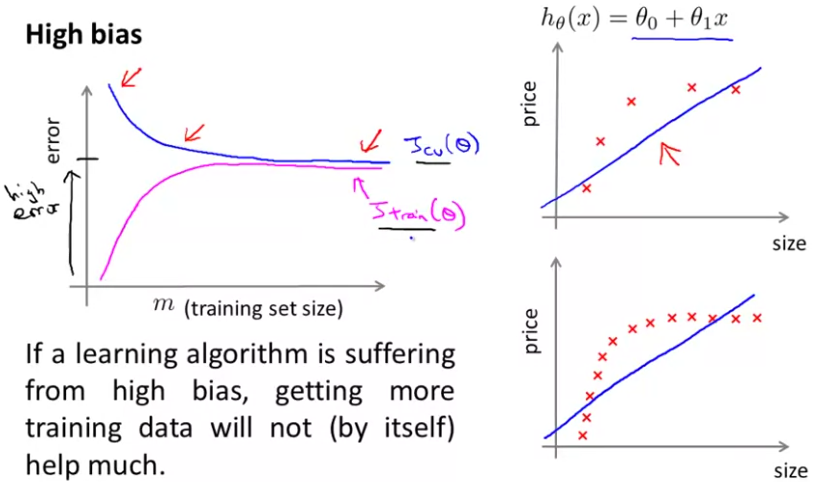


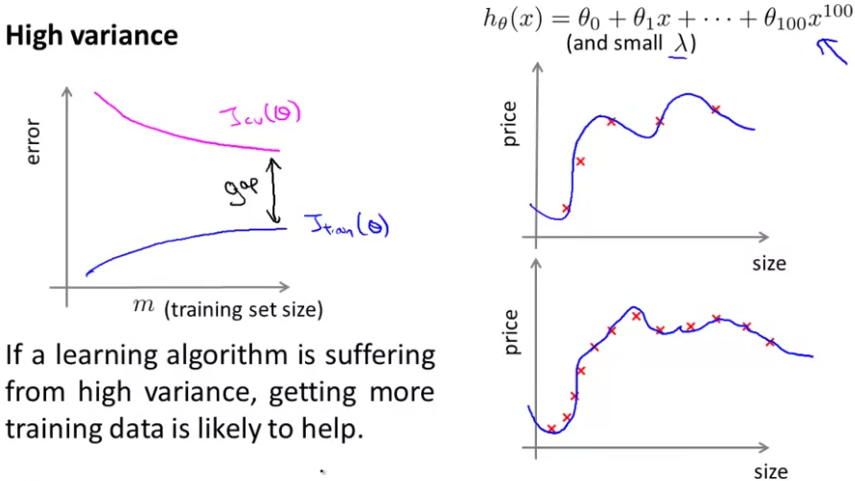
For choosing the best regularization parameter, we try many values for this parmeter and we choose the one with the lowest cross validation error.

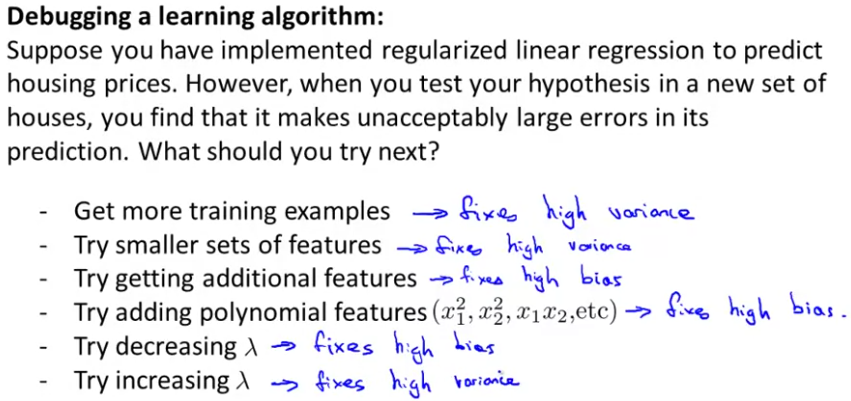


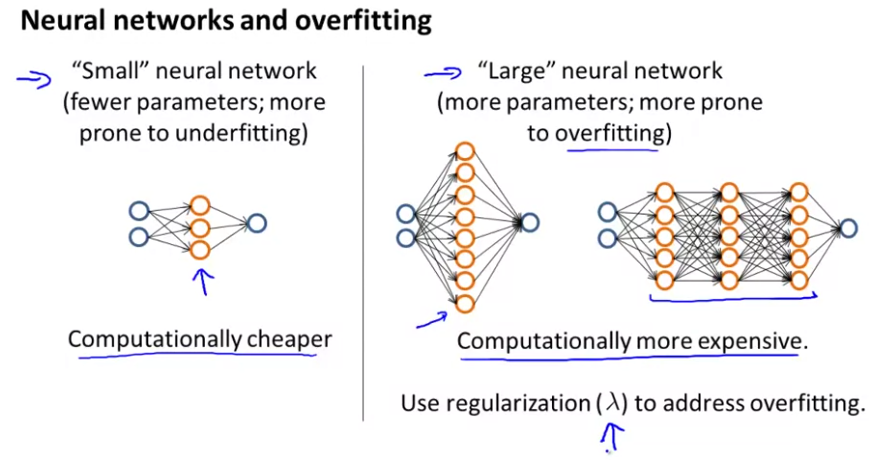
**Learning curves**



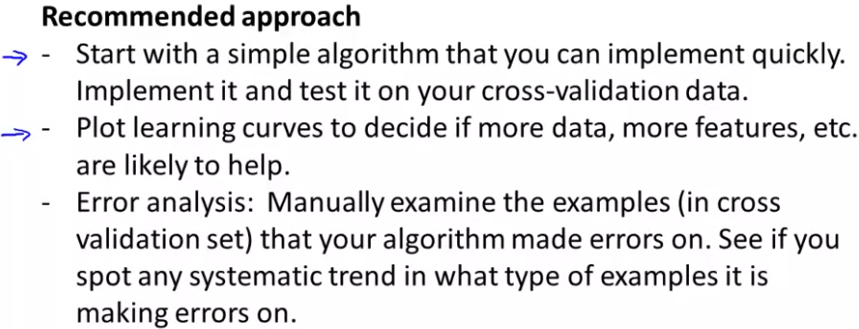






Usually, the bigger the neural network, the better.

**Machine learning system design**

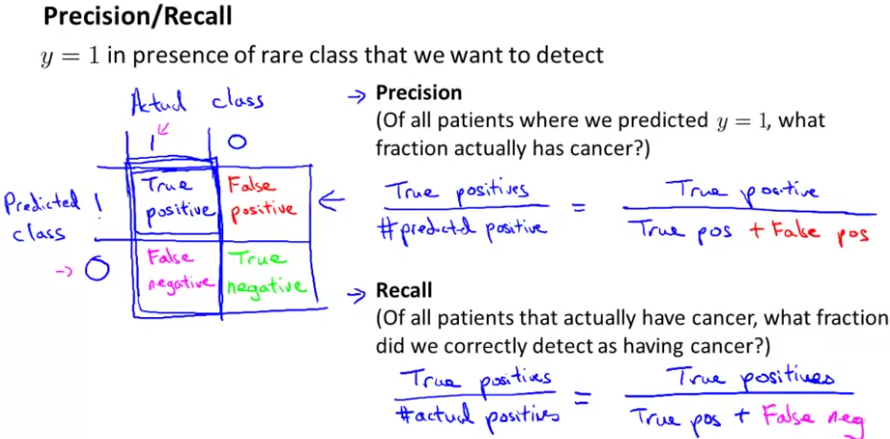


Once you have the initial implementation, this is then a powerful tool for deciding where to spend your time next. Because first you can look at the errors it makes and do this sort of error analysis to see what other mistakes it makes and use that to inspire further development.

And second, assuming your quick and dirty implementation incorporated a single real number evaluation metric, this can then be a vehicle for you to try out different ideas and quickly see if the different ideas you're trying out are improving the performance of your algorithm. And therefore, let you, maybe much more quickly make decisions about what things to fold in and what things to incorporate into your learning algorithm.

**Error Metrics for Skewed Classes**

Skewed classes are those in which the ratio of examples of one class is very close to one of two extremes. For these cases, we can use the precision/recall metrics:



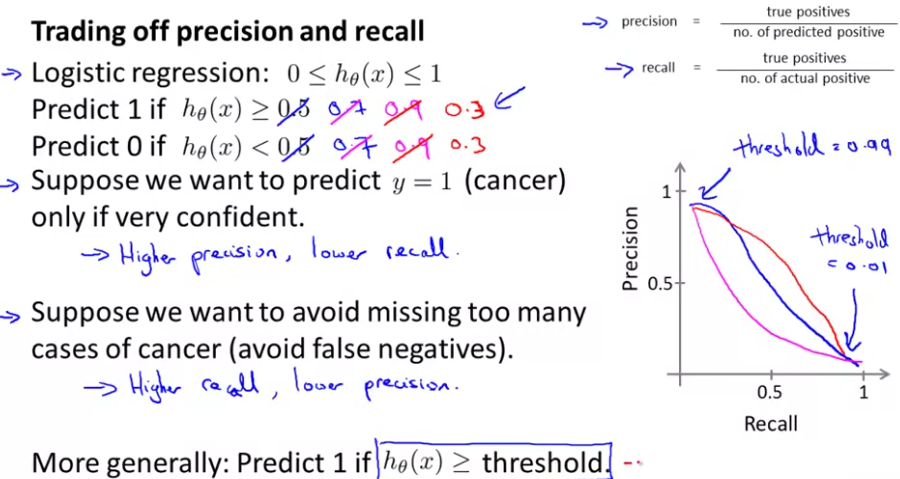
**Trading off between precision and recall**

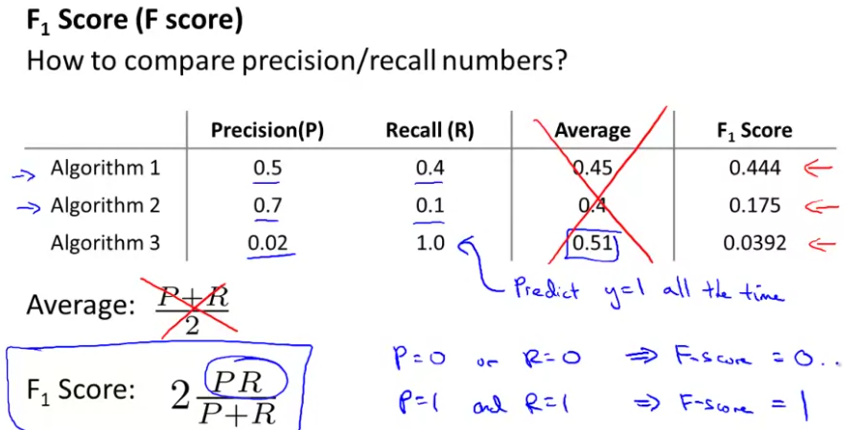
Un algoritmo con alta precisión nos asegura con una alta probabilidad que si detectamos que un paciente tiene cancer, efectivamente tendrá cancer.

En cambio, una exhaustividad alta nos indica que daremos con pocos falsos negativos. Es decir, que en pocas ocasiones ocurrirá que a un paciente que tiene cancer le diremos que no lo tiene.

En este caso, nos puede initeresar tener una mayor exhaustividad a tener una mayor precision.

Normalmente, existe un compromiso entre precision y exhaustividad:





**Data for Machine Learning**

When is it interesting to collect more data and how much data we must collect?

In order to know if we have made a good selection of features, an useful test is to ask the question: Can a human expert confidently predict y with the current parameters x?

