WEEK 6

Advice for Applying Machine Learning

Evaluating a Learning Algorithm

Deciding What to Try Next

By now you have seen a lot of different learning algorithms. And if you've been following along these videos you should consider yourself an expert on many state-of-the-art machine learning techniques. But even among people that

know a certain learning algorithm. There's often a huge difference between someone that really knows how to powerfully and effectively apply that algorithm, versus someone that's less familiar with some of the material that I'm about to teach and who doesn't really understand how to apply these algorithms and can end up wasting a lot of their time trying things out that don't really make sense. What I would like to do is make sure that if you are developing machine learning systems, that you know how to choose one of the most promising avenues to spend your time pursuing. And on this and the next few videos I'm going to give a number of practical suggestions, advice, guidelines on how to do that. And concretely what we'd focus on is the problem of, suppose you are developing a machine learning system or trying to improve the performance of a machine learning system, how do you go about deciding what are the proxy avenues to try next? To explain this, let's continue using our example of learning to predict housing prices. And let's say you've implement and regularize linear regression. Thus minimizing that cost function j. Now suppose that after you take your learn parameters, if you test your hypothesis on the new set of houses, suppose you find that this is making huge errors in this prediction of the housing prices. The question is what should you then try mixing in order to improve the learning ? There are many things that one can think of that could improve the performance of the learning algorithm. One thing they could try, is to get more training examples. And concretely, you can imagine, maybe, you know, setting up phone surveys, going door to door, to try to get more data on how much different houses sell for. And the sad thing is I've seen a lot of people spend a lot of time collecting more training examples, thinking oh, if we have twice as much or ten times as much training data, that is certainly going to help, right? But sometimes getting more training data doesn't actually help and in the next few videos we will see why, and we will see how you can avoid spending a lot of time collecting more training data in settings where it is just not going to help. Other things you might try are to well maybe try a smaller set of features. So if you have some set of features such as x1, x2, x3 and so on, maybe a large number of features. Maybe you want to spend time carefully selecting some small subset of them to prevent overfitting. Or maybe you need to get additional features. Maybe the current set of features aren't informative enough and you want to collect more data in the sense of getting more features. And once again this is the sort of project that can scale up the huge projects can you imagine getting phone surveys to find out more houses, or extra land surveys to find out more about the pieces of land and so on, so a huge project. And once again it would be nice to know in advance if this is going to help before we spend a lot of time doing something like this. We can also try adding polynomial features things like x2 square x2 square and product features x1, x2. We can still spend quite a lot of time thinking about that and we can also try other things like decreasing lambda, the regularization parameter or increasing lambda. Given a menu of options like these, some of which can easily scale up to six month or longer projects. Unfortunately, the most common method that people use to pick one of these is to go by gut feeling. In which what many people will do is sort of

randomly pick one of these options and maybe say, "Oh, lets go and get more training data." And easily spend six months collecting more training data or maybe someone else would rather be saying, "Well, let's go collect a lot more features on these houses in our data set." And I have a lot of times, sadly seen people spend, you know, literally 6 months doing one of these avenues that they have sort of at random only to discover six months later that that really wasn't a promising avenue to pursue. Fortunately, there is a pretty simple technique that can let you very quickly rule out half of the things on this list as being potentially promising things to pursue. And there is a very simple technique, that if you run, can easily rule out many of these options, and potentially save you a lot of time pursuing something that's just is not going to work. In the next two videos after this, I'm going to first talk about how to evaluate learning algorithms.

Debugging a learning algorithm:

Suppose you have implemented regularized linear regression to predict housing prices.

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

However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc.})$
- Try decreasing λ
- Try increasing λ

And in the next few videos after that, I'm going to talk about these techniques, which are called the machine learning diagnostics. And what a diagnostic is, is a test you can run, to get insight into what is or isn't working with an algorithm, and which will often give you insight as to what are promising things to try to improve a learning algorithm's performance. We'll talk about specific diagnostics later in this video sequence. But I should mention in advance that diagnostics can take time to implement and can sometimes, you know, take quite a lot of time to implement and understand but doing so can be a very good use of your time when you are developing learning algorithms because they can often save you from spending many months pursuing an avenue that you could have found out much earlier just was not going to be fruitful.

Machine learning diagnostic:

Diagnostic: A test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

Diagnostics can take time to implement, but doing so can be a very good use of your time.

Question

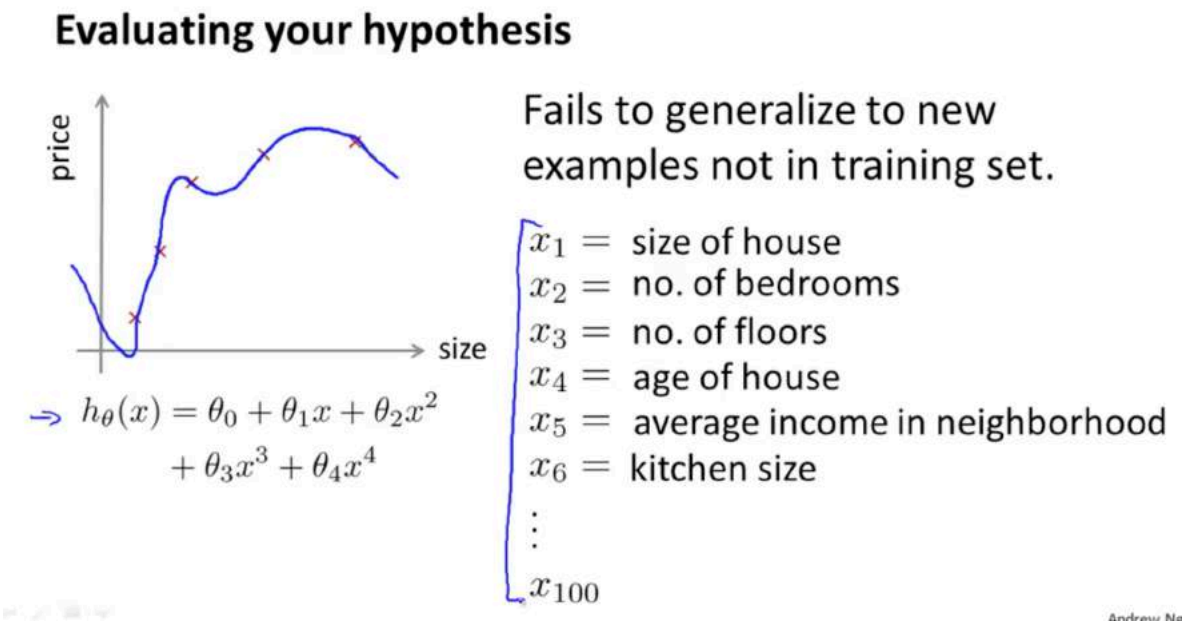
Which of the following statements about diagnostics are true? Check all that apply.

- [ ] It's hard to tell what will work to improve a learning algorithm, so the best approach is to go with gut feeling and just see what works.
- [x] Diagnostics can give guidance as to what might be more fruitful things to try to improve a learning algorithm.
- [x] Diagnostics can be time-consuming to implement and try, but they can still be a very good use of your time.
- [x] A diagnostic can sometimes rule out certain courses of action (changes to your learning algorithm) as being unlikely to improve its performance significantly.

So in the next few videos, I'm going to first talk about how evaluate your learning algorithms and after that I'm going to talk about some of these diagnostics which will hopefully let you much more effectively select more of the useful things to try mixing if your goal to improve the machine learning system.

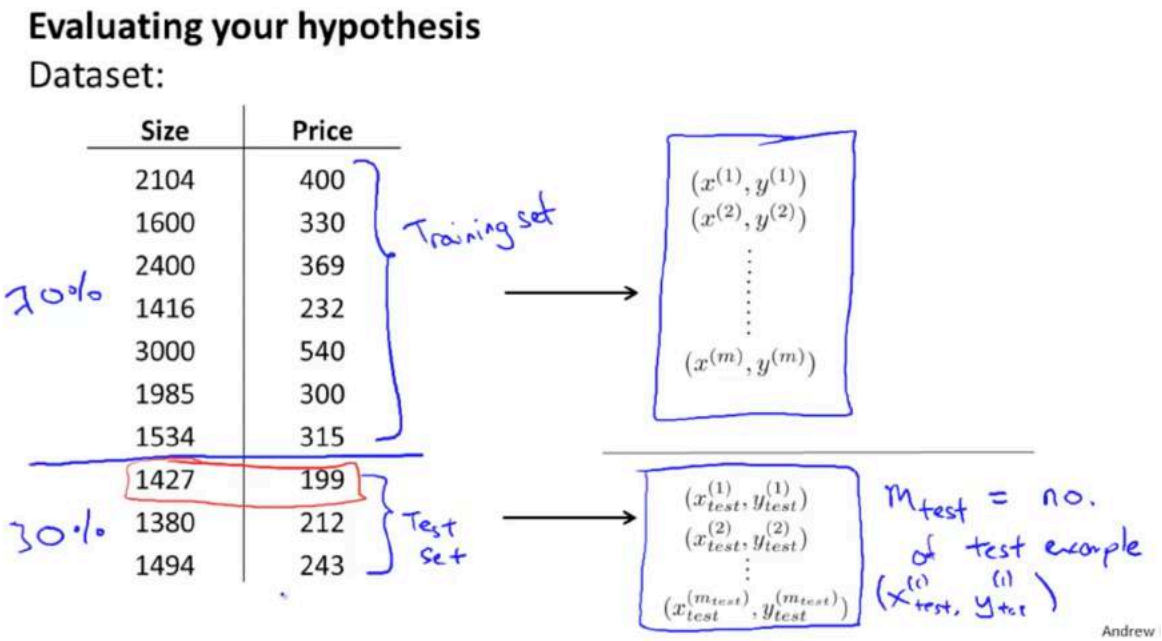
Evaluating a Hypothesis (Video)

In this video, I would like to talk about how to evaluate a hypothesis that has been learned by your algorithm. In later videos, we will build on this to talk about how to prevent in the problems of overfitting and underfitting as well. When we fit the parameters of our learning algorithm we think about choosing the parameters to minimize the training error. One might think that getting a really low value of training error might be a good thing, but we have already seen that just because a hypothesis has low training error, that doesn't mean it is necessarily a good hypothesis. And we've already seen the example of how a hypothesis can overfit. And therefore fail to generalize the new examples not in the training set. So how do you tell if the hypothesis might be overfitting. In this simple example we could plot the hypothesis h of x and just see what was going on. But in general for problems with more features than just one feature, for problems with a large number of features like these it becomes hard or may be impossible to plot what the hypothesis looks like and so we need some other way to evaluate our hypothesis.



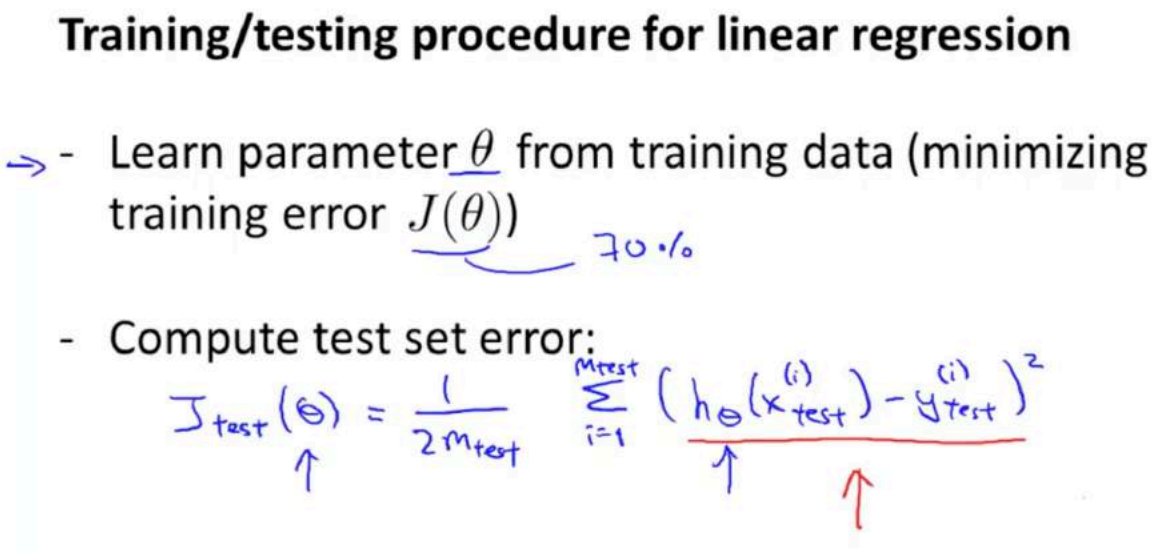
The standard way to evaluate a learned hypothesis is as follows. Suppose we have a data set like this. Here I have just shown 10 training examples, but of course usually we may have dozens or hundreds or maybe thousands of training examples. In order to make sure we can evaluate our hypothesis, what we are going to do is split the data we have into two portions. The first portion is going to be our usual training set and the second portion is going to be our test set, and a pretty typical split of this all the data we have into a training set and test set might be around say a 70%, 30% split. Worth more today to grade

the training set and relatively less to the test set. And so now, if we have some data set, we run a sine of say 70% of the data to be our training set where here "m" is as usual our number of training examples and the remainder of our data might then be assigned to become our test set. And here, I'm going to use the notation m subscript test to denote the number of test examples. And so in general, this subscript test is going to denote examples that come from a test set so that x1 subscript test, y1 subscript test is my first test example which I guess in this example might be this example over here. Finally, one last detail whereas here I've drawn this as though the first 70% goes to the training set and the last 30% to the test set. If there is any sort of ordinary to the data. That should be better to send a random 70% of your data to the training set and a random 30% of your data to the test set. So if your data were already randomly sorted, you could just take the first 70% and last 30% that if your data were not randomly ordered, it would be better to randomly shuffle or to randomly reorder the examples in your training set. Before you know sending the first 70% in the training set and the last 30% of the test set.

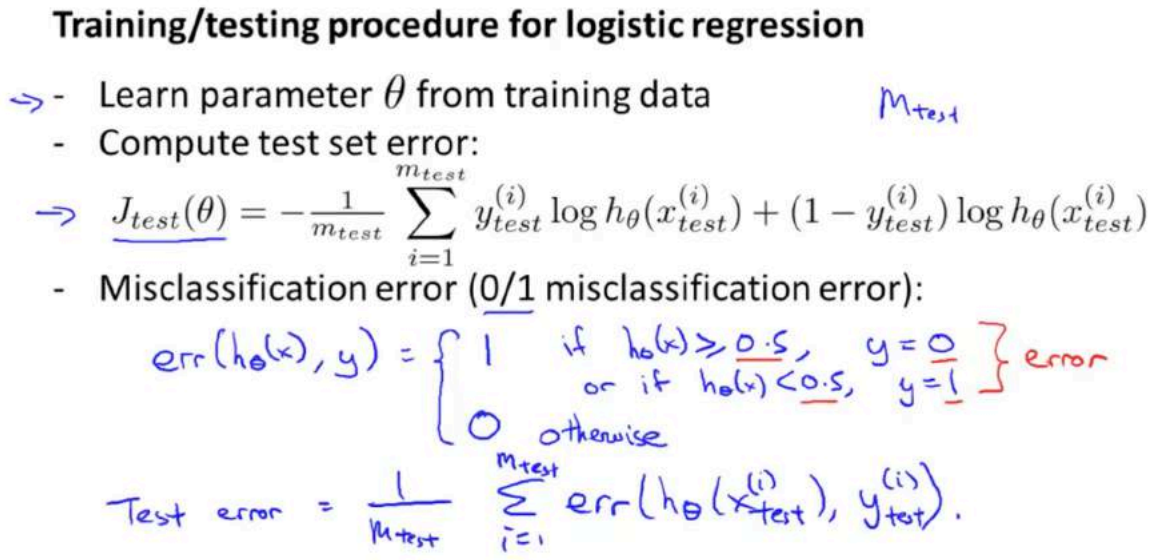


Here then is a fairly typical procedure for how you would train and test the learning algorithm and the learning regression. First, you learn the parameters theta from the training set so you minimize the usual training error objective j of theta, where j of theta here was defined using that 70% of all the data you have. There is only the training data. And then you would compute the test error. And I am going to denote the test error as j subscript test. And so what you do is take your parameter theta that you have learned from the training set, and plug it in here and compute your test set error. Which I am going to write as follows. So this is basically the average squared error as measured on your test set. It's pretty much what you'd expect. So if we run every test

example through your hypothesis with parameter theta and just measure the squared error that your hypothesis has on your m subscript test, test examples. And of course, this is the definition of the test set error if we are using linear regression and using the squared error metric.



How about if we were doing a classification problem and say using logistic regression instead. In that case, the procedure for training and testing say logistic regression is pretty similar first we will do the parameters from the training data, that first 70% of the data. And it will compute the test error as follows. It's the same objective function as we always use but we just logistic regression, except that now is define using our m subscript test, test examples. While this definition of the test set error j subscript test is perfectly reasonable. Sometimes there is an alternative test sets metric that might be easier to interpret, and that's the misclassification error. It's also called the zero one misclassification error, with zero one denoting that you either get an example right or you get an example wrong. Here's what I mean. Let me define the error of a prediction. That is h of x. And given the label y as equal to one if my hypothesis outputs the value greater than equal to five and Y is equal to zero or if my hypothesis outputs a value of less than 0.5 and y is equal to one, right, so both of these cases basic respond to if your hypothesis mislabeled the example assuming your threshold at an 0.5. So either thought it was more likely to be 1, but it was actually 0, or your hypothesis stored was more likely to be 0, but the label was actually 1. And otherwise, we define this error function to be zero. If your hypothesis basically classified the example y correctly. We could then define the test error, using the misclassification error metric to be one of the m tests of sum from i equals one to m subscript test of the error of h of x(i) test comma y(i). And so that's just my way of writing out that this is exactly the fraction of the examples in my test set that my hypothesis has mislabeled. And so that's the definition of the test set error using the misclassification error of the 0 1 misclassification metric.



So that's the standard technique for evaluating how good a learned hypothesis is. In the next video, we will adapt these ideas to helping us do things like choose what features like the degree polynomial to use with the learning algorithm or choose the regularization parameter for learning algorithm.

Evaluating a Hypothesis (Transcript)

Once we have done some trouble shooting for errors in our predictions by:

- Getting more training examples
- Trying smaller sets of features
- Trying additional features
- Trying polynomial features
- Increasing or decreasing λ

We can move on to evaluate our new hypothesis.

A hypothesis may have a low error for the training examples but still be inaccurate (because of overfitting). Thus, to evaluate a hypothesis, given a dataset of training examples, we can split up the data into two sets: a **training set** and a **test set.** Typically, the training set consists of 70 % of your data and the test set is the remaining 30 %.

The new procedure using these two sets is then:

1. Learn Θ and minimize $J_{train}(\Theta)$ using the training set
2. Compute the test set error $J_{test}(\Theta)$

The test set error

1. For linear regression: $J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\Theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$
2. For classification ~ Misclassification error (aka 0/1 misclassification error):

$$err(h_\Theta(x), y) = \begin{matrix} 1 & \text{if } h_\Theta(x) \geq 0.5 \text{ and } y = 0 \text{ or } h_\Theta(x) < 0.5 \text{ and } y = 1 \\ 0 & \text{otherwise} \end{matrix}$$

This gives us a binary 0 or 1 error result based on a misclassification. The average test error for the test set is:

$$\text{Test Error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\Theta(x_{test}^{(i)}), y_{test}^{(i)})$$

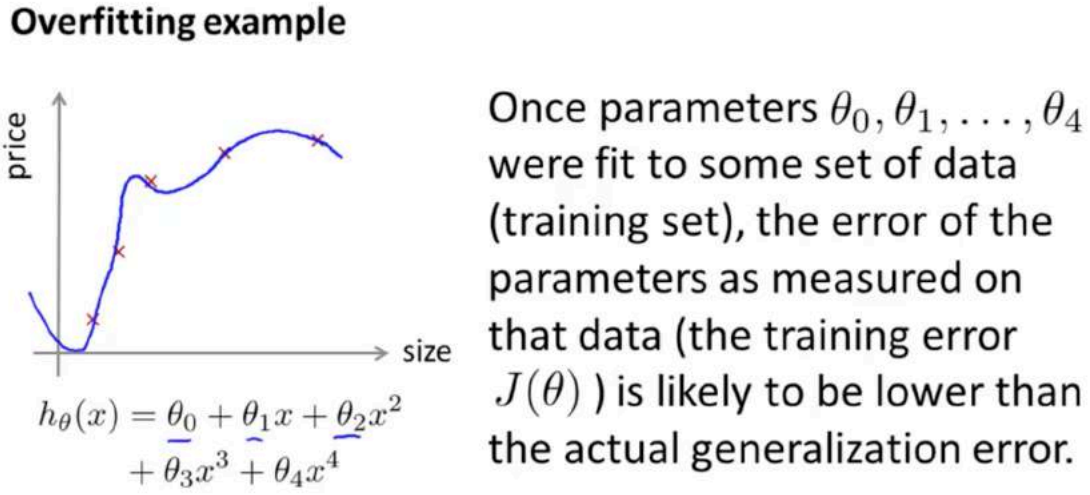
This gives us the proportion of the test data that was misclassified.

Model Selection and Train/ Validation/Test Sets (Video)

Suppose you're left to decide what degree of polynomial to fit to a data set. So that what features to include that gives you a learning algorithm. Or suppose you'd like to choose the regularization parameter longer for learning algorithm. How do you do that? This account model selection process. Browsers, and in

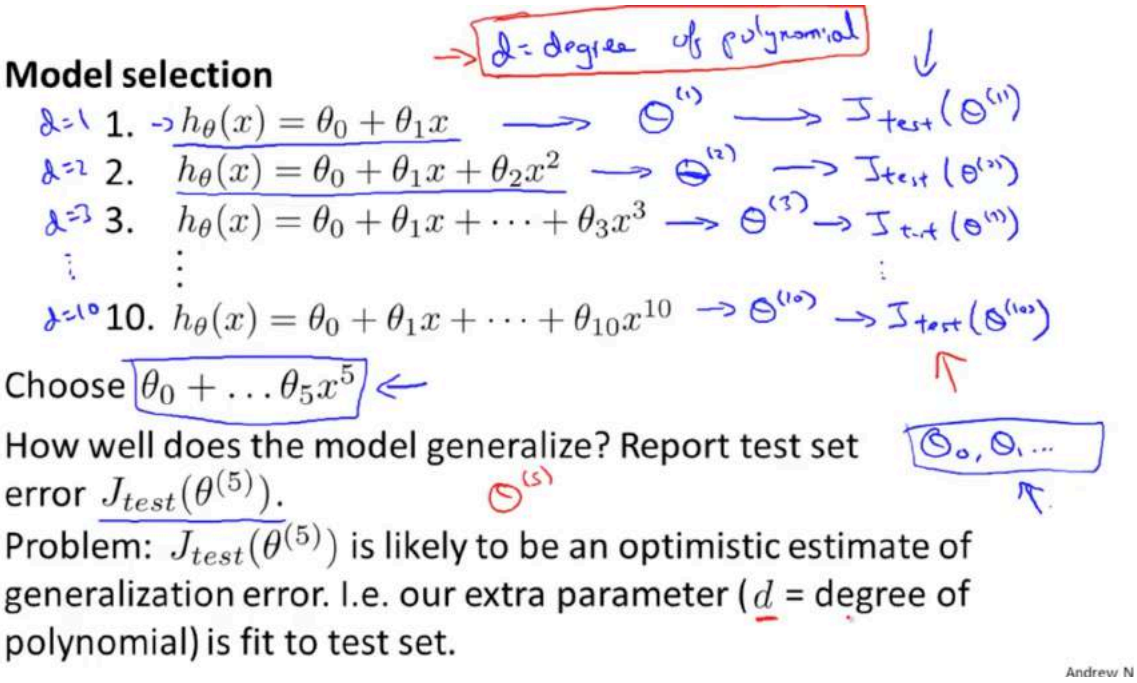
our discussion of how to do this, we'll talk about not just how to split your data into the train and test sets, but how to switch data into what we discover is called the train, validation, and test sets. We'll see in this video just what these things are, and how to use them to do model selection.

We've already seen a lot of times the problem of overfitting, in which just because a learning algorithm fits a training set well, that doesn't mean it's a good hypothesis. More generally, this is why the training set's error is not a good predictor for how well the hypothesis will do on new example. Concretely, if you fit some set of parameters. Theta0, theta1, theta2, and so on, to your training set. Then the fact that your hypothesis does well on the training set. Well, this doesn't mean much in terms of predicting how well your hypothesis will generalize to new examples not seen in the training set. And a more general principle is that once your parameter is what fit to some set of data. Maybe the training set, maybe something else. Then the error of your hypothesis as measured on that same data set, such as the training error, that's unlikely to be a good estimate of your actual generalization error. That is how well the hypothesis will generalize to new examples.

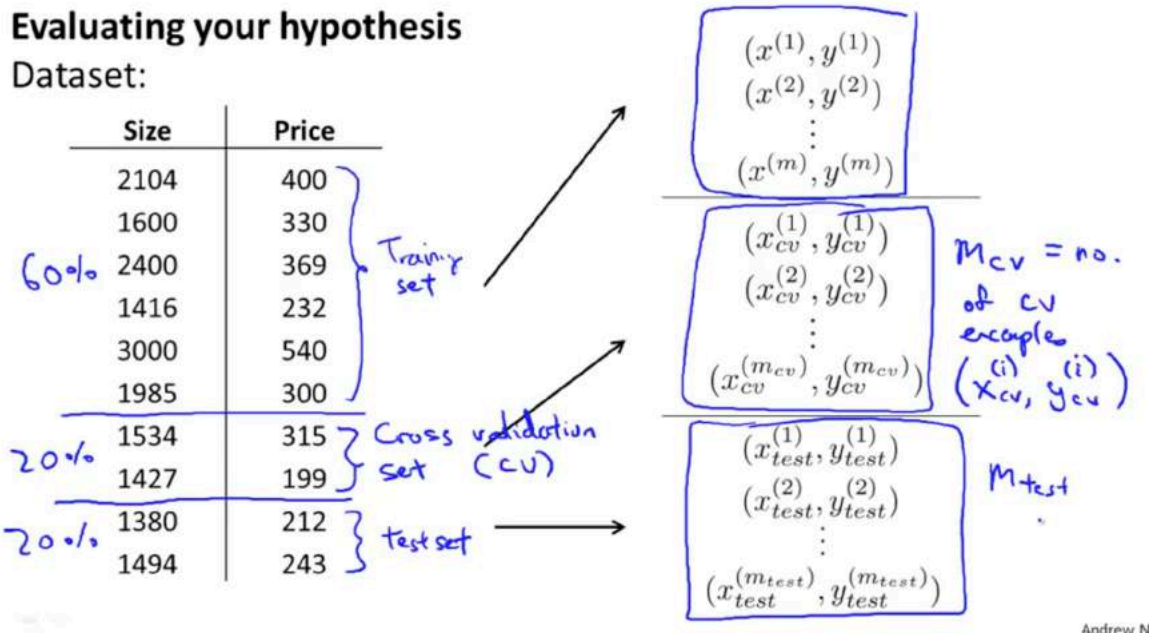


Now let's consider the model selection problem. Let's say you're trying to choose what degree polynomial to fit to data. So, should you choose a linear function, a quadratic function, a cubic function? All the way up to a 10th-order polynomial.So it's as if there's one extra parameter in this algorithm, which I'm going to denote d, which is, what degree of polynomial. Do you want to pick. So it's as if, in addition to the theta parameters, it's as if there's one more parameter, d, that you're trying to determine using your data set. So, the first option is d equals one, if you fit a linear function. We can choose d equals two, d equals three, all the way up to d equals 10. So, we'd like to fit this extra sort of parameter which I'm denoting by d. And concretely let's say that you want to

choose a model, that is choose a degree of polynomial, choose one of these 10 models. And fit that model and also get some estimate of how well your fitted hypothesis was generalize to new examples. Here's one thing you could do. What you could, first take your first model and minimize the training error. And this would give you some parameter vector theta. And you could then take your second model, the quadratic function, and fit that to your training set and this will give you some other. Parameter vector theta. In order to distinguish between these different parameter vectors, I'm going to use a superscript one superscript two there where theta superscript one just means the parameters I get by fitting this model to my training data. And theta superscript two just means the parameters I get by fitting this quadratic function to my training data and so on. By fitting a cubic model I get parenthesis three up to, well, say theta 10. And one thing we could do is that take these parameters and look at test error. So I can compute on my test set J test of one, J test of theta two, and so on. J test of theta three, and so on. So I'm going to take each of my hypotheses with the corresponding parameters and just measure the performance of on the test set. Now, one thing I could do then is, in order to select one of these models, I could then see which model has the lowest test set error. And let's just say for this example that I ended up choosing the fifth order polynomial. So, this seems reasonable so far. But now let's say I want to take my fifth hypothesis, this, this, fifth order model, and let's say I want to ask, how well does this model generalize? One thing I could do is look at how well my fifth order polynomial hypothesis had done on my test set. But the problem is this will not be a fair estimate of how well my hypothesis generalizes. And the reason is what we've done is we've fit this extra parameter d, that is this degree of polynomial. And what fits that parameter d, using the test set, namely, we chose the value of d that gave us the best possible performance on the test set. And so, the performance of my parameter vector theta5, on the test set, that's likely to be an overly optimistic estimate of generalization error. Right, so, that because I had fit this parameter d to my test set is no longer fair to evaluate my hypothesis on this test set, because I fit my parameters to this test set, I've chose the degree d of polynomial using the test set. And so my hypothesis is likely to do better on this test set than it would on new examples that it hasn't seen before, and that's which is, which is what I really care about. So just to reiterate, on the previous slide, we saw that if we fit some set of parameters, you know, say theta0, theta1, and so on, to some training set, then the performance of the fitted model on the training set is not predictive of how well the hypothesis will generalize to new examples. Is because these parameters were fit to the training set, so they're likely to do well on the training set, even if the parameters don't do well on other examples. And, in the procedure I just described on this line, we just did the same thing. And specifically, what we did was, we fit this parameter d to the test set. And by having fit the parameter to the test set, this means that the performance of the hypothesis on that test set may not be a fair estimate of how well the hypothesis is, is likely to do on examples we haven't seen before.



To address this problem, in a model selection setting, if we want to evaluate a hypothesis, this is what we usually do instead. Given the data set, instead of just splitting into a training test set, what we're going to do is then split it into three pieces. And the first piece is going to be called the training set as usual. So let me call this first part the training set. And the second piece of this data, I'm going to call the cross validation set. [SOUND] Cross validation. And the cross validation, as V-D. Sometimes it's also called the validation set instead of cross validation set. And then the loss can be to call the usual test set. And the pretty, pretty typical ratio at which to split these things will be to send 60% of your data's, your training set, maybe 20% to your cross validation set, and 20% to your test set. And these numbers can vary a little bit but this integration be pretty typical. And so our training sets will now be only maybe 60% of the data, and our cross-validation set, or our validation set, will have some number of examples. I'm going to denote that m subscript cv. So that's the number of cross-validation examples. Following our early notational convention I'm going to use xi cv comma y i cv, to denote the i cross validation example. And finally we also have a test set over here with our m subscript test being the number of test examples.



So, now that we've defined the training validation or cross validation and test sets. We can also define the training error, cross validation error, and test error. So here's my training error, and I'm just writing this as J subscript train of theta. This is pretty much the same things. These are the same thing as the J of theta that I've been writing so far, this is just a training set error you know, as measuring a training set and then J subscript cv my cross validation error, this is pretty much what you'd expect, just like the training error you've set measure it on a cross validation data set, and here's my test set error same as before.

Train/validation/test error

Training error:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$J(\theta)$

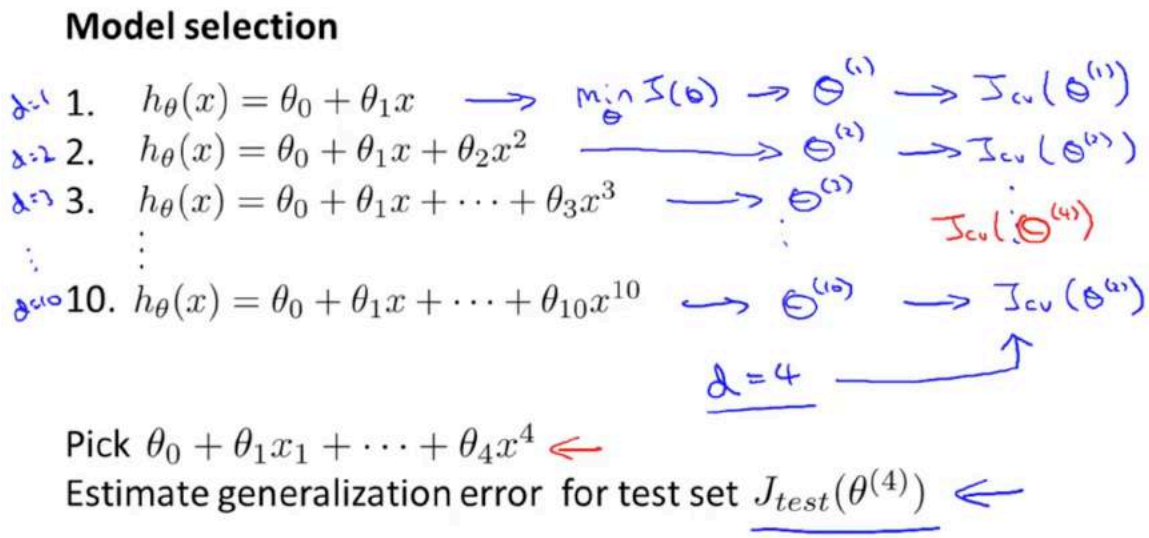
Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

So when faced with a model selection problem like this, what we're going to do is, instead of using the test set to select the model, we're instead going to use the validation set, or the cross validation set, to select the model. Concretely, we're going to first take our first hypothesis, take this first model, and say, minimize the cross function, and this would give me some parameter vector theta for the new model. And, as before, I'm going to put a superscript 1, just to denote that this is the parameter for the new model. We do the same thing for the quadratic model. Get some parameter vector theta two. Get some para, parameter vector theta three, and so on, down to theta ten for the polynomial. And what I'm going to do is, instead of testing these hypotheses on the test set, I'm instead going to test them on the cross validation set. And measure J subscript cv, to see how well each of these hypotheses do on my cross validation set. And then I'm going to pick the hypothesis with the lowest cross validation error. So for this example, let's say for the sake of argument, that it was my 4th order polynomial, that had the lowest cross validation error. So in that case I'm going to pick this fourth order polynomial model. And finally, what this means is that that parameter d, remember d was the degree of polynomial, right? So d equals two, d equals three, all the way up to d equals 10. What we've done is we'll fit that parameter d and we'll say d equals four. And we did so using the cross-validation set. And so this degree of polynomial, so the parameter, is no longer fit to the test set, and so we've not saved away the test set, and we can use the test set to measure, or to estimate the generalization error of the model that was selected. By the of them.



Question

Consider the model selection procedure where we choose the degree of polynomial using a cross validation set. For the final model (with parameters θ), we might generally expect $J_{CV}(\theta)$ To be lower than $J_{test}(\theta)$ because:

- (●) An extra parameter (d, the degree of the polynomial) has been fit to the cross validation set.
- ( ) An extra parameter (d, the degree of the polynomial) has been fit to the test set.
- ( ) The cross validation set is usually smaller than the test set.
- ( ) The cross validation set is usually larger than the test set.

So, that was model selection and how you can take your data, split it into a training, validation, and test set. And use your cross validation data to select the model and evaluate it on the test set. One final note, I should say that in. The machine learning, as of this practice today, there aren't many people that will do that early thing that I talked about, and said that, you know, it isn't such a good idea, of selecting your model using this test set. And then using the same test set to report the error as though selecting your degree of polynomial on the test set, and then reporting the error on the test set as though that were a good estimate of generalization error. That sort of practice is unfortunately many, many people do do it. If you have a massive, massive test that is maybe not a terrible thing to do, but many practitioners, most practitioners that machine learning tend to advise against that. And it's considered better practice to have separate train validation and test sets. I just warned you to sometimes people to do, you know, use the same data for the purpose of the validation set, and for the purpose of the test set. You need a training set and a test set, and that's good, that's practice, though you will see some people do it. But, if possible, I would recommend against doing that yourself.

Model Selection and Train/ Validation/Test Sets (Transcript)

Just because a learning algorithm fits a training set well, that does not mean it is a good hypothesis. It could over fit and as a result your predictions on the test set would be poor. The error of your hypothesis as measured on the data set with which you trained the parameters will be lower than the error on any other data set.

Given many models with different polynomial degrees, we can use a systematic approach to identify the 'best' function. In order to choose the model of your hypothesis, you can test each degree of polynomial and look at the error result.

One way to break down our dataset into the three sets is:

- Training set: 60%
- Cross validation set: 20%
- Test set: 20%

We can now calculate three separate error values for the three different sets using the following method:

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error);

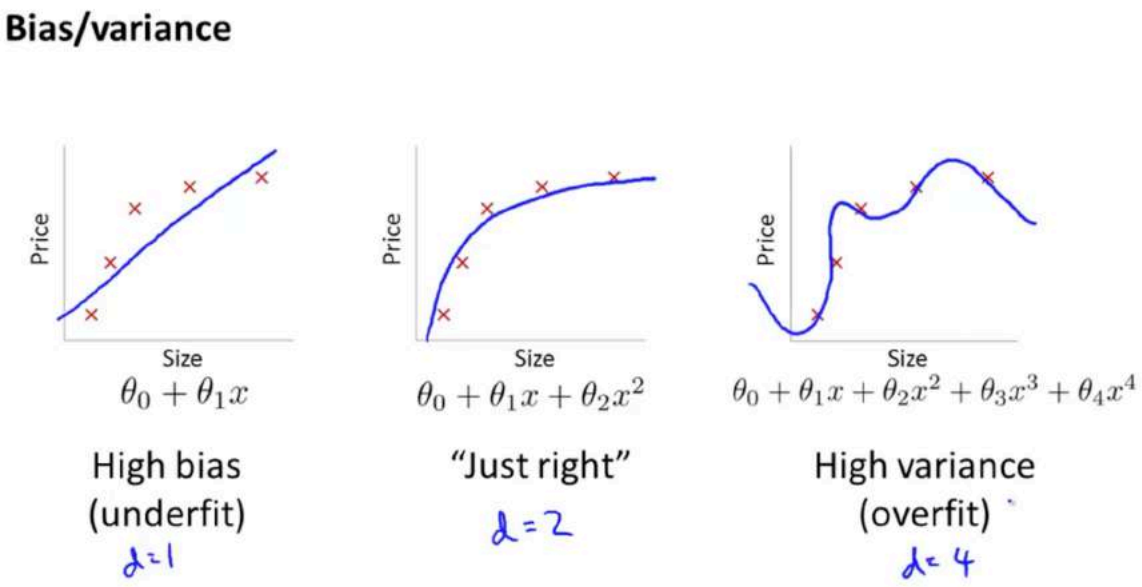
This way, the degree of the polynomial d has not been trained using the test set.

Bias vs. Variance

Diagnosing Bias vs. Variance (Video)

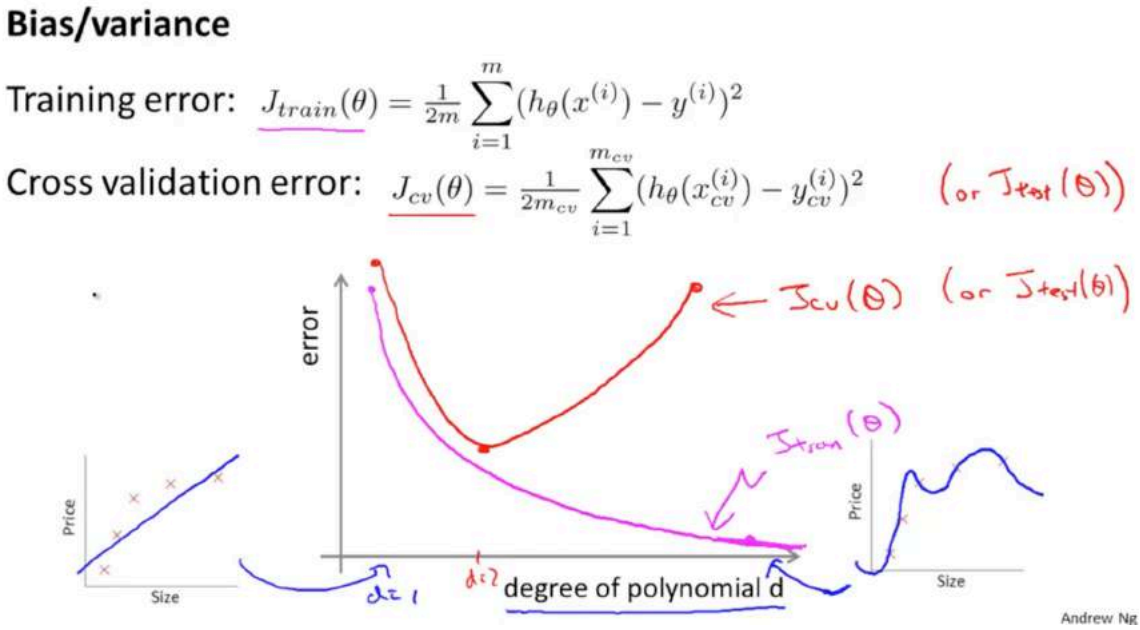
If you run the learning algorithm and it doesn't do as well as you are hoping, almost all the time it will be because you have either a high bias problem or a high variance problem. In other words they're either an underfitting problem or an overfitting problem. And in this case it's very important to figure out which of these two problems is bias or variance or a bit of both that you actually have. Because knowing which of these two things is happening would give a very strong indicator for whether the useful and promising ways to try to improve your algorithm. In this video, I would like to delve more deeply into this bias and various issue and understand them better as well as figure out how to

look at and evaluate knows whether or not we might have a bias problem or a variance problem. Since this would be critical to figuring out how to improve the performance of learning algorithm that you implement. So you've already seen this figure a few times, where if you fit two simple hypothesis, like a straight line that that underfits the data. If you fit a two complex hypothesis, then that might fit the training set perfectly but overfit the data and this may be hypothesis of some intermediate level of complexity, of some, maybe degree two polynomials are not too low and not too high degree. That's just right. And gives you the best generalization error out of these options.



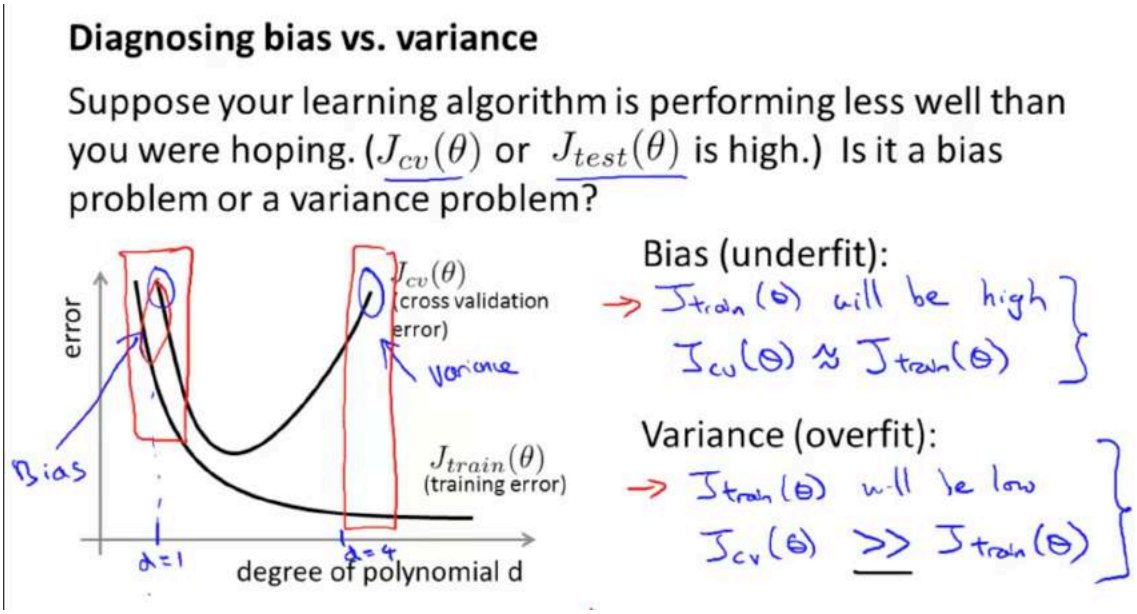
Now that we're armed with the notion of training and validation in test sets, we can understand the concepts of bias and variance a little bit better. Concretely, let our training error and cross validation error be defined as in the previous videos, just say, the squared error, the average squared error as measured on the 20 sets or as measured on the cross validation set. Now let's plot the following figure. On the horizontal axis I am going to plot the degree of polynomial, so as I go the right I'm going to be fitting higher and higher order polynomials. So, we'll do that for this figure, where maybe d equals 1, were going to be fitting very simple functions where as we are the right of this this may be d equals 4 or relatively may be even larger numbers. I'm going to be fitting very complex high order polynomials that might fit the training set with much more complex functions whereas we're here on the right of the horizontal axis, I have much larger values of these of a much higher degree polynomial, and so here that is going to correspond to fitting much more complex functions to your training set. Let's look at the training error and cause-validation error and plot them on this figure. Let's start with the training error. As we increase the degree of the polynomial, we're going to fit our training set better and better and so, if d equals 1 that ever rose to the high training error. If we have a

very high degree of polynomial, our training error is going to be really low. Maybe even zero, because it will fit the training set really well. And so as we increase of the greater polynomial we find typically that the training error decreases, so I'm going to write j subscript train of theta there, because our training error tends to decrease with the degree of the polynomial that we fit to the data. Next, let's look at the cross validation error. Often that matter, if we look at the test set error we'll get a pretty similar result as if we were to plot the cross validation error. So, we know that if d equals 1, we're fitting a very simple function, and so we may be underfitting the training set, and so we're going to go very high cross-validation error. If we fit, you know, an intermediate degree polynomial; we have a d equals 2 in our example in the previous slide, we are going to have a much lower cross-validation error, because we are just fitting, finding a much better fit to the data. And conversely if d were too high, so if d took on say a value of four, then we're again overfitting and so we end up with a high value for cross-validation error. So if you were to vary this smoothly and plot a curve you might end up with a curve like that, where that's Jcv of theta, and again if you plot j test of theta you get something very similar. And so this sort of plot also helps us to better understand the notions of bias and variance.



Concretely, if you have a learning algorithm that's not performing as well as you wanted it to, how can you figure out if your learning algorithm is suffering. Concretely, suppose you have applied a learning algorithm and it is not performing as well as your are hoping, so your cross-validation set error or your test set error is high. How can we figure out if the learning algorithm is suffering from high bias or if it is suffering from high variance. So the setting of a cross-validation error being high corresponds to either this regime or this regime. So this regime on the left corresponds to a high bias problem, that is, if

you are fitting an overly low order polynomial such as a plus one, when we really needed a higher order polynomial to fit the data. Whereas in contrast, this regime corresponds to a high variance problem. That is, if d--the degree of polynomial--was too large for the data set that we have. And this figure gives us a clue for how to distinguish between these two cases. Concretely, for the high bias case, that is, the case of under fitting, what we find is that both the cross validation error and the training error are going to be high. So, if your algorithm is suffering from a bias problem, the training set error would be high and you may find that the cross validation error will also be high. It might be close, maybe just slightly higher then a training error. And so, if you see this combination, that's a sign that your algorithm may be suffering from high bias. In contrast; if your algorithm is suffering from high variance; then, if you look here, we'll notice that, J train, that is the training error, is going to be low. That is, you're fitting the training set very well. Whereas, your cross validation error, assuming that this say the squared error which we're trying to minimize. Whereas in contrast; your error on a cross validation set or your cross function like cross validation set, will be much bigger than your training set error. This double greater than sign, here, it means much bigger than, all right. So, it's much greater than to multiply great to great. So this is a double greater than sign, that is the map symbol for much greater than denoted by two greater than signs. And so if you see this combination, then what you find. And so if you see this combination of values, then that is a clue that your learning algorithm may be suffering from high variance and might be overfitting. And the key that distinguishes these two cases is if you have a high bias problem your training set error will also be high as your hypothesis just not fitting the training set well. And if you have a high variance problem, your training set error will usually be low, that is much lower than the cross validation error.



Question

Suppose you have a classification problem. The (misclassification) error is defined as $\frac{1}{m}\sum_{i=1}^{m} \text{err}(h_\theta(x^{(i)}), y^{(i)})$, and the cross validation (misclassification) error is similarly defined, using the cross validation examples $(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$. Suppose your training error is 0.10, and your cross validation error is 0.30. What problem is the algorithm most likely to be suffering from?

- ○ High bias (overfitting)
- ○ High bias (underfitting)
- ◉ High variance (overfitting)
- ○ High variance (underfitting)

So, hopefully that gives you a somewhat better understanding of the two problems of bias and variance. I still have a lot more to say about bias and variance in the next few videos. But what we will see later; is that by diagnosing, whether a learning algorithm may be suffering from high bias or a high variance. I'll show you even more details on how to do that in later videos. We'll see that by figuring out whether a learning algorithm may be suffering from high bias or a combination of both that that would give us much better guidance for what might be promising things to try in order to improve the performance of the learning algorithm.

Diagnosing Bias vs. Variance (Transcript)

In this section we examine the relationship between the degree of the polynomial d and the underfitting or overfitting of our hypothesis.

- We need to distinguish whether **bias** or **variance** is the problem contributing to bad predictions.
- High bias is underfitting and high variance is overfitting. Ideally, we need to find a golden mean between these two.

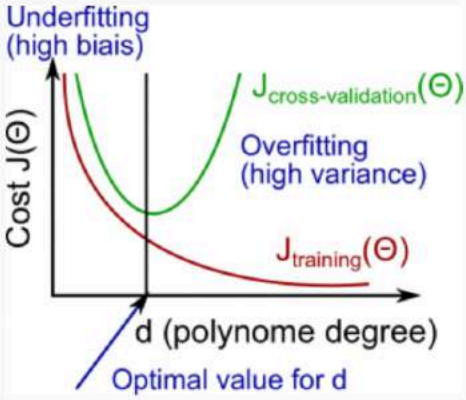
The training error will tend to **decrease** as we increase the degree d of the polynomial.

At the same time, the cross validation error will tend to **decrease** as we increase d up to a point, and then it will **increase** as d is increased, forming a convex curve.

High bias (underfitting): both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ will be high. Also, $J_{CV}(\Theta) \approx J_{train}(\Theta)$.

High variance (overfitting): $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be much greater than $J_{train}(\Theta)$.

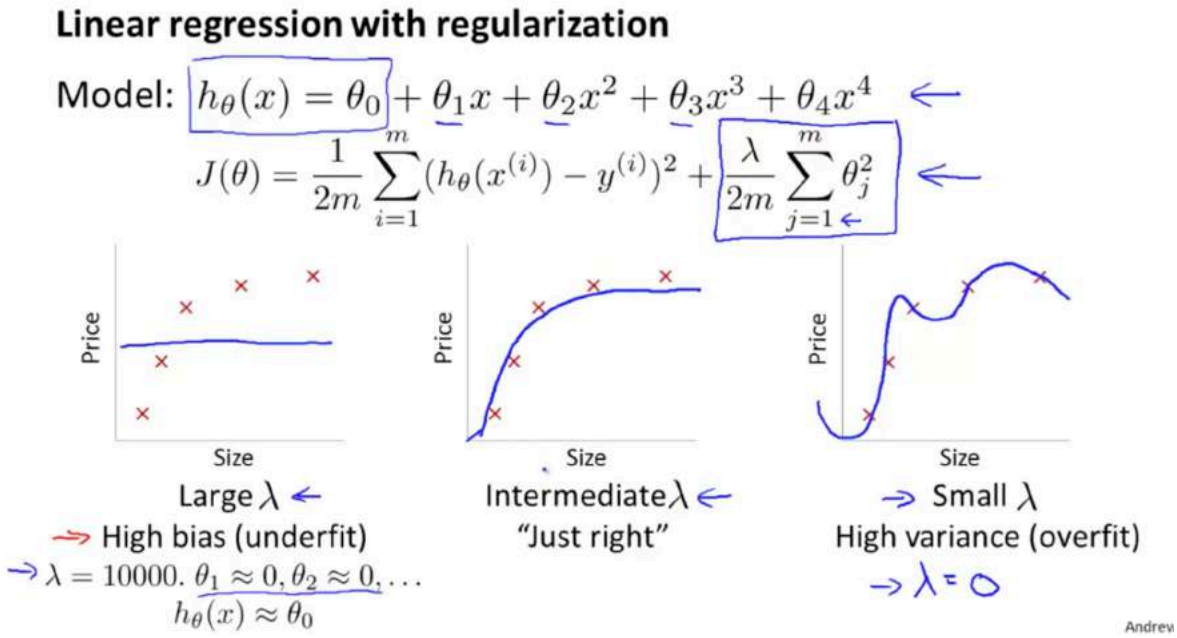
The is summarized in the figure below:



Regularization and Bias/Variance (Video)

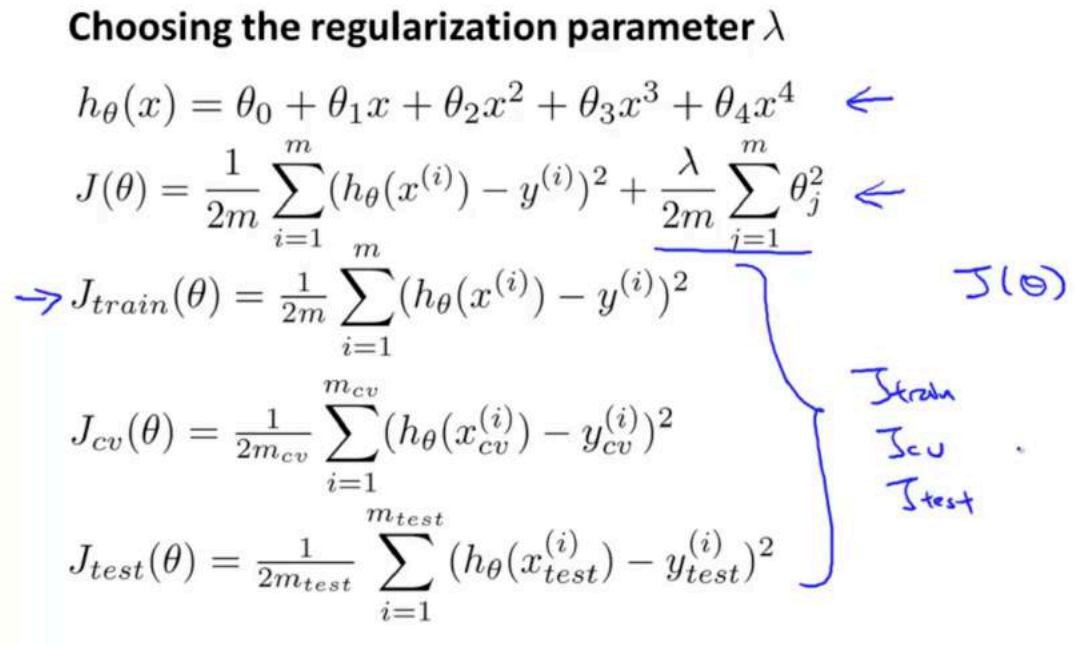
You've seen how regularization can help prevent over-fitting. But how does it affect the bias and variances of a learning algorithm? In this video I'd like to go deeper into the issue of bias and variances and talk about how it interacts with and is affected by the regularization of your learning algorithm. Suppose we're fitting a high auto polynomial, like that showed here, but to prevent over fitting we need to use regularization, like that shown here. So we have this regularization term to try to keep the values of the prem to small. And as usual, the regularizations comes from J = 1 to m, rather than j = 0 to m. Let's consider three cases. The first is the case of the very large value of the regularization parameter lambda, such as if lambda were equal to 10,000. Some huge value. In this case, all of these parameters, theta 1, theta 2, theta 3, and so on would be heavily penalized and so we end up with most of these parameter values

being closer to zero. And the hypothesis will be roughly h of x, just equal or approximately equal to theta zero. So we end up with a hypothesis that more or less looks like that, more or less a flat, constant straight line. And so this hypothesis has high bias and it badly under fits this data set, so the horizontal straight line is just not a very good model for this data set. At the other extreme is if we have a very small value of lambda, such as if lambda were equal to zero. In that case, given that we're fitting a high order polynomial, this is a usual over-fitting setting. In that case, given that we're fitting a high-order polynomial, basically, without regularization or with very minimal regularization, we end up with our usual high-variance, over fitting setting. This is basically if lambda is equal to zero, we're just fitting with our regularization, so that over fits the hypothesis. And it's only if we have some intermediate value of longer that is neither too large nor too small that we end up with parameters data that give us a reasonable fit to this data.



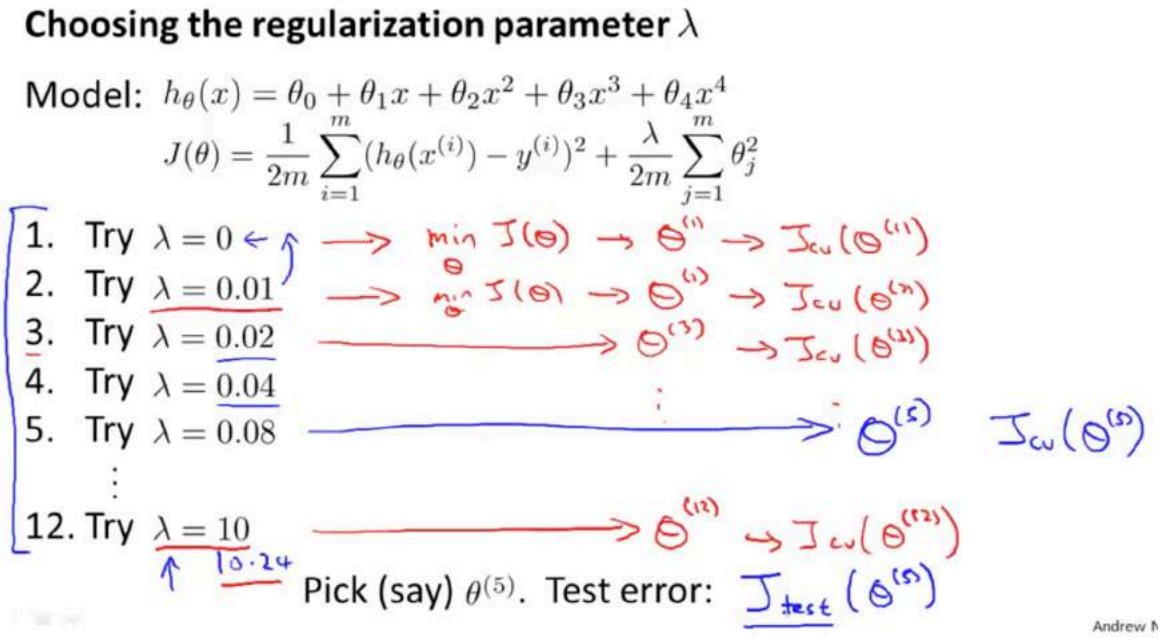
So, how can we automatically choose a good value for the regularization parameter? Just to reiterate, here's our model, and here's our learning algorithm's objective. For the setting where we're using regularization, let me define J train(theta) to be something different, to be the optimization objective, but without the regularization term. Previously, in an earlier video, when we were not using regularization I define J train of data to be the same as J of theta as the cause function but when we're using regularization when the six well under term we're going to define J train my training set to be just my sum of squared errors on the training set or my average squared error on the training set without taking into account that regularization. And similarly I'm then also going to define the cross validation sets error and to test that error as before to be the average sum of squared errors on the cross validation in the test sets so just to summarize my definitions of J train J CU and J test are

just the average square there one half of the other square record on the training validation of the test set without the extra regularization term.



So, this is how we can automatically choose the regularization parameter lambda. So what I usually do is maybe have some range of values of lambda I want to try out. So I might be considering not using regularization or here are a few values I might try lambda considering lambda = 0.01, 0.02, 0.04, and so on. And I usually set these up in multiples of two, until some maybe larger value if I were to do these in multiples of 2 I'd end up with a 10.24. It's 10 exactly, but this is close enough. And the three to four decimal places won't effect your result that much. So, this gives me maybe 12 different models. And I'm trying to select a month corresponding to 12 different values of the regularization of the parameter lambda. And of course you can also go to values less than 0.01 or values larger than 10 but I've just truncated it here for convenience. Given the issue of these 12 models, what we can do is then the following, we can take this first model with lambda equals zero and minimize my cost function J of data and this will give me some parameter of active data. And similar to the earlier video, let me just denote this as theta super script one. And then I can take my second model with lambda set to 0.01 and minimize my cost function now using lambda equals 0.01 of course. To get some different parameter vector theta. Let me denote that theta(2). And for that I end up with theta(3). So if part for my third model. And so on until for my final model with lambda set to 10 or 10.24, I end up with this theta(12). Next, I can talk all of these hypotheses, all of these parameters and use my cross validation set to validate them so I can look at my first model, my second model, fit to these different values of the regularization parameter, and evaluate them with my cross validation set based in measure the average

square error of each of these square vector parameters theta on my cross validation sets. And I would then pick whichever one of these 12 models gives me the lowest error on the trans validation set. And let's say, for the sake of this example, that I end up picking theta 5, the 5th order polynomial, because that has the lowest cause validation error. Having done that, finally what I would do if I wanted to report each test set error, is to take the parameter theta 5 that I've selected, and look at how well it does on my test set. So once again, here is as if we've fit this parameter, theta, to my cross-validation set, which is why I'm setting aside a separate test set that I'm going to use to get a better estimate of how well my parameter vector, theta, will generalize to previously unseen examples. So that's model selection applied to selecting the regularization parameter lambda.

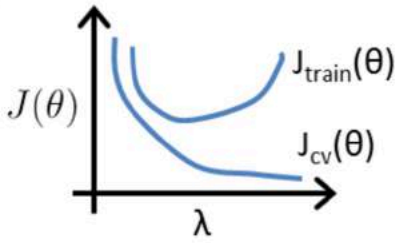


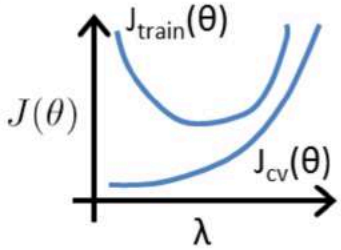
Question

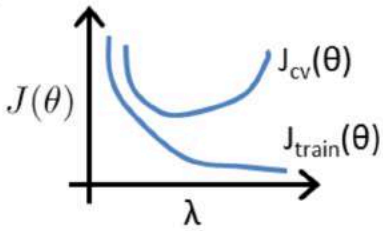
Consider regularized logistic regression. Let

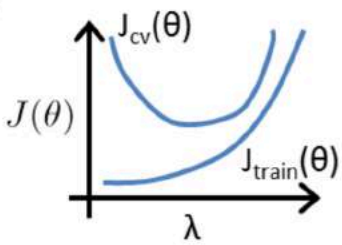
- $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=2}^{n}\theta_j^2\right]$
- $J_{\text{train}}(\theta) = \frac{1}{2m_{\text{train}}}\left[\sum_{i=1}^{m_{\text{train}}}(h_\theta(x_{\text{train}}^{(i)}) - y_{\text{train}}^{(i)})^2\right]$
- $J_{\text{CV}}(\theta) = \frac{1}{2m_{\text{CV}}}\left[\sum_{i=1}^{m_{\text{CV}}}(h_\theta(x_{\text{CV}}^{(i)}) - y_{\text{CV}}^{(i)})^2\right]$

Suppose you plot J_{train} and J_{CV} as a function of the regularization parameter λ. which of the following plots do you expect to get?





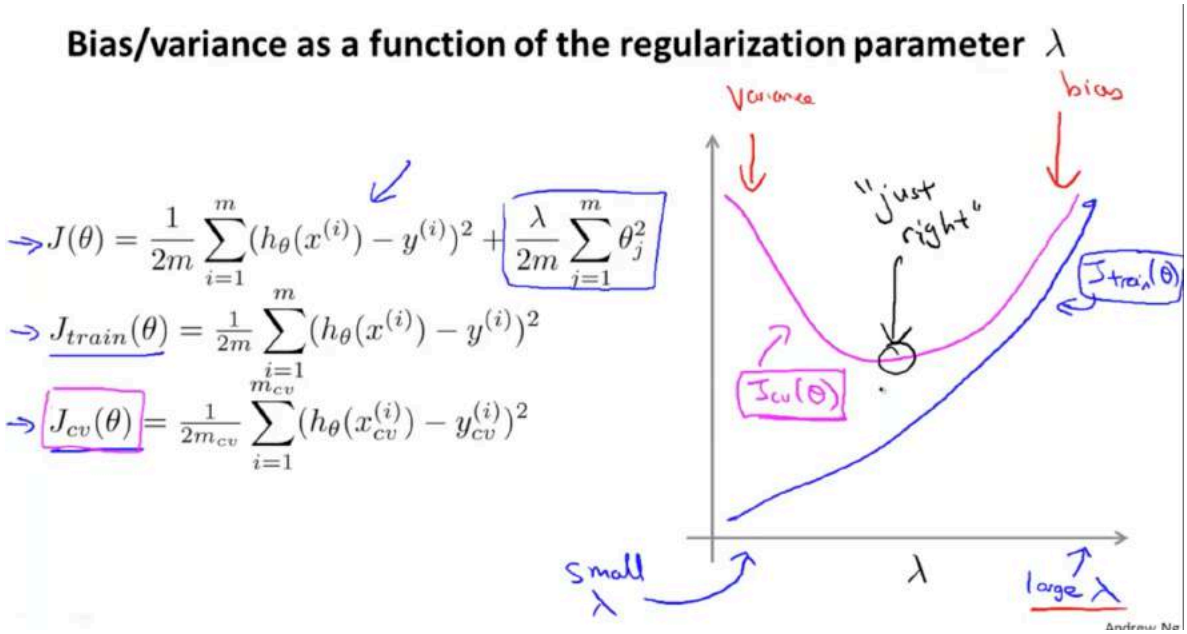




Correct

The last thing I'd like to do in this video is get a better understanding of how cross validation and training error vary as we vary the regularization parameter lambda. And so just a reminder right, that was our original cost on j of theta. But for this purpose we're going to define training error without using

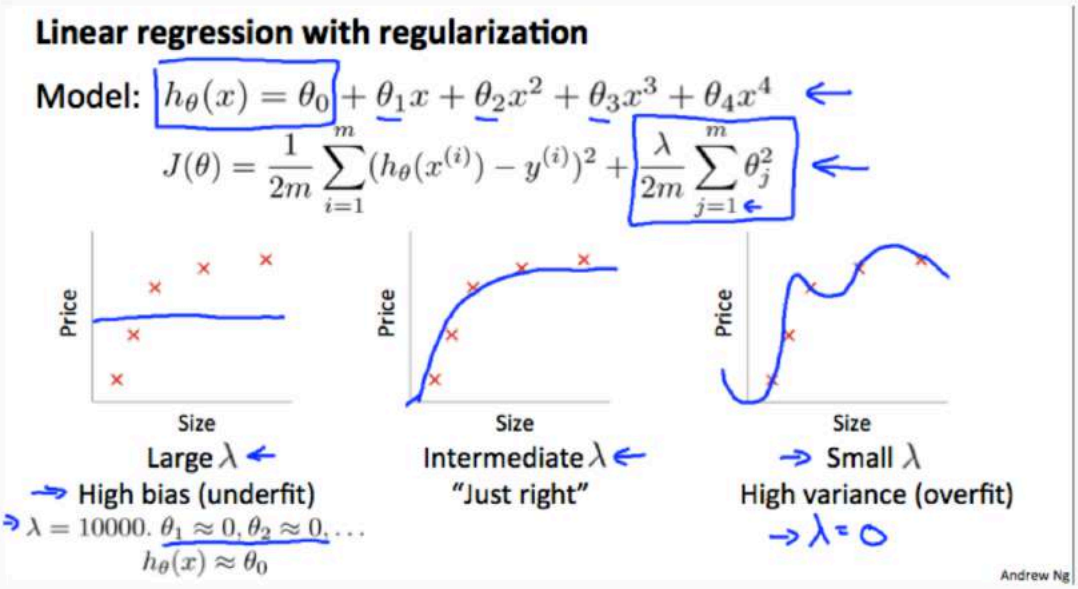
a regularization parameter, and cross validation error without using the regularization parameter. And what I'd like to do is plot this Jtrain and plot this Jcv, meaning just how well does my hypothesis do on the training set and how does my hypothesis do when it cross validation sets. As I vary my regularization parameter lambda. So as we saw earlier if lambda is small then we're not using much regularization and we run a larger risk of over fitting whereas if lambda is large that is if we were on the right part of this horizontal axis then, with a large value of lambda, we run the higher risk of having a biased problem, so if you plot J train and J cv, what you find is that, for small values of lambda, you can fit the trading set relatively way cuz you're not regularizing. So, for small values of lambda, the regularization term basically goes away, and you're just minimizing pretty much just gray arrows. So when lambda is small, you end up with a small value for Jtrain, whereas if lambda is large, then you have a high bias problem, and you might not feel your training that well, so you end up the value up there. So Jtrain of theta will tend to increase when lambda increases, because a large value of lambda corresponds to high bias where you might not even fit your trainings that well, whereas a small value of lambda corresponds to, if you can really fit a very high degree polynomial to your data, let's say. After the cost validation error we end up with a figure like this, where over here on the right, if we have a large value of lambda, we may end up under fitting, and so this is the bias regime. And so the cross validation error will be high. Let me just leave all of that to this Jcv (theta) because so, with high bias, we won't be fitting, we won't be doing well in cross validation sets, whereas here on the left, this is the high variance regime, where we have two smaller value with longer, then we may be over fitting the data. And so by over fitting the data, then the cross validation error will also be high. And so, this is what the cross validation error and what the trading error may look like on a trading stance as we vary the regularization parameter lambda. And so once again, it will often be some intermediate value of lambda that is just right or that works best In terms of having a small cross validation error or a small test theta. And whereas the curves I've drawn here are somewhat cartoonish and somewhat idealized so on the real data set the curves you get may end up looking a little bit more messy and just a little bit more noisy then this. For some data sets you will really see these for sorts of trends and by looking at a plot of the hold-out cross validation error you can either manual, automatically try to select a point that minimizes the cross validation error and select the value of lambda corresponding to low cross validation error. When I'm trying to pick the regularization parameter lambda for learning algorithm, often I find that plotting a figure like this one shown here helps me understand better what's going on and helps me verify that I am indeed picking a good value for the regularization parameter monitor.



So hopefully that gives you more insight into regularization and it's effects on the bias and variance of a learning algorithm. By now you've seen bias and variance from a lot of different perspectives. And what we like to do in the next video is take all the insights we've gone through and build on them to put together a diagnostic that's called learning curves, which is a tool that I often use to diagnose if the learning algorithm may be suffering from a bias problem or a variance problem, or a little bit of both.

Regularization and Bias/Variance (Transcript)

Note: [The regularization term below and through out the video should be $\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$ and **NOT** $\frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$]



In the figure above, we see that as λ increases, our fit becomes more rigid. On the other hand, as λ approaches 0, we tend to over overfit the data. So how do we choose our parameter λ to get it 'just right' ? In order to choose the model and the regularization term λ, we need to:

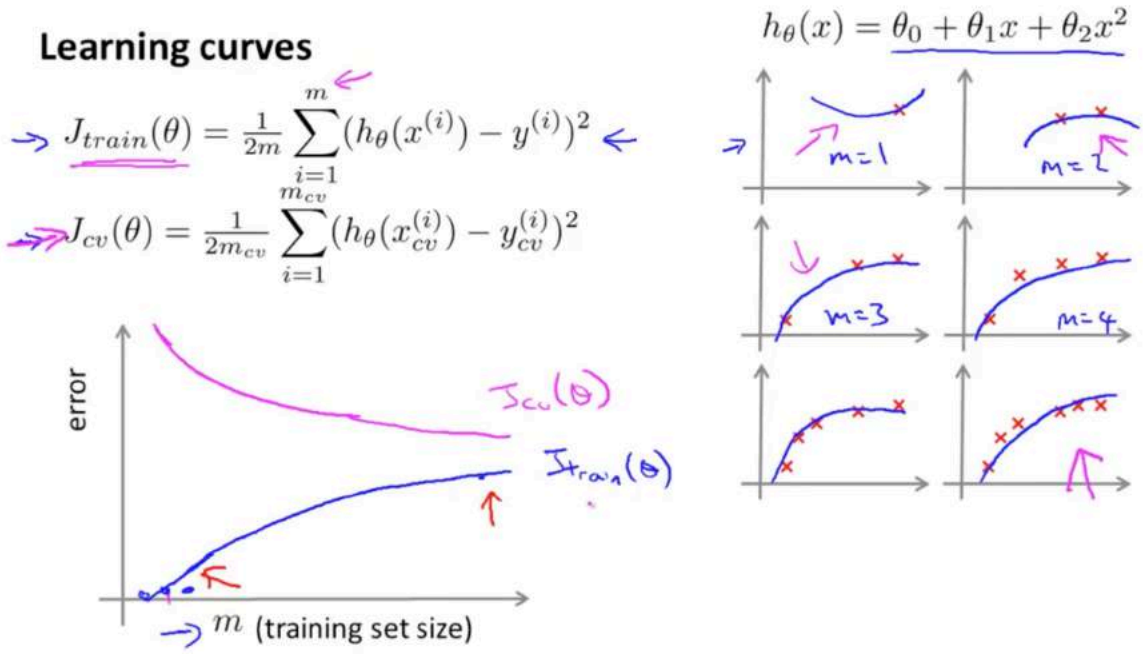
1. Create a list of lambdas (i.e. λ∈{0,0.01,0.02,0.04,0.08,0.16,0.32,0.64,1.28,2.56,5.12,10.24});
2. Create a set of models with different degrees or any other variants.
3. Iterate through the λs and for each λ go through all the models to learn some Θ.
4. Compute the cross validation error using the learned Θ (computed with λ) on the $J_{CV}(\Theta)$ **without** regularization or λ = 0.
5. Select the best combo that produces the lowest error on the cross validation set.
6. Using the best combo Θ and λ, apply it on $J_{test}(\Theta)$ to see if it has a good generalization of the problem.

Learning Curves (Video)

In this video, I'd like to tell you about learning curves. Learning curves is often a very useful thing to plot. If either you wanted to sanity check that your algorithm is working correctly, or if you want to improve the performance of the algorithm. And learning curves is a tool that I actually use very often to try to diagnose if a physical learning algorithm may be suffering from bias, sort of variance problem or a bit of both. Here's what a learning curve is. To plot a learning curve, what I usually do is plot j train which is, say, average squared error on my training set or Jcv which is the average squared error on my cross validation set. And I'm going to plot that as a function of m, that is as a function of the number of training examples I have. And so m is usually a

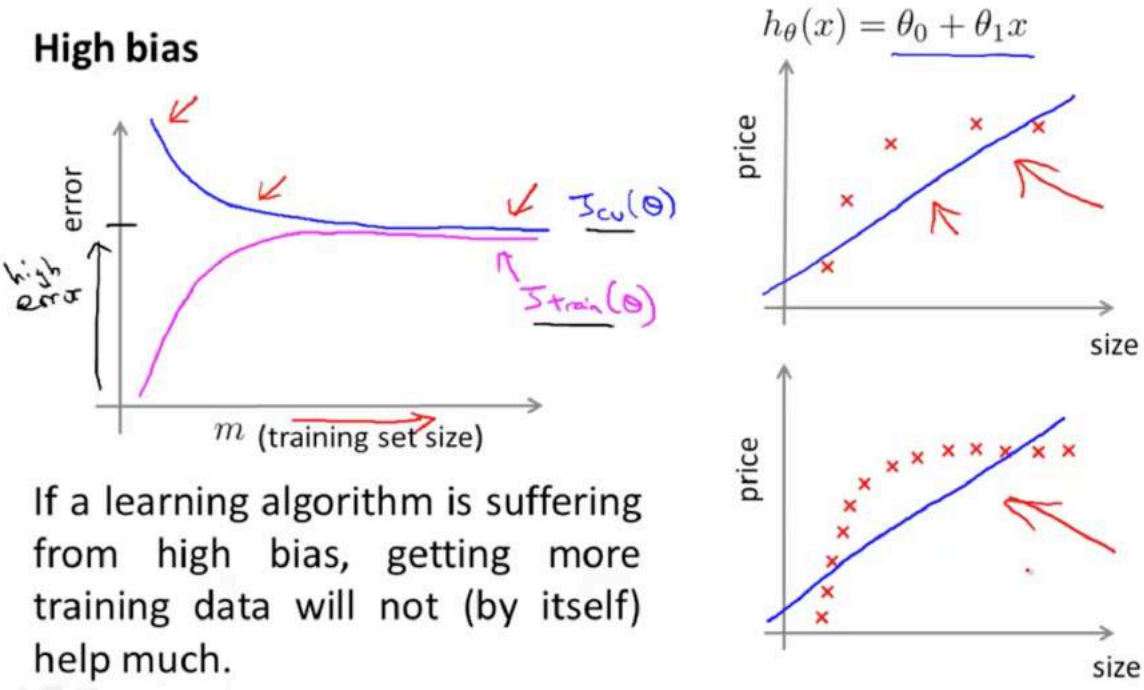
constant like maybe I just have, you know, a 100 training examples but what I'm going to do is artificially with use my training set exercise. So, I deliberately limit myself to using only, say, 10 or 20 or 30 or 40 training examples and plot what the training error is and what the cross validation is for this smallest training set exercises. So let's see what these plots may look like. Suppose I have only one training example like that shown in this this first example here and let's say I'm fitting a quadratic function. Well, I have only one training example. I'm going to be able to fit it perfectly right? You know, just fit the quadratic function. I'm going to have 0 error on the one training example. If I have two training examples. Well the quadratic function can also fit that very well. So, even if I am using regularization, I can probably fit this quite well. And if I am using no neural regularization, I'm going to fit this perfectly and if I have three training examples again. Yeah, I can fit a quadratic function perfectly so if m equals 1 or m equals 2 or m equals 3, my training error on my training set is going to be 0 assuming I'm not using regularization or it may slightly large in 0 if I'm using regularization and by the way if I have a large training set and I'm artificially restricting the size of my training set in order to J train. Here if I set M equals 3, say, and I train on only three examples, then, for this figure I am going to measure my training error only on the three examples that actually fit my data too and so even I have to say a 100 training examples but if I want to plot what my training error is the m equals 3. What I'm going to do is to measure the training error on the three examples that I've actually fit to my hypothesis 2. And not all the other examples that I have deliberately omitted from the training process. So just to summarize what we've seen is that if the training set size is small then the training error is going to be small as well. Because you know, we have a small training set is going to be very easy to fit your training set very well may be even perfectly now say we have m equals 4 for example. Well then a quadratic function can be a longer fit this data set perfectly and if I have m equals 5 then you know, maybe quadratic function will fit to stay there so so, then as my training set gets larger. It becomes harder and harder to ensure that I can find the quadratic function that process through all my examples perfectly. So in fact as the training set size grows what you find is that my average training error actually increases and so if you plot this figure what you find is that the training set error that is the average error on your hypothesis grows as m grows and just to repeat when the intuition is that when m is small when you have very few training examples. It's pretty easy to fit every single one of your training examples perfectly and so your error is going to be small whereas when m is larger then gets harder all the training examples perfectly and so your training set error becomes more larger now, how about the cross validation error. Well, the cross validation is my error on this cross validation set that I haven't seen and so, you know, when I have a very small training set, I'm not going to generalize well, just not going to do well on that. So, right, this hypothesis here doesn't look like a good one, and it's only when I get a larger training set that, you know, I'm starting to get hypotheses that maybe fit the data somewhat better. So your cross validation error and your test set error will tend to decrease as your training set size

increases because the more data you have, the better you do at generalizing to new examples. So, just the more data you have, the better the hypothesis you fit. So if you plot j train, and Jcv this is the sort of thing that you get.



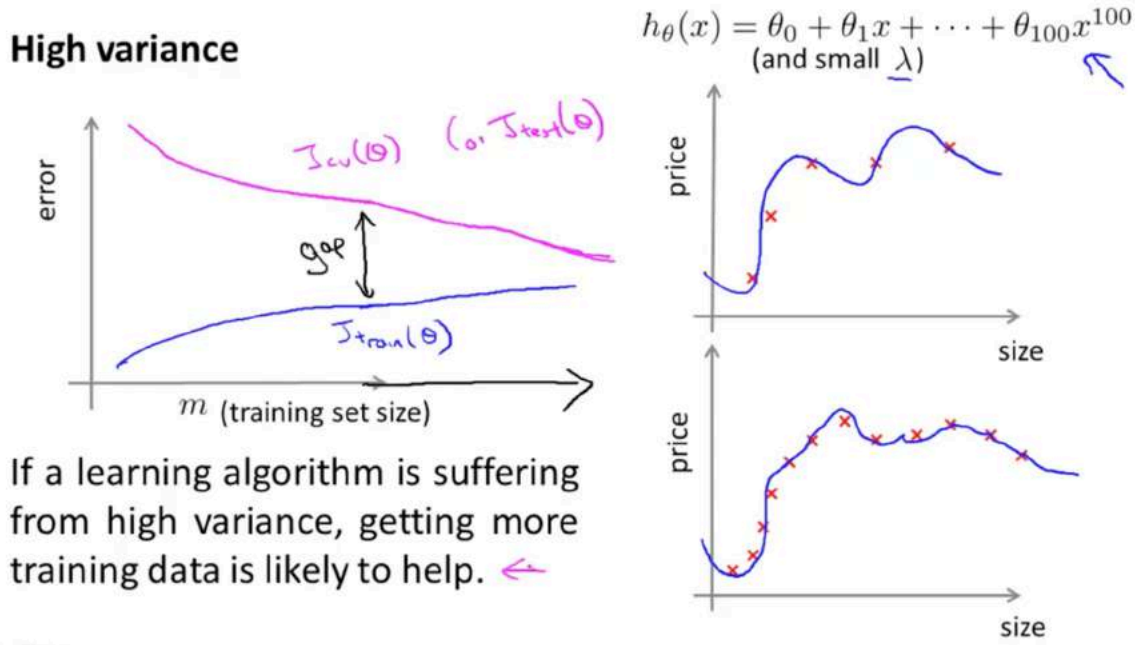
Now let's look at what the learning curves may look like if we have either high bias or high variance problems. Suppose your hypothesis has high bias and to explain this I'm going to use a, set an example, of fitting a straight line to data that, you know, can't really be fit well by a straight line. Now let's think what would happen if we were to increase the training set size. So if instead of five examples like what I've drawn there, imagine that we have a lot more training examples. Well what happens, if you fit a straight line to this. What you find is that, you end up with you know, pretty much the same straight line. I mean a straight line that just cannot fit this data and getting a ton more data, well the straight line isn't going to change that much. This is the best possible straight-line fit to this data, but the straight line just can't fit this data set that well. So, if you plot across validation error, this is what it will look like. Option on the left, if you have already a minuscule training set size like you know, maybe just one training example and is not going to do well. But by the time you have reached a certain number of training examples, you have almost fit the best possible straight line, and even if you end up with a much larger training set size, a much larger value of m, you know, you're basically getting the same straight line, and so, the cross-validation error - let me label that - or test set error or plateau out, or flatten out pretty soon, once you reached beyond a certain the number of training examples, unless you pretty much fit the best possible straight line. And how about training error? Well, the training error will again be small. And what you find in the high bias case is that the training error will end up close to the cross validation error, because you have so few parameters

and so much data, at least when m is large. The performance on the training set and the cross validation set will be very similar. And so, this is what your learning curves will look like, if you have an algorithm that has high bias. And finally, the problem with high bias is reflected in the fact that both the cross validation error and the training error are high, and so you end up with a relatively high value of both Jcv and the j train. This also implies something very interesting, which is that, if a learning algorithm has high bias, as we get more and more training examples, that is, as we move to the right of this figure, we'll notice that the cross validation error isn't going down much, it's basically fattened up, and so if learning algorithms are really suffering from high bias. Getting more training data by itself will actually not help that much, and as our figure example in the figure on the right, here we had only five training. examples, and we fill certain straight line. And when we had a ton more training data, we still end up with roughly the same straight line. And so if the learning algorithm has high bias give me a lot more training data. That doesn't actually help you get a much lower cross validation error or test set error. So knowing if your learning algorithm is suffering from high bias seems like a useful thing to know because this can prevent you from wasting a lot of time collecting more training data where it might just not end up being helpful.



Next let us look at the setting of a learning algorithm that may have high variance. Let us just look at the training error in a around if you have very smart training set like five training examples shown on the figure on the right and if we're fitting say a very high order polynomial, and I've written a hundredth degree polynomial which really no one uses, but just an illustration.

And if we're using a fairly small value of lambda, maybe not zero, but a fairly small value of lambda, then we'll end up, you know, fitting this data very well that with a function that overfits this. So, if the training set size is small, our training error, that is, j train of theta will be small. And as this training set size increases a bit, you know, we may still be overfitting this data a little bit but it also becomes slightly harder to fit this data set perfectly, and so, as the training set size increases, we'll find that j train increases, because it is just a little harder to fit the training set perfectly when we have more examples, but the training set error will still be pretty low. Now, how about the cross validation error? Well, in high variance setting, a hypothesis is overfitting and so the cross validation error will remain high, even as we get you know, a moderate number of training examples and, so maybe, the cross validation error may look like that. And the indicative diagnostic that we have a high variance problem, is the fact that there's this large gap between the training error and the cross validation error. And looking at this figure. If we think about adding more training data, that is, taking this figure and extrapolating to the right, we can kind of tell that, you know the two curves, the blue curve and the magenta curve, are converging to each other. And so, if we were to extrapolate this figure to the right, then it seems it likely that the training error will keep on going up and the cross-validation error would keep on going down. And the thing we really care about is the cross-validation error or the test set error, right? So in this sort of figure, we can tell that if we keep on adding training examples and extrapolate to the right, well our cross validation error will keep on coming down. And, so, in the high variance setting, getting more training data is, indeed, likely to help. And so again, this seems like a useful thing to know if your learning algorithm is suffering from a high variance problem, because that tells you, for example that it may be be worth your while to see if you can go and get some more training data. Now, on the previous slide and this slide, I've drawn fairly clean fairly idealized curves. If you plot these curves for an actual learning algorithm, sometimes you will actually see, you know, pretty much curves, like what I've drawn here. Although, sometimes you see curves that are a little bit noisier and a little bit messier than this. But plotting learning curves like these can often tell you, can often help you figure out if your learning algorithm is suffering from bias, or variance or even a little bit of both. So when I'm trying to improve the performance of a learning algorithm, one thing that I'll almost always do is plot these learning curves, and usually this will give you a better sense of whether there is a bias or variance And in the next video we'll see how this can help suggest specific actions is to take, or to not take, in order to try to improve the performance of your learning algorithm.



Question

In which of the following circumstances is getting more training data likely to significantly help a learning algorithm's performance?

- [ ] Algorithm is suffering from high bias.

Un-selected is correct

- [x] Algorithm is suffering from high variance.

Correct

- [x] $J_{\text{CV}}(\theta)$ (cross validation error) is much larger than $J_{\text{train}}(\theta)$ (training error).

Correct

- [ ] $J_{\text{CV}}(\theta)$ (cross validation error) is about the same as $J_{\text{train}}(\theta)$ (training error).

Un-selected is correct

Learning Curves (Transcript)

Training an algorithm on a very few number of data points (such as 1, 2 or 3) will easily have 0 errors because we can always find a quadratic curve that touches exactly those number of points. Hence:

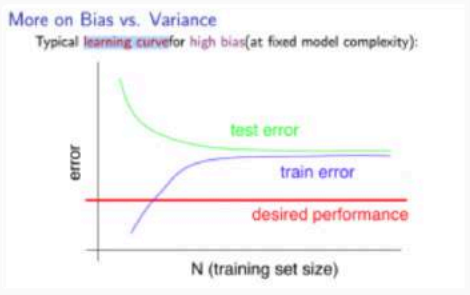
- As the training set gets larger, the error for a quadratic function increases.
- The error value will plateau out after a certain m, or training set size.

Experiencing high bias:

Low training set size: causes $J_{train}(\Theta)$ to be low and $J_{CV}(\Theta)$ to be high.

Large training set size: causes both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ to be high with $J_{train}(\Theta) \approx J_{CV}(\Theta)$.

If a learning algorithm is suffering from **high bias**, getting more training data will not **(by itself)** help much.

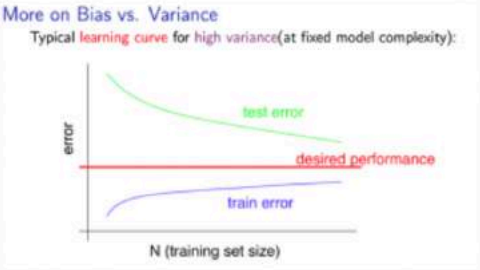


Experiencing high variance:

Low training set size: $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be high.

Large training set size: $J_{train}(\Theta)$ increases with training set size and $J_{CV}(\Theta)$ continues to decrease without leveling off. Also, $J_{train}(\Theta) < J_{CV}(\Theta)$ but the difference between them remains significant.

If a learning algorithm is suffering from **high variance**, getting more training data is likely to help.



Deciding What to do Next Revisited

(Video)

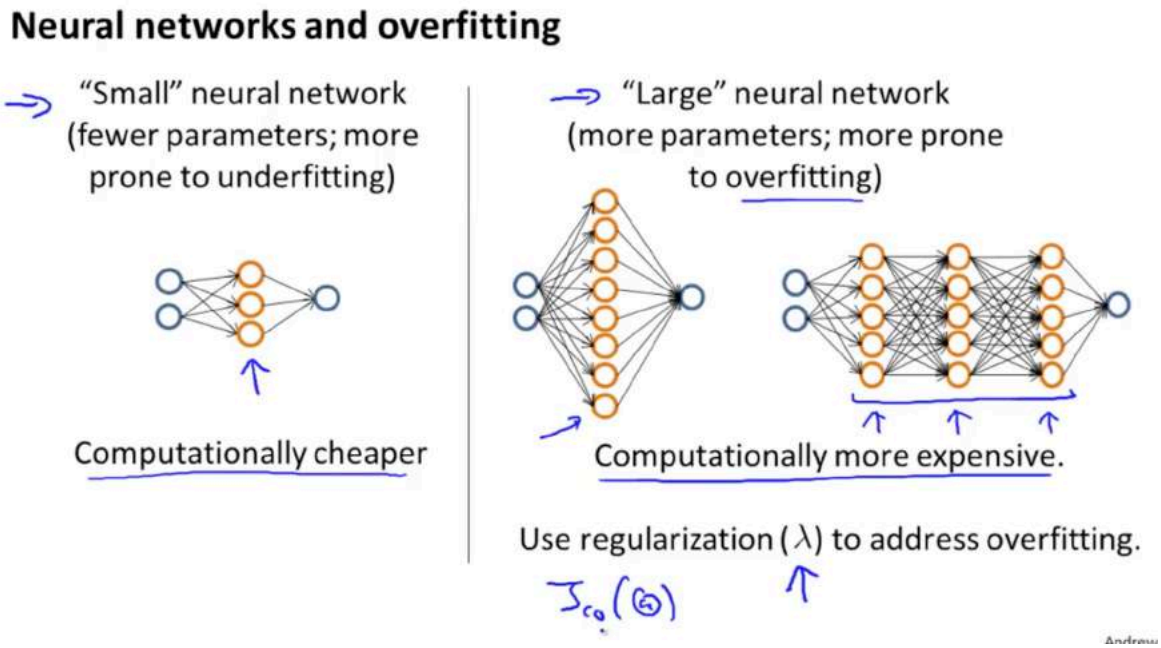
We've talked about how to evaluate learning algorithms, talked about model selection, talked a lot about bias and variance. So how does this help us figure out what are potentially fruitful, potentially not fruitful things to try to do to improve the performance of a learning algorithm. Let's go back to our original motivating example and go for the result. So here is our earlier example of maybe having fit regularized linear regression and finding that it doesn't work as well as we're hoping. We said that we had this menu of options. So is there some way to figure out which of these might be fruitful options? The first thing all of this was getting more training examples. What this is good for, is this helps to fix high variance. And concretely, if you instead have a high bias problem and don't have any variance problem, then we saw in the previous video that getting more training examples, while maybe just isn't going to help much at all. So the first option is useful only if you, say, plot the learning curves and figure out that you have at least a bit of a variance, meaning that the cross-validation error is, you know, quite a bit bigger than your training set error. How about trying a smaller set of features? Well, trying a smaller set of features, that's again something that fixes high variance. And in other words, if you figure out, by looking at learning curves or something else that you used, that have a high bias problem; then for goodness sakes, don't waste your time trying to carefully select out a smaller set of features to use. Because if you have a high bias problem, using fewer features is not going to help. Whereas in contrast, if you look at the learning curves or something else you figure out that you have a high variance problem, then, indeed trying to select out a smaller set of features, that might indeed be a very good use of your time. How about trying to get additional features, adding features, usually, not always, but usually we think of this as a solution for fixing high bias problems. So if you are adding extra features it's usually because your current hypothesis is too simple, and so we want to try to get additional features to make our hypothesis better able to fit the training set. And similarly, adding polynomial features; this is another way of adding features and so there is another way to try to fix the high bias And, if concretely if your learning curves show you that you still have a high variance problem, then, you know, again this is maybe a less good use of your And finally, decreasing and increasing lambda. This are quick and easy to try, I guess these are less likely to be a waste of, you know, many months of your life. But decreasing lambda, you already know fixes high bias. In case this isn't clear to you, you know, I do encourage you to pause the video and think through this that convince yourself that decreasing lambda helps fix high bias, whereas increasing lambda fixes high variance. And if you aren't sure why this is the case, do pause the video and make sure you can convince yourself that this is the case. Or take a look at the curves that we were plotting at the end of the previous video and try to make sure you understand why these are the case.

Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc})$ → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

Finally, let us take everything we have learned and relate it back to neural networks and so, here is some practical advice for how I usually choose the architecture or the connectivity pattern of the neural networks I use. So, if you are fitting a neural network, one option would be to fit, say, a pretty small neural network with you know, relatively few hidden units, maybe just one hidden unit. If you're fitting a neural network, one option would be to fit a relatively small neural network with, say, relatively few, maybe only one hidden layer and maybe only a relatively few number of hidden units. So, a network like this might have relatively few parameters and be more prone to underfitting. The main advantage of these small neural networks is that the computation will be cheaper. An alternative would be to fit a, maybe relatively large neural network with either more hidden units--there's a lot of hidden in one there--or with more hidden layers. And so these neural networks tend to have more parameters and therefore be more prone to overfitting. One disadvantage, often not a major one but something to think about, is that if you have a large number of neurons in your network, then it can be more computationally expensive. Although within reason, this is often hopefully not a huge problem. The main potential problem of these much larger neural networks is that it could be more prone to overfitting and it turns out if you're applying neural network very often using a large neural network often it's actually the larger, the better but if it's overfitting, you can then use regularization to address overfitting, usually using a larger neural network by using regularization to address is overfitting that's often more effective than using a smaller neural network. And the main possible disadvantage is that it can be more computationally expensive. And finally, one of the other decisions is, say, the number of hidden layers you want to have, right? So, do you want one hidden layer or do you want three hidden layers, as we've shown here, or do you want two hidden layers? And usually, as I think I said in the previous video, using a single hidden layer is a reasonable default, but if you want to

choose the number of hidden layers, one other thing you can try is find yourself a training cross-validation, and test set split and try training neural networks with one hidden layer or two hidden layers or three hidden layers and see which of those neural networks performs best on the cross-validation sets. You take your three neural networks with one, two and three hidden layers, and compute the cross validation error at Jcv and all of them and use that to select which of these is you think the best neural network.



Question

Suppose you fit a neural network with one hidden layer to a training set. You find that the cross validation error $J_{\text{CV}}(\theta)$ is much larger than the training error $J_{\text{train}}(\theta)$. Is increasing the number of hidden units likely to help?

- Yes, because this increases the number of parameters and lets the network represent more complex functions.
- Yes, because it is currently suffering from high bias.
- No, because it is currently suffering from high bias, so adding hidden units is unlikely to help.
- **No, because it is currently suffering from high variance, so adding hidden units is unlikely to help.**

So, that's it for bias and variance and ways like learning curves, who tried to diagnose these problems. As far as what you think is implied, for one might be

truthful or not truthful things to try to improve the performance of a learning algorithm. If you understood the contents of the last few videos and if you apply them you actually be much more effective already and getting learning algorithms to work on problems and even a large fraction, maybe the majority of practitioners of machine learning here in Silicon Valley today doing these things as their full-time jobs. So I hope that these pieces of advice on by experience in diagnostics will help you to much effectively and powerfully apply learning and get them to work very well.

Deciding What to do Next Revisited (Transcript)

Our decision process can be broken down as follows:

- **Getting more training examples:** Fixes high variance
- **Trying smaller sets of features:** Fixes high variance
- **Adding features:** Fixes high bias
- **Adding polynomial features:** Fixes high bias
- **Decreasing λ:** Fixes high bias
- **Increasing λ:** Fixes high variance.

Diagnosing Neural Networks

- A neural network with fewer parameters is **prone to underfitting**. It is also **computationally cheaper**.
- A large neural network with more parameters is **prone to overfitting**. It is also **computationally expensive**. In this case you can use regularization (increase λ) to address the overfitting.

Using a single hidden layer is a good starting default. You can train your neural network on a number of hidden layers using your cross validation set. You can then select the one that performs best.

Model Complexity Effects:

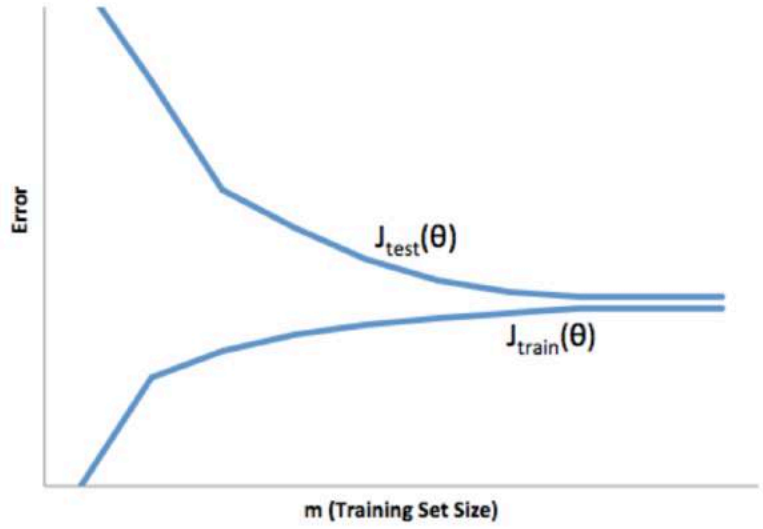
- Lower-order polynomials (low model complexity) have high bias and low variance. In this case, the model fits poorly consistently.
- Higher-order polynomials (high model complexity) fit the training data extremely well and the test data extremely poorly. These have low bias on the training data, but very high variance.
- In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

Review

Quiz

Answer: B

1. You train a learning algorithm, and find that it has unacceptably high error on the test set. You plot the learning curve, and obtain the figure below. Is the algorithm suffering from high bias, high variance, or neither?



- ( ) Neither
- ( ) High bias
- (•) High variance

2. Suppose you have implemented regularized logistic regression to classify what object is in an image (i.e., to do object recognition). However, when you test your hypothesis on a new set of images, you find that it makes unacceptably large errors with its predictions on the new images. However, your hypothesis performs **well** (has low error) on the training set. Which of the following are promising steps to take? Check all that apply.

- [ ] Try adding polynomial features.
- [ ] Use fewer training examples.
- [x] Get more training examples.
- [x] Try using a smaller set of features.

3. Suppose you have implemented regularized logistic regression to predict what items customers will purchase on a web shopping site. However, when you test your hypothesis on a new set of customers, you find that it makes unacceptably large errors in its predictions. Furthermore, the hypothesis performs **poorly** on the training set. Which of the following might be promising steps to take? Check all that apply.

- [x] Try decreasing the regularization parameter λ.
- [ ] Use fewer training examples.
- [x] Try adding polynomial features.
- [ ] Try evaluating the hypothesis on a cross validation set rather than the test set.

4. Which of the following statements are true? Check all that apply.

- [x] A typical split of a dataset into training, validation and test sets might be 60% training set, 20% validation set, and 20% test set.
- [x] Suppose you are using linear regression to predict housing prices, and your dataset comes sorted in order of increasing sizes of houses. It is then important to randomly shuffle the dataset before splitting it into training, validation and test sets, so that we don't have all the smallest houses going into the training set, and all the largest houses going into the test set.
- [ ] It is okay to use data from the test set to choose the regularization parameter λ, but not the model parameters (θ).
- [ ] Suppose you are training a logistic regression classifier using polynomial features and want to select what degree polynomial (denoted d in the lecture videos) to use. After training the classifier on the entire training set, you decide to use a subset of the training examples as a validation set. This will work just as well as having a validation set that is separate (disjoint) from the training set.

5. Which of the following statements are true? Check all that apply.

- [ ] If a neural network has much lower training error than test error, then adding more layers will help bring the test error down because we can fit the test set better.
- [x] When debugging learning algorithms, it is useful to plot a learning curve to understand if there is a high bias or high variance problem.
- [x] If a learning algorithm is suffering from high bias, only adding more training examples may **not** improve the test error significantly.
- [x] A model with more parameters is more prone to overfitting and typically has higher variance.

Machine Learning System Design

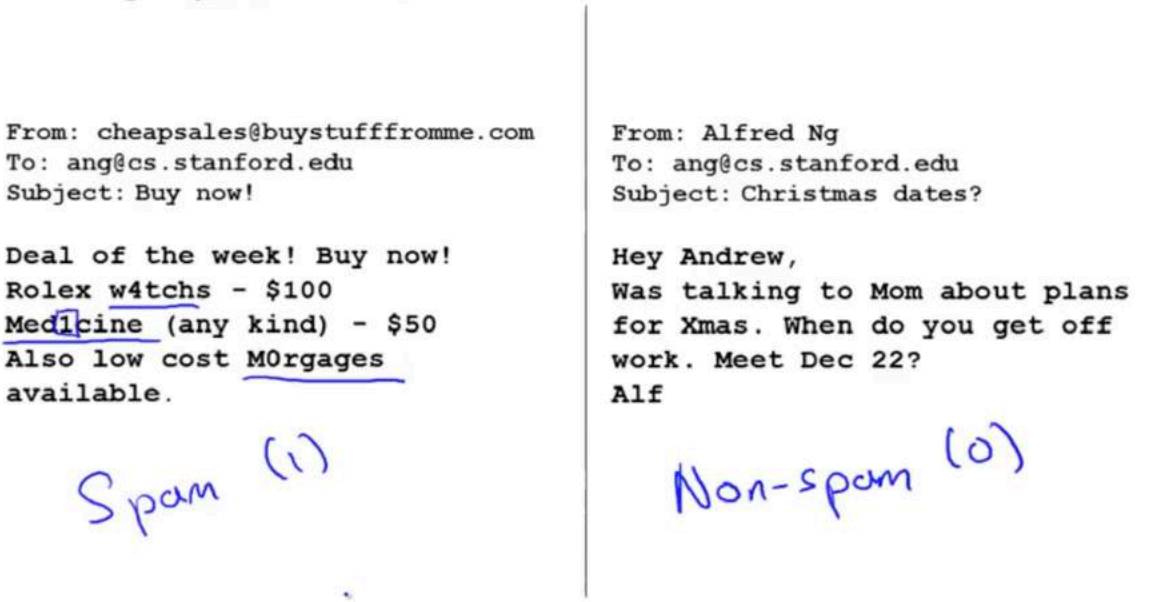
Building a Spam Classifier

Prioritizing What to Work On (Video)

In the next few videos I'd like to talk about machine learning system design. These videos will touch on the main issues that you may face when designing a complex machine learning system, and will actually try to give advice on how to strategize putting together a complex machine learning system. In case this next set of videos seems a little disjointed that's because these videos will touch on a range of the different issues that you may come across when designing complex learning systems. And even though the next set of videos may seem somewhat less mathematical, I think that this material may turn out to be very useful, and potentially huge time savers when you're building big machine learning systems. Concretely, I'd like to begin with the issue of prioritizing how to spend your time on what to work on, and I'll begin with an example on spam classification.

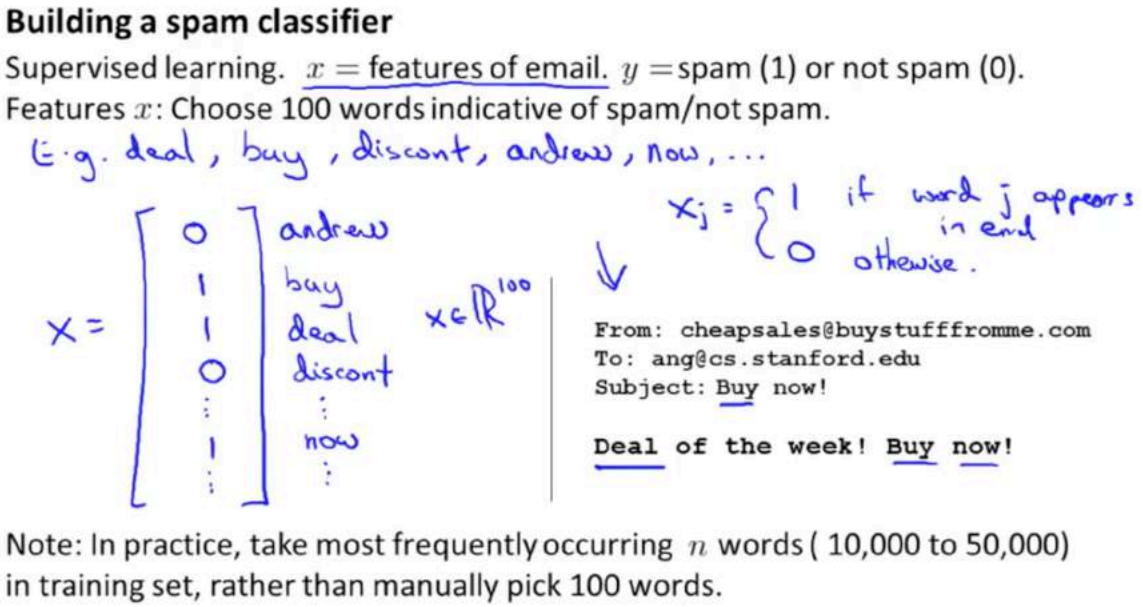
Let's say you want to build a spam classifier. Here are a couple of examples of obvious spam and non-spam emails. if the one on the left tried to sell things. And notice how spammers will deliberately misspell words, like Vincent with a 1 there, and mortgages. And on the right as maybe an obvious example of non-stamp email, actually email from my younger brother. Let's say we have a labeled training set of some number of spam emails and some non-spam emails denoted with labels y equals 1 or 0, how do we build a classifier using supervised learning to distinguish between spam and non-spam?

Building a spam classifier



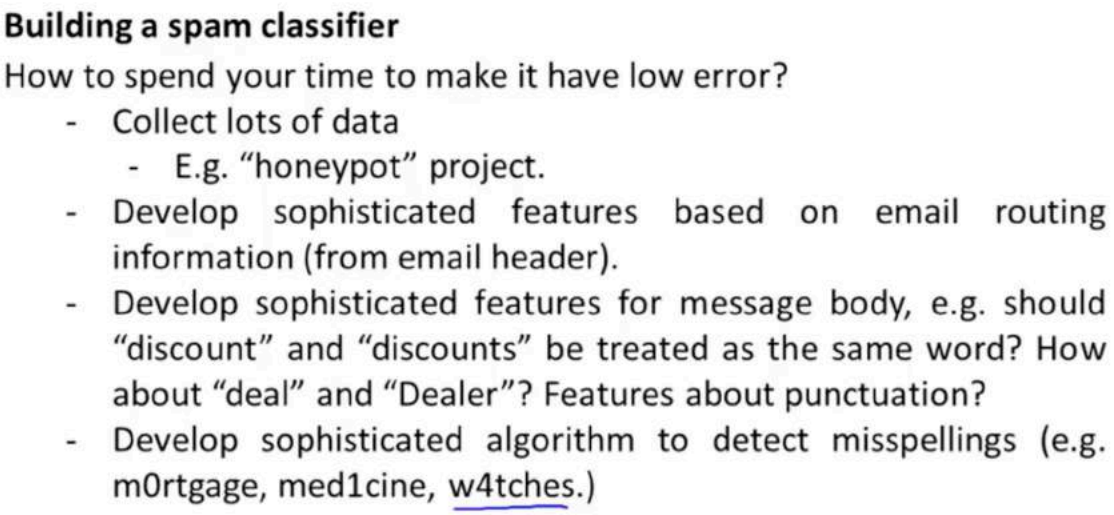
In order to apply supervised learning, the first decision we must make is how do we want to represent x, that is the features of the email. Given the features x and the labels y in our training set, we can then train a classifier, for example using logistic regression. Here's one way to choose a set of features for our emails. We could come up with, say, a list of maybe a hundred words that we think are indicative of whether e-mail is spam or non-spam, for example, if a piece of e-mail contains the word 'deal' maybe it's more likely to be spam if it contains the word 'buy' maybe more likely to be spam, a word like 'discount' is more likely to be spam, whereas if a piece of email contains my name, Andrew, maybe that means the person actually knows who I am and that might mean it's less likely to be spam. And maybe for some reason I think the word "now" may be indicative of non-spam because I get a lot of urgent emails, and so on, and maybe we choose a hundred words or so. Given a piece of email, we can then take this piece of email and encode it into a feature vector as follows. I'm going to take my list of a hundred words and sort them in alphabetical order say. It doesn't have to be sorted. But, you know, here's a, here's my list of words, just count and so on, until eventually I'll get down to now, and so on and given a piece of e-mail like that shown on the right, I'm going to check and see whether or not each of these words appears in the e-mail and then I'm going to define a feature vector x where in this piece of an email on the right, my name doesn't appear so I'm gonna put a zero there. The word "by" does appear, so I'm gonna put a one there and I'm just gonna put one's or zeroes. I'm gonna put a one even though the word "by" occurs twice. I'm not gonna recount how many times the word occurs. The word "due" appears, I put a one there. The word "discount" doesn't appear, at least not in this this little short email, and so on. The word "now" does appear and so on. So I put ones and zeroes in this

feature vector depending on whether or not a particular word appears. And in this example my feature vector would have to mention one hundred, if I have a hundred, if if I chose a hundred words to use for this representation and each of my features Xj will basically be 1 if you have a particular word that, we'll call this word j, appears in the email and Xj would be zero otherwise. Okay. So that gives me a feature representation of a piece of email. By the way, even though I've described this process as manually picking a hundred words, in practice what's most commonly done is to look through a training set, and in the training set depict the most frequently occurring n words where n is usually between ten thousand and fifty thousand, and use those as your features. So rather than manually picking a hundred words, here you look through the training examples and pick the most frequently occurring words like ten thousand to fifty thousand words, and those form the features that you are going to use to represent your email for spam classification.



Now, if you're building a spam classifier one question that you may face is, what's the best use of your time in order to make your spam classifier have higher accuracy, you have lower error. One natural inclination is going to collect lots of data. Right? And in fact there's this tendency to think that, well the more data we have the better the algorithm will do. And in fact, in the email spam domain, there are actually pretty serious projects called Honey Pot Projects, which create fake email addresses and try to get these fake email addresses into the hands of spammers and use that to try to collect tons of spam email, and therefore you know, get a lot of spam data to train learning algorithms. But we've already seen in the previous sets of videos that getting lots of data will often help, but not all the time. But for most machine learning problems, there are a lot of other things you could usually imagine doing to improve performance. For spam, one thing you might think of is to develop

more sophisticated features on the email, maybe based on the email routing information. And this would be information contained in the email header. So, when spammers send email, very often they will try to obscure the origins of the email, and maybe use fake email headers. Or send email through very unusual sets of computer service. Through very unusual routes, in order to get the spam to you. And some of this information will be reflected in the email header. And so one can imagine, looking at the email headers and trying to develop more sophisticated features to capture this sort of email routing information to identify if something is spam. Something else you might consider doing is to look at the email message body, that is the email text, and try to develop more sophisticated features. For example, should the word 'discount' and the word 'discounts' be treated as the same words or should we have treat the words 'deal' and 'dealer' as the same word? Maybe even though one is lower case and one in capitalized in this example. Or do we want more complex features about punctuation because maybe spam is using exclamation marks a lot more. I don't know. And along the same lines, maybe we also want to develop more sophisticated algorithms to detect and maybe to correct to deliberate misspellings, like mortgage, medicine, watches. Because spammers actually do this, because if you have watches with a 4 in there then well, with the simple technique that we talked about just now, the spam classifier might not equate this as the same thing as the word "watches," and so it may have a harder time realizing that something is spam with these deliberate misspellings. And this is why spammers do it.



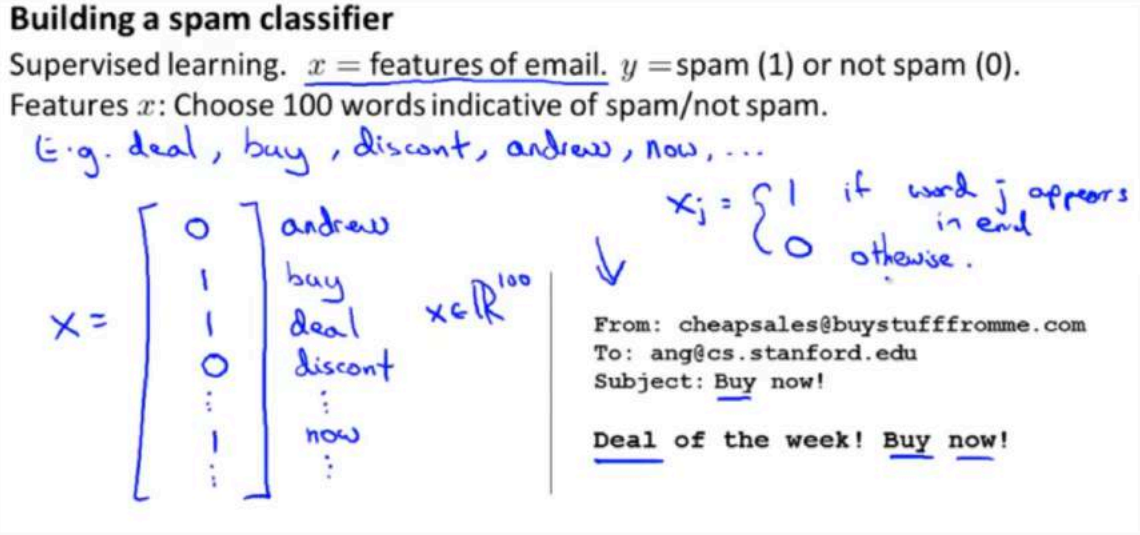
While working on a machine learning problem, very often you can brainstorm lists of different things to try, like these. By the way, I've actually worked on the spam problem myself for a while. And I actually spent quite some time on it. And even though I kind of understand the spam problem, I actually know a bit about it, I would actually have a very hard time telling you of these four options which is the best use of your time so what happens, frankly what happens far

too often is that a research group or product group will randomly fixate on one of these options. And sometimes that turns out not to be the most fruitful way to spend your time depending, you know, on which of these options someone ends up randomly fixating on. By the way, in fact, if you even get to the stage where you brainstorm a list of different options to try, you're probably already ahead of the curve. Sadly, what most people do is instead of trying to list out the options of things you might try, what far too many people do is wake up one morning and, for some reason, just, you know, have a weird gut feeling that, "Oh let's have a huge honeypot project to go and collect tons more data" and for whatever strange reason just sort of wake up one morning and randomly fixate on one thing and just work on that for six months. But I think we can do better. And in particular what I'd like to do in the next video is tell you about the concept of error analysis and talk about the way where you can try to have a more systematic way to choose amongst the options of the many different things you might work, and therefore be more likely to select what is actually a good way to spend your time, you know for the next few weeks, or next few days or the next few months.

Prioritizing What to Work On (Transcript)

System Design Example:

Given a data set of emails, we could construct a vector for each email. Each entry in this vector represents a word. The vector normally contains 10,000 to 50,000 entries gathered by finding the most frequently used words in our data set. If a word is to be found in the email, we would assign its respective entry a 1, else if it is not found, that entry would be a 0. Once we have all our x vectors ready, we train our algorithm and finally, we could use it to classify if an email is a spam or not.



So how could you spend your time to improve the accuracy of this classifier?

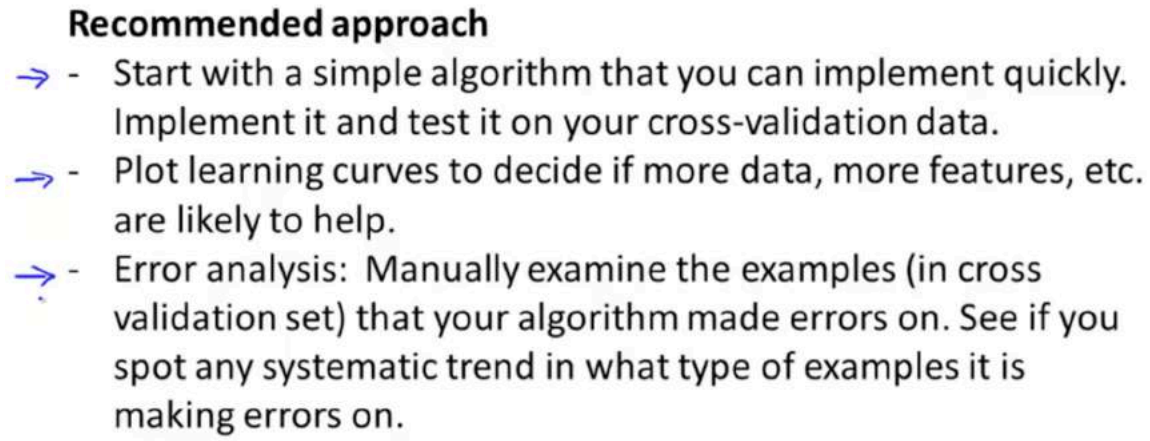
- Collect lots of data (for example "honeypot" project but doesn't always work)
- Develop sophisticated features (for example: using email header data in spam emails)
- Develop algorithms to process your input in different ways (recognizing misspellings in spam).

It is difficult to tell which of the options will be most helpful.

Error Analysis (Video)

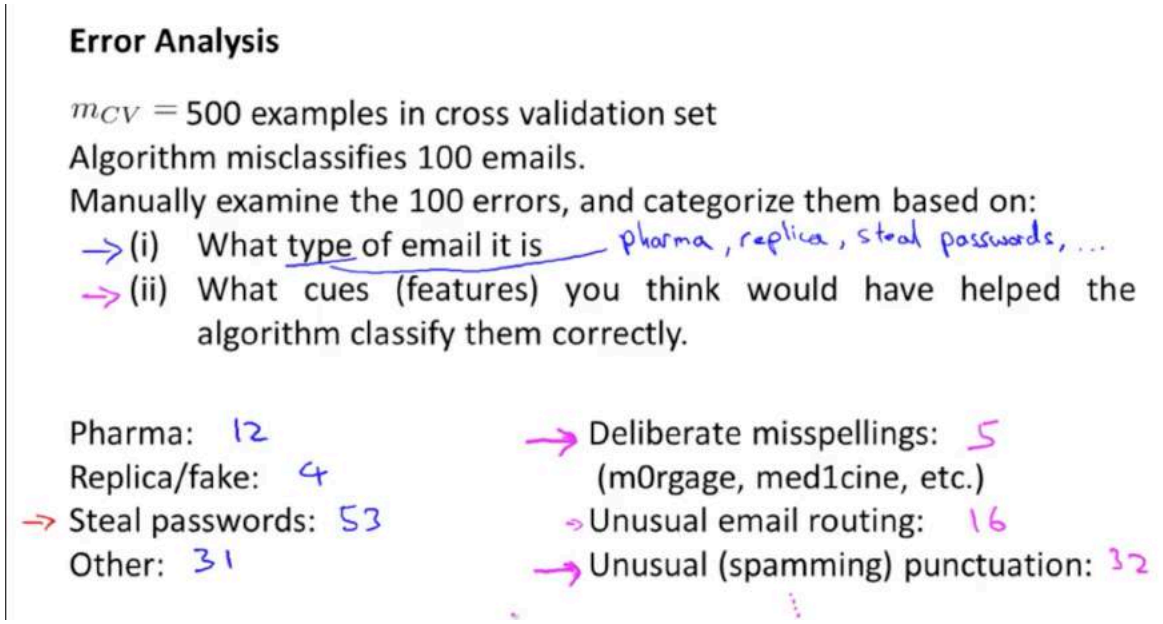
In the last video I talked about how, when faced with a machine learning problem, there are often lots of different ideas for how to improve the algorithm. In this video, let's talk about the concept of error analysis. Which will hopefully give you a way to more systematically make some of these decisions. If you're starting work on a machine learning problem, or building a machine learning application. It's often considered very good practice to start, not by building a very complicated system with lots of complex features and so on. But to instead start by building a very simple algorithm that you can implement quickly. And when I start with a learning problem what I usually do is spend at most one day, like literally at most 24 hours, To try to get

something really quick and dirty. Frankly not at all sophisticated system but get something really quick and dirty running, and implement it and then test it on my cross-validation data. Once you've done that you can then plot learning curves, this is what we talked about in the previous set of videos. But plot learning curves of the training and test errors to try to figure out if you're learning algorithm maybe suffering from high bias or high variance, or something else. And use that to try to decide if having more data, more features, and so on are likely to help. And the reason that this is a good approach is often, when you're just starting out on a learning problem, there's really no way to tell in advance. Whether you need more complex features, or whether you need more data, or something else. And it's just very hard to tell in advance, that is, in the absence of evidence, in the absence of seeing a learning curve. It's just incredibly difficult to figure out where you should be spending your time. And it's often by implementing even a very, very quick and dirty implementation. And by plotting learning curves, that helps you make these decisions. So if you like you can to think of this as a way of avoiding whats sometimes called premature optimization in computer programming. And this idea that says we should let evidence guide our decisions on where to spend our time rather than use gut feeling, which is often wrong. In addition to plotting learning curves, one other thing that's often very useful to do is what's called error analysis. And what I mean by that is that when building say a spam classifier. I will often look at my cross validation set and manually look at the emails that my algorithm is making errors on. So look at the spam e-mails and non-spam e-mails that the algorithm is misclassifying and see if you can spot any systematic patterns in what type of examples it is misclassifying. And often, by doing that, this is the process that will inspire you to design new features. Or they'll tell you what are the current things or current shortcomings of the system. And give you the inspiration you need to come up with improvements to it.



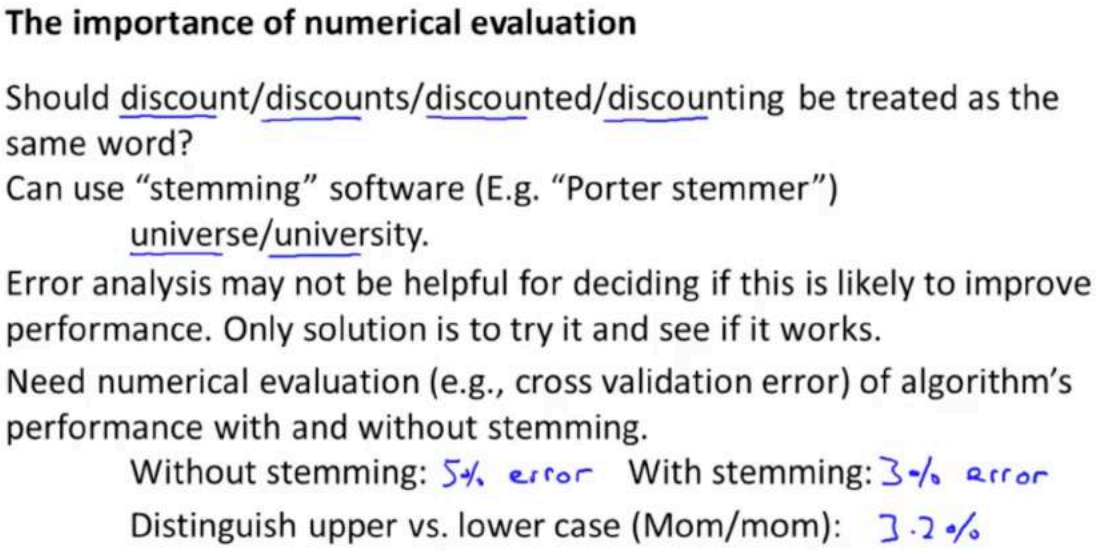
Concretely, here's a specific example. Let's say you've built a spam classifier

and you have 500 examples in your cross validation set. And let's say in this example that the algorithm has a very high error rate. And this classifies 100 of these cross validation examples. So what I do is manually examine these 100 errors and manually categorize them. Based on things like what type of email it is, what cues or what features you think might have helped the algorithm classify them correctly. So, specifically, by what type of email it is, if I look through these 100 errors, I might find that maybe the most common types of spam emails in these classifies are maybe emails on pharma or pharmacies, trying to sell drugs. Maybe emails that are trying to sell replicas such as fake watches, fake random things, maybe some emails trying to steal passwords,. These are also called phishing emails, that's another big category of emails, and maybe other categories. So in terms of classify what type of email it is, I would actually go through and count up my hundred emails. Maybe I find that 12 of them is label emails, or pharma emails, and maybe 4 of them are emails trying to sell replicas, that sell fake watches or something. And maybe I find that 53 of them are these what's called phishing emails, basically emails trying to persuade you to give them your password. And 31 emails are other types of emails. And it's by counting up the number of emails in these different categories that you might discover, for example. That the algorithm is doing really, particularly poorly on emails trying to steal passwords. And that may suggest that it might be worth your effort to look more carefully at that type of email and see if you can come up with better features to categorize them correctly. And, also what I might do is look at what cues or what additional features might have helped the algorithm classify the emails. So let's say that some of our hypotheses about things or features that might help us classify emails better are. Trying to detect deliberate misspellings versus unusual email routing versus unusual spamming punctuation. Such as if people use a lot of exclamation marks. And once again I would manually go through and let's say I find five cases of this and 16 of this and 32 of this and a bunch of other types of emails as well. And if this is what you get on your cross validation set, then it really tells you that maybe deliberate spellings is a sufficiently rare phenomenon that maybe it's not worth all the time trying to write algorithms that detect that. But if you find that a lot of spammers are using, you know, unusual punctuation, then maybe that's a strong sign that it might actually be worth your while to spend the time to develop more sophisticated features based on the punctuation. So this sort of error analysis, which is really the process of manually examining the mistakes that the algorithm makes, can often help guide you to the most fruitful avenues to pursue. And this also explains why I often recommend implementing a quick and dirty implementation of an algorithm. What we really want to do is figure out what are the most difficult examples for an algorithm to classify. And very often for different algorithms, for different learning algorithms they'll often find similar categories of examples difficult. And by having a quick and dirty implementation, that's often a quick way to let you identify some errors and quickly identify what are the hard examples. So that you can focus your effort on those.



Lastly, when developing learning algorithms, one other useful tip is to make sure that you have a numerical evaluation of your learning algorithm. And what I mean by that is you if you're developing a learning algorithm, it's often incredibly helpful. If you have a way of evaluating your learning algorithm that just gives you back a single real number, maybe accuracy, maybe error. But the single real number that tells you how well your learning algorithm is doing. I'll talk more about this specific concept in later videos, but here's a specific example. Let's say we're trying to decide whether or not we should treat words like discount, discounts, discounted, discounting as the same word. So you know maybe one way to do that is to just look at the first few characters in the word like, you know. If you just look at the first few characters of a word, then you figure out that maybe all of these words roughly have similar meanings. In natural language processing, the way that this is done is actually using a type of software called stemming software. And if you ever want to do this yourself, search on a web-search engine for the porter stemmer, and that would be one reasonable piece of software for doing this sort of stemming, which will let you treat all these words, discount, discounts, and so on, as the same word. But using a stemming software that basically looks at the first few alphabets of a word, more of less, it can help, but it can hurt. And it can hurt because for example, the software may mistake the words universe and university as being the same thing. Because, you know, these two words start off with the same alphabets. So if you're trying to decide whether or not to use stemming software for a spam cross classifier, it's not always easy to tell. And in particular, error analysis may not actually be helpful for deciding if this sort of stemming idea is a good idea. Instead, the best way to figure out if using stemming software is good to help your classifier is if you have a way to very quickly just try it and see if it works. And in order to do this, having a way to numerically evaluate your algorithm is going to be very helpful. Concretely,

maybe the most natural thing to do is to look at the cross validation error of the algorithm's performance with and without stemming. So, if you run your algorithm without stemming and end up with 5 percent classification error. And you rerun it and you end up with 3 percent classification error, then this decrease in error very quickly allows you to decide that it looks like using stemming is a good idea. For this particular problem, there's a very natural, single, real number evaluation metric, namely the cross validation error. We'll see later examples where coming up with this sort of single, real number evaluation metric will need a little bit more work. But as we'll see in a later video, doing so would also then let you make these decisions much more quickly of say, whether or not to use stemming. And, just as one more quick example, let's say that you're also trying to decide whether or not to distinguish between upper versus lower case. So, you know, as the word, mom, were upper case, and versus lower case m, should that be treated as the same word or as different words? Should this be treated as the same feature, or as different features? And so, once again, because we have a way to evaluate our algorithm. If you try this down here, if I stopped distinguishing upper and lower case, maybe I end up with 3.2 percent error. And I find that therefore, this does worse than if I use only stemming. So, this let's me very quickly decide to go ahead and to distinguish or to not distinguish between upper and lowercase. So when you're developing a learning algorithm, very often you'll be trying out lots of new ideas and lots of new versions of your learning algorithm. If every time you try out a new idea, if you end up manually examining a bunch of examples again to see if it got better or worse, that's gonna make it really hard to make decisions on. Do you use stemming or not? Do you distinguish upper and lower case or not? But by having a single real number evaluation metric, you can then just look and see, oh, did the arrow go up or did it go down? And you can use that to much more rapidly try out new ideas and almost right away tell if your new idea has improved or worsened the performance of the learning algorithm. And this will let you often make much faster progress. So the recommended, strongly recommended the way to do error analysis is on the cross validations there rather than the test set. But, you know, there are people that will do this on the test set, even though that's definitely a less mathematic appropriate, certainly a less recommended way to, thing to do than to do error analysis on your cross validation set.



Question

Why is the recommended approach to perform error analysis using the cross validation data used to compute $J_{\text{CV}}(\theta)$ rather than the test data used to compute $J_{\text{test}}(\theta)$?

- The cross validation data set is usually large.
- This process will give a lower error on the test set.
- If we develop new features by examining the test set, then we may end up choosing features that work well specifically for the test set, so Jtest(θ) is no longer a good estimate of how well we generalize to new examples.

  Correct
- Doing so is less likely to lead to choosing an excessive number of features.

So, to wrap up this video, when starting on a new machine learning problem, what I almost always recommend is to implement a quick and dirty implementation of your learning out of them. And I've almost never seen anyone spend too little time on this quick and dirty implementation. I've pretty much only ever seen people spend much too much time building their first, supposedly, quick and dirty implementation. So really, don't worry about it being too quick, or don't worry about it being too dirty. But really, implement something as quickly as you can. And 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 Analysis (Transcript)

The recommended approach to solving machine learning problems is to:

- Start with a simple algorithm, implement it quickly, and test it early on your cross validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Manually examine the errors on examples in the cross validation set and try to spot a trend where most of the errors were made.

For example, assume that we have 500 emails and our algorithm misclassifies a 100 of them. We could manually analyze the 100 emails and categorize them based on what type of emails they are. We could then try to come up with new cues and features that would help us classify these 100 emails correctly. Hence, if most of our misclassified emails are those which try to steal passwords, then we could find some features that are particular to those emails and add them to our model. We could also see how classifying each word according to its root changes our error rate:

The importance of numerical evaluation

Should discount/discounts/discounted/discounting be treated as the same word?
Can use "stemming" software (E.g. "Porter stemmer")
universe/university.
Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.
Need numerical evaluation (e.g., cross validation error) of algorithm's performance with and without stemming.
Without stemming: 5% error With stemming: 3% error
Distinguish upper vs. lower case (Mom/mom): 3.2%

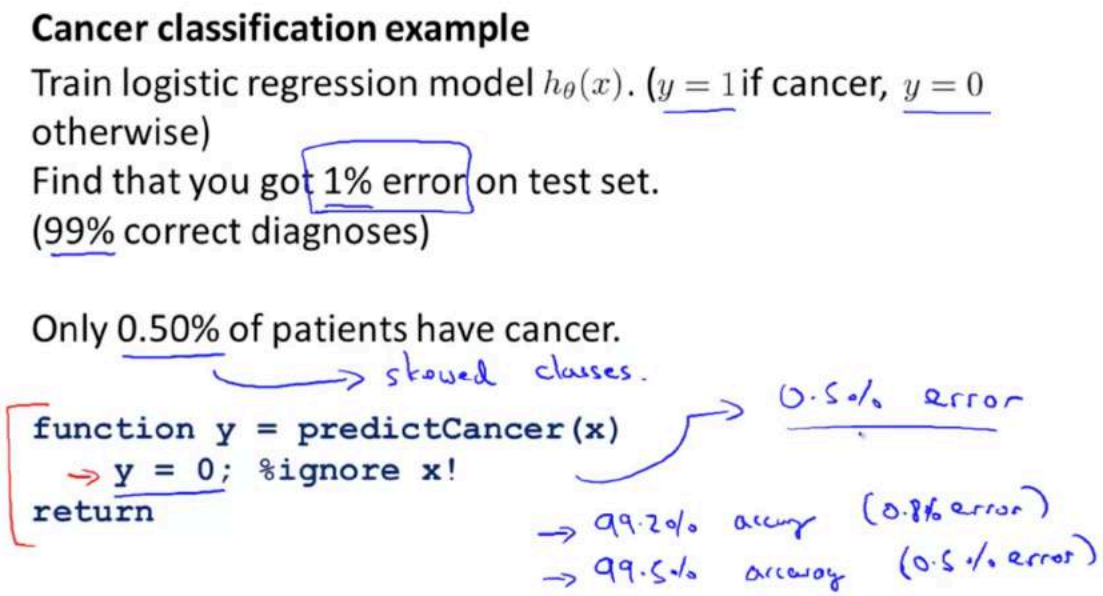
It is very important to get error results as a single, numerical value. Otherwise it is difficult to assess your algorithm's performance. For example if we use stemming, which is the process of treating the same word with different forms (fail/failing/failed) as one word (fail), and get a 3% error rate instead of 5%, then we should definitely add it to our model. However, if we try to distinguish between upper case and lower case letters and end up getting a 3.2% error rate instead of 3%, then we should avoid using this new feature. Hence, we should try new things, get a numerical value for our error rate, and based on our result decide whether we want to keep the new feature or not.

Handling Skewed Data

Error Metrics for Skewed Classes

In the previous video, I talked about error analysis and the importance of having error metrics, that is of having a single real number evaluation metric for your learning algorithm to tell how well it's doing. In the context of evaluation and of error metrics, there is one important case, where it's particularly tricky to come up with an appropriate error metric, or evaluation metric, for your learning algorithm. That case is the case of what's called skewed classes. Let me tell you what that means. Consider the problem of cancer classification, where we have features of medical patients and we want to decide whether or not they have cancer. So this is like the malignant versus benign tumor classification example that we had earlier. So let's say y equals 1 if the patient has cancer and y equals 0 if they do not. We have trained the progression classifier and let's say we test our classifier on a test set and find that we get 1 percent error. So, we're making 99% correct diagnosis. Seems like a really impressive result, right. We're correct 99% percent of the time. But now, let's say we find out that only 0.5 percent of patients in our training test sets actually have cancer. So only half a percent of the patients that come through our screening process have cancer. In this case, the 1% error no longer looks so impressive. And in particular, here's a piece of code, here's actually a piece of non learning code that takes this input of features x and it ignores it. It just sets y equals 0 and always predicts, you know, nobody has cancer and this algorithm would actually get 0.5 percent error. So this is even better than the 1% error that we were getting just now and this is a non learning algorithm

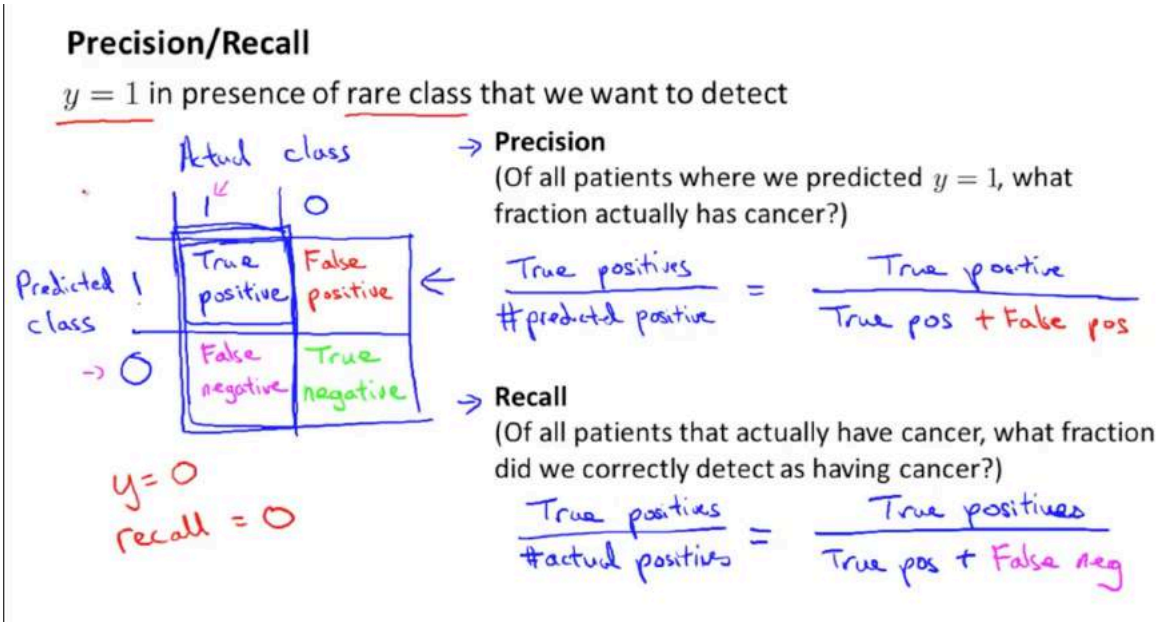
that you know, it is just predicting y equals 0 all the time. So this setting of when the ratio of positive to negative examples is very close to one of two extremes, where, in this case, the number of positive examples is much, much smaller than the number of negative examples because y equals one so rarely, this is what we call the case of skewed classes. We just have a lot more of examples from one class than from the other class. And by just predicting y equals 0 all the time, or maybe our predicting y equals 1 all the time, an algorithm can do pretty well. So the problem with using classification error or classification accuracy as our evaluation metric is the following. Let's say you have one joining algorithm that's getting 99.2% accuracy. So, that's a 0.8% error. Let's say you make a change to your algorithm and you now are getting 99.5% accuracy. That is 0.5% error. So, is this an improvement to the algorithm or not? One of the nice things about having a single real number evaluation metric is this helps us to quickly decide if we just need a good change to the algorithm or not. By going from 99.2% accuracy to 99.5% accuracy. You know, did we just do something useful or did we just replace our code with something that just predicts y equals zero more often? So, if you have very skewed classes it becomes much harder to use just classification accuracy, because you can get very high classification accuracies or very low errors, and it's not always clear if doing so is really improving the quality of your classifier because predicting y equals 0 all the time doesn't seem like a particularly good classifier. But just predicting y equals 0 more often can bring your error down to, you know, maybe as low as 0.5%.



When we're faced with such a skewed classes therefore we would want to come up with a different error metric or a different evaluation metric. One such evaluation metric are what's called precision recall. Let me explain what that

is. Let's say we are evaluating a classifier on the test set. For the examples in the test set the actual class of that example in the test set is going to be either one or zero, right, if there is a binary classification problem. And what our learning algorithm will do is it will, you know, predict some value for the class and our learning algorithm will predict the value for each example in my test set and the predicted value will also be either one or zero. So let me draw a two by two table as follows, depending on a full of these entries depending on what was the actual class and what was the predicted class. If we have an example where the actual class is one and the predicted class is one then that's called an example that's a true positive, meaning our algorithm predicted that it's positive and in reality the example is positive. If our learning algorithm predicted that something is negative, class zero, and the actual class is also class zero then that's what's called a true negative. We predicted zero and it actually is zero. To find the other two boxes, if our learning algorithm predicts that the class is one but the actual class is zero, then that's called a false positive. So that means our algorithm for the patient is cancelled out in reality if the patient does not. And finally, the last box is a zero, one. That's called a false negative because our algorithm predicted zero, but the actual class was one. And so, we have this little sort of two by two table based on what was the actual class and what was the predicted class. So here's a different way of evaluating the performance of our algorithm. We're going to compute two numbers. The first is called precision - and what that says is, of all the patients where we've predicted that they have cancer, what fraction of them actually have cancer? So let me write this down, the precision of a classifier is the number of true positives divided by the number that we predicted as positive, right? So of all the patients that we went to those patients and we told them, "We think you have cancer." Of all those patients, what fraction of them actually have cancer? So that's called precision. And another way to write this would be true positives and then in the denominator is the number of predicted positives, and so that would be the sum of the, you know, entries in this first row of the table. So it would be true positives divided by true positives. I'm going to abbreviate positive as POS and then plus false positives, again abbreviating positive using POS. So that's called precision, and as you can tell high precision would be good. That means that all the patients that we went to and we said, "You know, we're very sorry. We think you have cancer," high precision means that of that group of patients most of them we had actually made accurate predictions on them and they do have cancer. The second number we're going to compute is called recall, and what recall say is, if all the patients in, let's say, in the test set or the cross-validation set, but if all the patients in the data set that actually have cancer, what fraction of them that we correctly detect as having cancer. So if all the patients have cancer, how many of them did we actually go to them and you know, correctly told them that we think they need treatment. So, writing this down, recall is defined as the number of positives, the number of true positives, meaning the number of people that have cancer and that we correctly predicted have cancer and we take that and divide that by, divide that by the number of actual

positives, so this is the right number of actual positives of all the people that do have cancer. What fraction do we directly flag and you know, send the treatment. So, to rewrite this in a different form, the denominator would be the number of actual positives as you know, is the sum of the entries in this first column over here. And so writing things out differently, this is therefore, the number of true positives, divided by the number of true positives plus the number of false negatives. And so once again, having a high recall would be a good thing. So by computing precision and recall this will usually give us a better sense of how well our classifier is doing. And in particular if we have a learning algorithm that predicts y equals zero all the time, if it predicts no one has cancer, then this classifier will have a recall equal to zero, because there won't be any true positives and so that's a quick way for us to recognize that, you know, a classifier that predicts y equals 0 all the time, just isn't a very good classifier. And more generally, even for settings where we have very skewed classes, it's not possible for an algorithm to sort of "cheat" and somehow get a very high precision and a very high recall by doing some simple thing like predicting y equals 0 all the time or predicting y equals 1 all the time. And so we're much more sure that a classifier of a high precision or high recall actually is a good classifier, and this gives us a more useful evaluation metric that is a more direct way to actually understand whether, you know, our algorithm may be doing well. So one final note in the definition of precision and recall, that we would define precision and recall, usually we use the convention that y is equal to 1, in the presence of the more rare class. So if we are trying to detect. rare conditions such as cancer, hopefully that's a rare condition, precision and recall are defined setting y equals 1, rather than y equals 0, to be sort of that the presence of that rare class that we're trying to detect. And by using precision and recall, we find, what happens is that even if we have very skewed classes, it's not possible for an algorithm to you know, "cheat" and predict y equals 1 all the time, or predict y equals 0 all the time, and get high precision and recall. And in particular, if a classifier is getting high precision and high recall, then we are actually confident that the algorithm has to be doing well, even if we have very skewed classes.



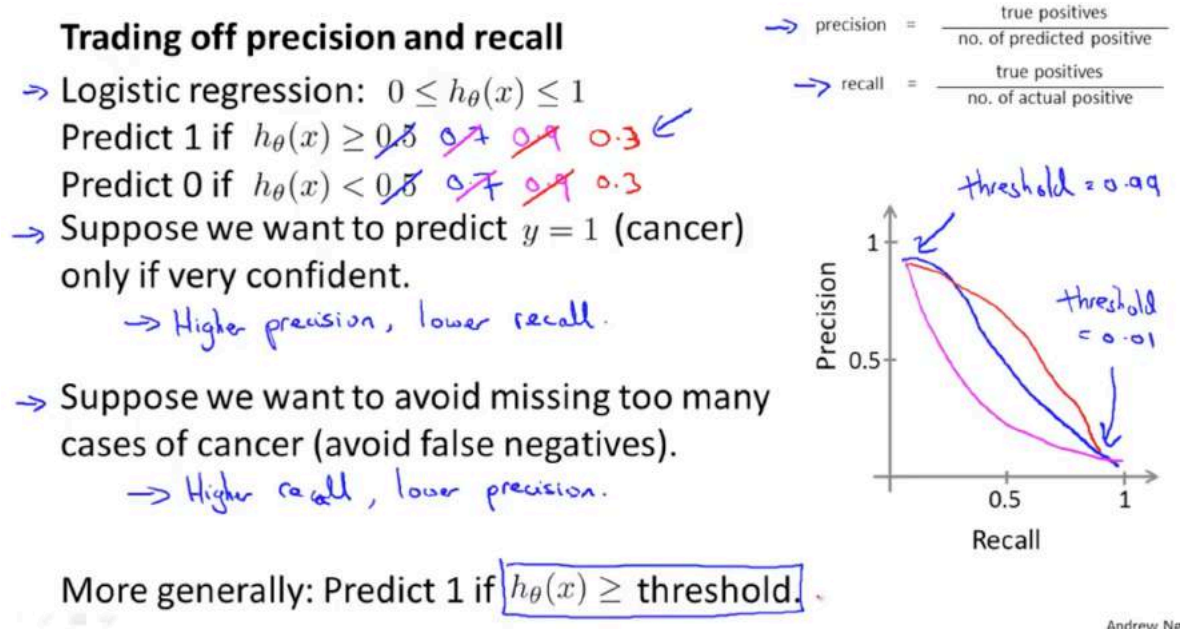
So for the problem of skewed classes precision recall gives us more direct insight into how the learning algorithm is doing and this is often a much better way to evaluate our learning algorithms, than looking at classification error or classification accuracy, when the classes are very skewed.

Trading Off Precision and Recall

In the last video, we talked about precision and recall as an evaluation metric for calcification problems with skewed constants. For many applications, we'll want to somehow control the trade-off between precision and recall. Let me tell you how to do that and also show you some even more effective ways to use precision and recall as an evaluation metric for learning algorithms. As a reminder, here are the definitions of precision and recall from the previous video. Let's continue our cancer classification example, where y equals 1 if the patient has cancer and y equals 0 otherwise. And let's say we're trained in logistic regression classifier which outputs probability between 0 and 1. So, as usual, we're going to predict 1, y equals 1, if h(x) is greater or equal to 0.5. And predict 0 if the hypothesis outputs a value less than 0.5. And this classifier may give us some value for precision and some value for recall. But now, suppose we want to predict that the patient has cancer only if we're very confident that they really do. Because if you go to a patient and you tell them

that they have cancer, it's going to give them a huge shock. What we give is a seriously bad news, and they may end up going through a pretty painful treatment process and so on. And so maybe we want to tell someone that we think they have cancer only if they are very confident. One way to do this would be to modify the algorithm, so that instead of setting this threshold at 0.5, we might instead say that we will predict that y is equal to 1 only if h(x) is greater or equal to 0.7. So this is like saying, we'll tell someone they have cancer only if we think there's a greater than or equal to, 70% chance that they have cancer. And, if you do this, then you're predicting someone has cancer only when you're more confident and so you end up with a classifier that has higher precision. Because all of the patients that you're going to and saying, we think you have cancer, although those patients are now ones that you're pretty confident actually have cancer. And so a higher fraction of the patients that you predict have cancer will actually turn out to have cancer because making those predictions only if we're pretty confident. But in contrast this classifier will have lower recall because now we're going to make predictions, we're going to predict y = 1 on a smaller number of patients. Now, can even take this further. Instead of setting the threshold at 0.7, we can set this at 0.9. Now we'll predict y=1 only if we are more than 90% certain that the patient has cancer. And so, a large fraction of those patients will turn out to have cancer. And so this would be a higher precision classifier will have lower recall because we want to correctly detect that those patients have cancer. Now consider a different example. Suppose we want to avoid missing too many actual cases of cancer, so we want to avoid false negatives. In particular, if a patient actually has cancer, but we fail to tell them that they have cancer then that can be really bad. Because if we tell a patient that they don't have cancer, then they're not going to go for treatment. And if it turns out that they have cancer, but we fail to tell them they have cancer, well, they may not get treated at all. And so that would be a really bad outcome because they die because we told them that they don't have cancer. They fail to get treated, but it turns out they actually have cancer. So, suppose that, when in doubt, we want to predict that y=1. So, when in doubt, we want to predict that they have cancer so that at least they look further into it, and these can get treated in case they do turn out to have cancer. In this case, rather than setting higher probability threshold, we might instead take this value and instead set it to a lower value. So maybe 0.3 like so, right? And by doing so, we're saying that, you know what, if we think there's more than a 30% chance that they have cancer we better be more conservative and tell them that they may have cancer so that they can seek treatment if necessary. And in this case what we would have is going to be a higher recall classifier, because we're going to be correctly flagging a higher fraction of all of the patients that actually do have cancer. But we're going to end up with lower precision because a higher fraction of the patients that we said have cancer, a high fraction of them will turn out not to have cancer after all. And by the way, just as a sider, when I talk about this to other students, I've been told before, it's pretty amazing, some of my students say, is how I can tell the story both ways. Why we might want to have higher precision

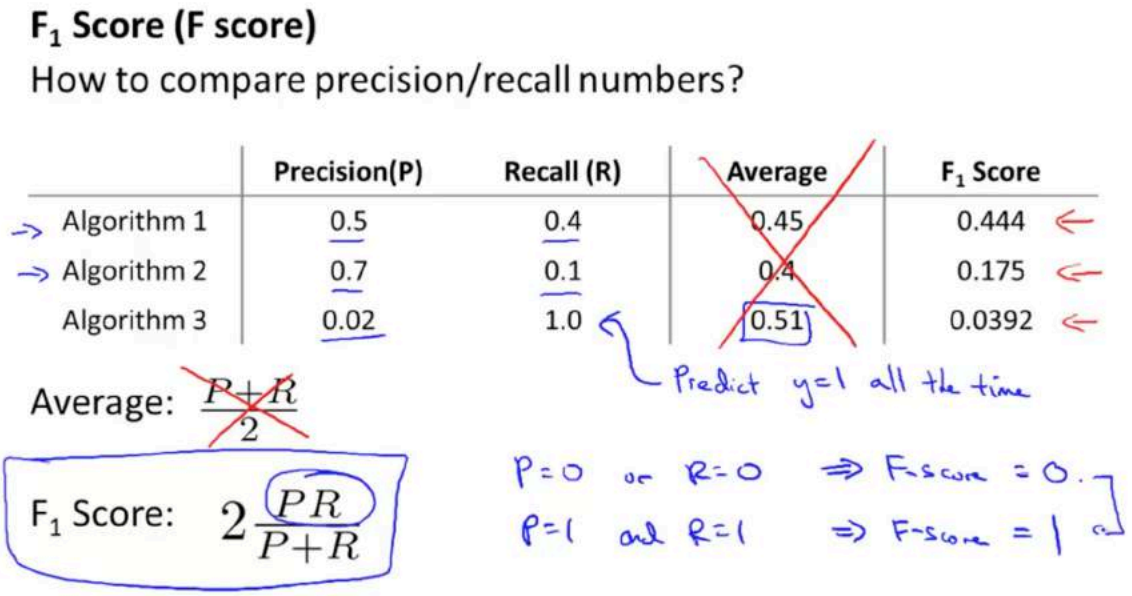
or higher recall and the story actually seems to work both ways. But I hope the details of the algorithm is true and the more general principle is depending on where you want, whether you want higher precision- lower recall, or higher recall- lower precision. You can end up predicting y=1 when h(x) is greater than some threshold. And so in general, for most classifiers there is going to be a trade off between precision and recall, and as you vary the value of this threshold that we join here, you can actually plot out some curve that trades off precision and recall. Where a value up here, this would correspond to a very high value of the threshold, maybe threshold equals 0.99. So that's saying, predict y=1 only if we're more than 99% confident, at least 99% probability this one. So that would be a high precision, relatively low recall. Where as the point down here, will correspond to a value of the threshold that's much lower, maybe equal 0.01, meaning, when in doubt at all, predict y=1, and if you do that, you end up with a much lower precision, higher recall classifier. And as you vary the threshold, if you want you can actually trace of a curve for your classifier to see the range of different values you can get for precision recall. And by the way, the precision-recall curve can look like many different shapes. Sometimes it will look like this, sometimes it will look like that. Now there are many different possible shapes for the precision-recall curve, depending on the details of the classifier.



So, this raises another interesting question which is, is there a way to choose this threshold automatically? Or more generally, if we have a few different algorithms or a few different ideas for algorithms, how do we compare different precision recall numbers? Concretely, suppose we have three different learning algorithms. So actually, maybe these are three different learning algorithms, maybe these are the same algorithm but just with different values for the threshold. How do we decide which of these algorithms is best? One of

the things we talked about earlier is the importance of a single real number evaluation metric. And that is the idea of having a number that just tells you how well is your classifier doing. But by switching to the precision recall metric we've actually lost that. We now have two real numbers. And so we often, we end up face the situations like if we trying to compare Algorithm 1 and Algorithm 2, we end up asking ourselves, is the precision of 0.5 and a recall of 0.4, was that better or worse than a precision of 0.7 and recall of 0.1? And, if every time you try out a new algorithm you end up having to sit around and think, well, maybe 0.5/0.4 is better than 0.7/0.1, or maybe not, I don't know. If you end up having to sit around and think and make these decisions, that really slows down your decision making process for what changes are useful to incorporate into your algorithm. Whereas in contrast, if we have a single real number evaluation metric like a number that just tells us is algorithm 1 or is algorithm 2 better, then that helps us to much more quickly decide which algorithm to go with. It helps us as well to much more quickly evaluate different changes that we may be contemplating for an algorithm. So how can we get a single real number evaluation metric? One natural thing that you might try is to look at the average precision and recall. So, using P and R to denote precision and recall, what you could do is just compute the average and look at what classifier has the highest average value. But this turns out not to be such a good solution, because similar to the example we had earlier it turns out that if we have a classifier that predicts y=1 all the time, then if you do that you can get a very high recall, but you end up with a very low value of precision. Conversely, if you have a classifier that predicts y equals zero, almost all the time, that is that it predicts y=1 very sparingly, this corresponds to setting a very high threshold using the notation of the previous y. Then you can actually end up with a very high precision with a very low recall. So, the two extremes of either a very high threshold or a very low threshold, neither of that will give a particularly good classifier. And the way we recognize that is by seeing that we end up with a very low precision or a very low recall. And if you just take the average of (P+R)/2 from this example, the average is actually highest for Algorithm 3, even though you can get that sort of performance by predicting y=1 all the time and that's just not a very good classifier, right? You predict y=1 all the time, just normal useful classifier, but all it does is prints out y=1. And so Algorithm 1 or Algorithm 2 would be more useful than Algorithm 3. But in this example, Algorithm 3 has a higher average value of precision recall than Algorithms 1 and 2. So we usually think of this average of precision and recall as not a particularly good way to evaluate our learning algorithm. In contrast, there's a different way for combining precision and recall. This is called the F Score and it uses that formula. And so in this example, here are the F Scores. And so we would tell from these F Scores, it looks like Algorithm 1 has the highest F Score, Algorithm 2 has the second highest, and Algorithm 3 has the lowest. And so, if we go by the F Score we would pick probably Algorithm 1 over the others. The F Score, which is also called the F1 Score, is usually written F1 Score that I have here, but often people will just say F Score, either term is used. Is a little bit like taking the average of precision and recall, but it

gives the lower value of precision and recall, whichever it is, it gives it a higher weight. And so, you see in the numerator here that the F Score takes a product of precision and recall. And so if either precision is 0 or recall is equal to 0, the F Score will be equal to 0. So in that sense, it kind of combines precision and recall, but for the F Score to be large, both precision and recall have to be pretty large. I should say that there are many different possible formulas for combing precision and recall. This F Score formula is really maybe a, just one out of a much larger number of possibilities, but historically or traditionally this is what people in Machine Learning seem to use. And the term F Score, it doesn't really mean anything, so don't worry about why it's called F Score or F1 Score. But this usually gives you the effect that you want because if either a precision is zero or recall is zero, this gives you a very low F Score, and so to have a high F Score, you kind of need a precision or recall to be one. And concretely, if P=0 or R=0, then this gives you that the F Score = 0. Whereas a perfect F Score, so if precision equals one and recall equals 1, that will give you an F Score, that's equal to 1 times 1 over 2 times 2, so the F Score will be equal to 1, if you have perfect precision and perfect recall. And intermediate values between 0 and 1, this usually gives a reasonable rank ordering of different classifiers.



Question

You have trained a logistic regression classifier and plan to make predictions according to:

- Predict $y = 1$ if $h_\theta(x) \geq \text{threshold}$
- Predict $y = 0$ if $h_\theta(x) < \text{threshold}$

For different values of the threshold parameter, you get different values of precision (P) and recall (R). Which of of the following would be a reasonable way to pick the value to use for the threshold?

- ○ Measure precision (P) and recall (R) on the **test set** and choose the value of threshold which maximizes $\frac{P+R}{2}$
- ○ Measure precision (P) and recall (R) on the **test set** and choose the value of threshold which maximizes $2\frac{PR}{P+R}$
- ○ Measure precision (P) and recall (R) on the **cross validation set** and choose the value of threshold which maximizes $\frac{P+R}{2}$
- ● Measure precision (P) and recall (R) on the **cross validation set** and choose the value of threshold which maximizes $2\frac{PR}{P+R}$

So in this video, we talked about the notion of trading off between precision and recall, and how we can vary the threshold that we use to decide whether to predict y=1 or y=0. So it's the threshold that says, do we need to be at least 70% confident or 90% confident, or whatever before we predict y=1. And by varying the threshold, you can control a trade off between precision and recall. We also talked about the F Score, which takes precision and recall, and again, gives you a single real number evaluation metric. And of course, if your goal is to automatically set that threshold to decide what's really y=1 and y=0, one pretty reasonable way to do that would also be to try a range of different values of thresholds. So you try a range of values of thresholds and evaluate these different thresholds on, say, your cross-validation set and then to pick whatever value of threshold gives you the highest F Score on your cross validation set [INAUDIBLE]. And that be a pretty reasonable way to automatically choose the threshold for your classifier as well.

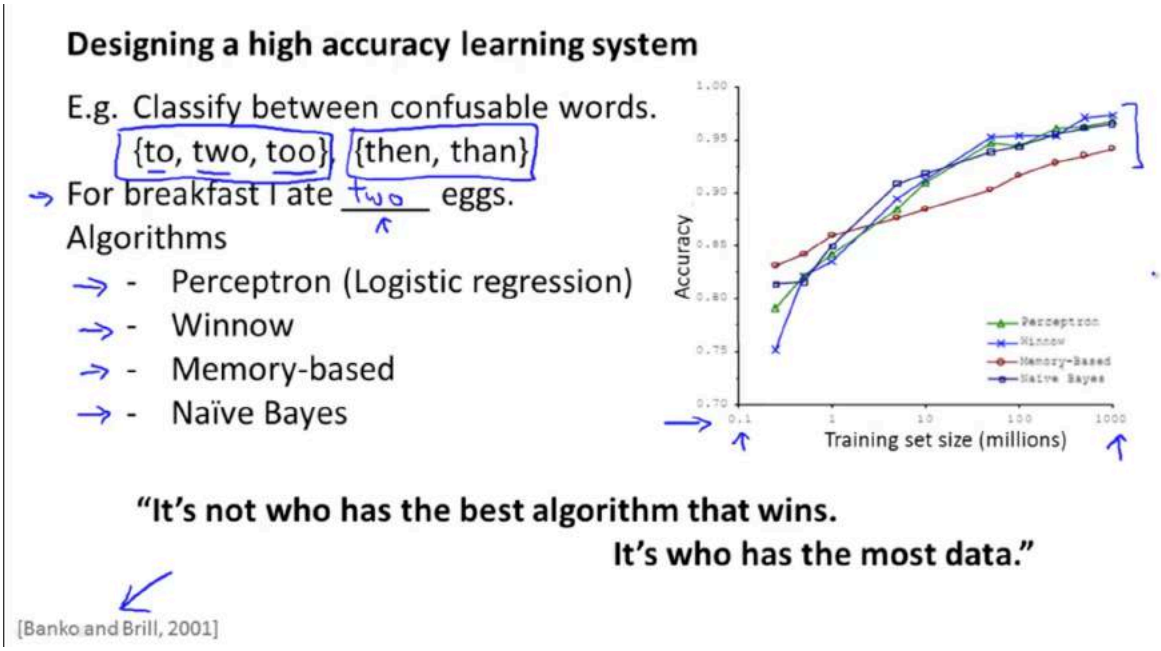
Using Large Datasets

Data For Machine Learning

In the previous video, we talked about evaluation metrics. In this video, I'd like to switch tracks a bit and touch on another important aspect of machine learning system design, which will often come up, which is the issue of how much data to train on. Now, in some earlier videos, I had cautioned against blindly going out and just spending lots of time collecting lots of data, because it's only sometimes that that would actually help. But it turns out that under certain conditions, and I will say in this video what those conditions are, getting a lot of data and training on a certain type of learning algorithm, can be a very effective way to get a learning algorithm to do very good performance. And this arises often enough that if those conditions hold true for your problem and if you're able to get a lot of data, this could be a very good way to get a very high performance learning algorithm. So in this video, let's talk more about that.

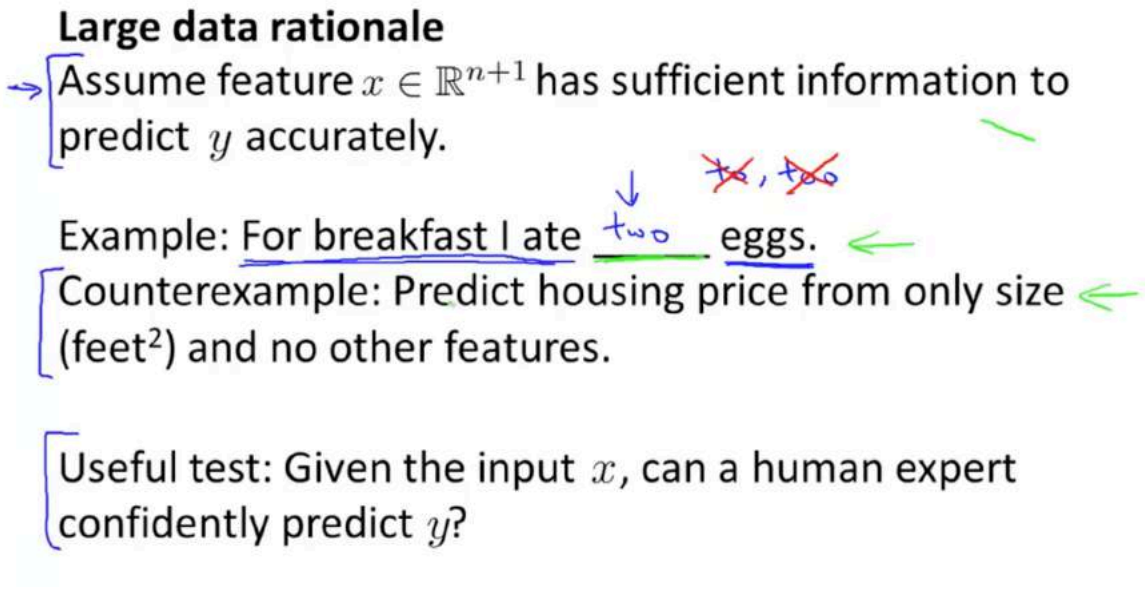
Let me start with a story. Many, many years ago, two researchers that I know, Michelle Banko and Eric Broule ran the following fascinating study. They were interested in studying the effect of using different learning algorithms versus trying them out on different training set sciences, they were considering the problem of classifying between confusable words, so for example, in the sentence: for breakfast I ate, should it be to, two or too? Well, for this example, for breakfast I ate two, 2 eggs. So, this is one example of a set of confusable words and that's a different set. So they took machine learning problems like these, sort of supervised learning problems to try to categorize what is the appropriate word to go into a certain position in an English sentence. They took a few different learning algorithms which were, you know, sort of considered state of the art back in the day, when they ran the study in 2001, so they took a variance, roughly a variance on logistic regression called the Perceptron. They also took some of their algorithms that were fairly out back then but somewhat less used now so when the algorithm also very similar to which is a regression but different in some ways, much used somewhat less, used not too much right now took what's called a memory based learning algorithm again used somewhat less now. But I'll talk a little bit about that later. And they used a naive based algorithm, which is something they'll actually talk about in this course. The exact algorithms of these details aren't important. Think of this as, you know, just picking four different classification algorithms and really the exact algorithms aren't important. But what they did was they varied the training set size and tried out these learning algorithms on the range of training set sizes and that's the result they got. And the trends are

very clear right first most of these outer rooms give remarkably similar performance. And second, as the training set size increases, on the horizontal axis is the training set size in millions go from you know a hundred thousand up to a thousand million that is a billion training examples. The performance of the algorithms all pretty much monotonically increase and the fact that if you pick any algorithm may be pick a "inferior algorithm" but if you give that "inferior algorithm" more data, then from these examples, it looks like it will most likely beat even a "superior algorithm". So since this original study which is very influential, there's been a range of many different studies showing similar results that show that many different learning algorithms you know tend to, can sometimes, depending on details, can give pretty similar ranges of performance, but what can really drive performance is you can give the algorithm a ton of training data. And this is, results like these has led to a saying in machine learning that often in machine learning it's not who has the best algorithm that wins, it's who has the most data So when is this true and when is this not true? Because we have a learning algorithm for which this is true then getting a lot of data is often maybe the best way to ensure that we have an algorithm with very high performance rather than you know, debating worrying about exactly which of these items to use.



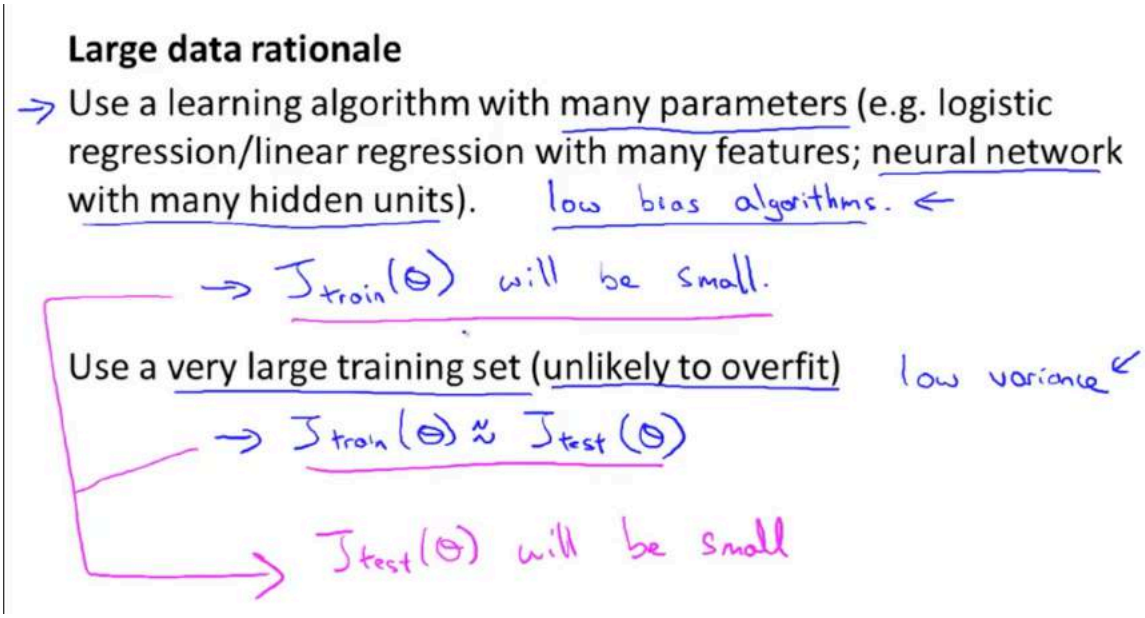
Let's try to lay out a set of assumptions under which having a massive training set we think will be able to help. Let's assume that in our machine learning problem, the features x have sufficient information with which we can use to predict y accurately. For example, if we take the confusable words all of them that we had on the previous slide. Let's say that it features x capture what are the surrounding words around the blank that we're trying to fill in. So the features capture then we want to have, sometimes for breakfast I have black eggs. Then yeah that is pretty much information to tell me that the word I want

in the middle is TWO and that is not word TO and its not the word TOO. So the features capture, you know, one of these surrounding words then that gives me enough information to pretty unambiguously decide what is the label y or in other words what is the word that I should be using to fill in that blank out of this set of three confusable words. So that's an example what the future ex has sufficient information for specific y. For a counter example. Consider a problem of predicting the price of a house from only the size of the house and from no other features. So if you imagine I tell you that a house is, you know, 500 square feet but I don't give you any other features. I don't tell you that the house is in an expensive part of the city. Or if I don't tell you that the house, the number of rooms in the house, or how nicely furnished the house is, or whether the house is new or old. If I don't tell you anything other than that this is a 500 square foot house, well there's so many other factors that would affect the price of a house other than just the size of a house that if all you know is the size, it's actually very difficult to predict the price accurately. So that would be a counter example to this assumption that the features have sufficient information to predict the price to the desired level of accuracy. The way I think about testing this assumption, one way I often think about it is, how often I ask myself. Given the input features x, given the features, given the same information available as well as learning algorithm. If we were to go to human expert in this domain. Can a human experts actually or can human expert confidently predict the value of y. For this first example if we go to, you know an expert human English speaker. You go to someone that speaks English well, right, then a human expert in English just read most people like you and me will probably we would probably be able to predict what word should go in here, to a good English speaker can predict this well, and so this gives me confidence that x allows us to predict y accurately, but in contrast if we go to an expert in human prices. Like maybe an expert realtor, right, someone who sells houses for a living. If I just tell them the size of a house and I tell them what the price is well even an expert in pricing or selling houses wouldn't be able to tell me and so this is fine that for the housing price example knowing only the size doesn't give me enough information to predict the price of the house.



So, let's say, this assumption holds. Let's see then, when having a lot of data could help. Suppose the features have enough information to predict the value of y. And let's suppose we use a learning algorithm with a large number of parameters so maybe logistic regression or linear regression with a large number of features. Or one thing that I sometimes do, one thing that I often do actually is using neural network with many hidden units. That would be another learning algorithm with a lot of parameters. So these are all powerful learning algorithms with a lot of parameters that can fit very complex functions. So, I'm going to call these, I'm going to think of these as low-bias algorithms because you know we can fit very complex functions and because we have a very powerful learning algorithm, they can fit very complex functions. Chances are, if we run these algorithms on the data sets, it will be able to fit the training set well, and so hopefully the training error will be slow. Now let's say, we use a massive, massive training set, in that case, if we have a huge training set, then hopefully even though we have a lot of parameters but if the training set is sort of even much larger than the number of parameters then hopefully these albums will be unlikely to overfit. Right because we have such a massive training set and by unlikely to overfit what that means is that the training error will hopefully be close to the test error. Finally putting these two together that the train set error is small and the test set error is close to the training error what this two together imply is that hopefully the test set error will also be Another way to think about this is that in order to have a high performance learning algorithm we want it not to have high bias and not to have high variance. So the bias problem we're going to address by making sure we have a learning algorithm with many parameters and so that gives us a low bias algorithm and by using a very large training set, this ensures that we don't have a variance problem here. So hopefully our algorithm will have no variance and so is by pulling these two together, that we end up with a low bias and a

low variance learning algorithm and this allows us to do well on the test set. And fundamentally it's a key ingredients of assuming that the features have enough information and we have a rich class of functions that's why it guarantees low and then it having a massive training set that that's what guarantees more variance.



Question

Having a large training set can help significantly improve a learning algorithm's performance. However, the large training set is **unlikely** to help when:

- [x] The features x do not contain enough information to predict y accurately (such as predicting a house's price from only its size), and we are using a simple learning algorithm such as logistic regression.

  Correct

- [ ] We are using a learning algorithm with a large number of features (i.e. one with "low bias").

  Un-selected is correct

- [x] The features x do not contain enough information to predict y accurately (such as predicting a house's price from only its size), even if we are using a neural network with a large number of hidden units.

  Correct

- [ ] We are not using regularization (e.g. the regularization parameter $\lambda = 0$).

  Un-selected is correct

So this gives us a set of conditions rather hopefully some understanding of what's the sort of problem where if you have a lot of data and you train a

learning algorithm with lot of parameters, that might be a good way to give a high performance learning algorithm and really, I think the key test that I often ask myself are first, can a human experts look at the features x and confidently predict the value of y. Because that's sort of a certification that y can be predicted accurately from the features x and second, can we actually get a large training set, and train the learning algorithm with a lot of parameters in the training set and if you can't do both then that's more often give you a very kind performance learning algorithm.

Review

Quiz

1. You are working on a spam classification system using regularized logistic regression. "Spam" is a positive class (y = 1) and "not spam" is the negative class (y = 0). You have trained your classifier and there are m = 1000 examples in the cross-validation set. The chart of predicted class vs. actual class is:

	Actual Class: 1	**Actual Class: 0**
Predicted Class: 1	85	890
Predicted Class: 0	15	10

For reference:

- Accuracy = (true positives + true negatives) / (total examples)
- Precision = (true positives) / (true positives + false positives)
- Recall = (true positives) / (true positives + false negatives)
- F_1 score = (2 * precision * recall) / (precision + recall)

What is the classifier's recall (as a value from 0 to 1)?

Enter your answer in the box below. If necessary, provide at least two values after the decimal point.

0.85

2. Suppose a massive dataset is available for training a learning algorithm. Training on a lot of data is likely to give good performance when two of the following conditions hold true.

Which are the two?

- [x] The features x contain sufficient information to predict y accurately. (For example, one way to verify this is if a human expert on the domain can confidently predict y when given only x).
- [x] We train a learning algorithm with a large number of parameters (that is able to learn/represent fairly complex functions).
- [ ] We train a learning algorithm with a small number of parameters (that is thus unlikely to overfit).
- [ ] When we are willing to include high order polynomial features of x (such as x_1^2, x_2^2, x_1x_2, etc.).

3. Suppose you have trained a logistic regression classifier which is outputing $h_\theta(x)$.

Currently, you predict 1 if $h_\theta(x) \geq \text{threshold}$, and predict 0 if $h_\theta(x) < \text{threshold}$, where currently the threshold is set to 0.5.

Suppose you **increase** the threshold to 0.7. Which of the following are true? Check all that apply.

- [ ] The classifier is likely to now have higher recall.
- [ ] The classifier is likely to have unchanged precision and recall, and thus the same F_1 score.
- [x] The classifier is likely to now have higher precision.
- [ ] The classifier is likely to have unchanged precision and recall, but higher accuracy.

4. Suppose you are working on a spam classifier, where spam emails are positive examples ($y = 1$) and non-spam emails are negative examples ($y = 0$). You have a training set of emails in which 99% of the emails are non-spam and the other 1% is spam. Which of the following statements are true? Check all that apply.

- [ ] If you always predict spam (output $y = 1$), your classifier will have a recall of 0% and precision of 99%.

- [x] If you always predict non-spam (output $y = 0$), your classifier will have a recall of 0%.
- [x] If you always predict non-spam (output $y = 0$), your classifier will have an accuracy of 99%.
- [x] If you always predict spam (output $y = 1$), your classifier will have a recall of 100% and precision of 1%.

5. Which of the following statements are true? Check all that apply.

- [x] The "error analysis" process of manually examining the examples which your algorithm got wrong can help suggest what are good steps to take (e.g., developing new features) to improve your algorithm's performance.
- [ ] After training a logistic regression classifier, you **must** use 0.5 as your threshold for predicting whether an example is positive or negative.

- [ ] It is a good idea to spend a lot of time collecting a **large** amount of data before building your first version of a learning algorithm.
- [ ] If your model is underfitting the training set, then obtaining more data is likely to help.
- [x] Using a **very large** training set makes it unlikely for model to overfit the training data.

WEEK 7

Large Margin Classification

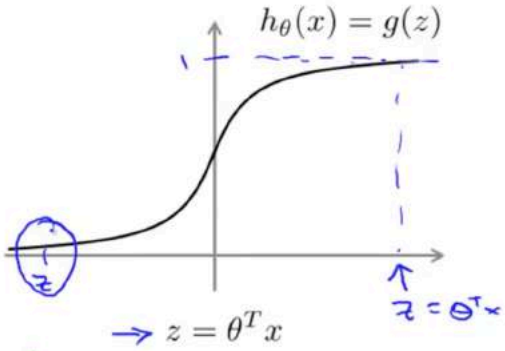
Optimization Objective

By now, you've seen a range of difference learning algorithms. With supervised learning, the performance of many supervised learning algorithms will be pretty similar, and what matters less often will be whether you use learning algorithm a or learning algorithm b, but what matters more will often be things like the amount of data you create these algorithms on, as well as your skill in applying these algorithms. Things like your choice of the features you design to give to the learning algorithms, and how you choose the regularization parameter, and things like that. But, there's one more algorithm that is very powerful and is very widely used both within industry and academia, and that's called the support vector machine. And compared to both logistic regression and neural networks, the Support Vector Machine, or SVM sometimes gives a cleaner, and sometimes more powerful way of learning complex non-linear functions. And so let's take the next videos to talk about that. Later in this course, I will do a quick survey of a range of different supervisory algorithms just as a very briefly describe them. But the support vector machine, given its popularity and how powerful it is, this will be the last of the supervisory algorithms that I'll spend a significant amount of time on in this course as with our development other learning algorithms, we're gonna start by talking about the optimization objective. So, let's get started on this algorithm.

In order to describe the support vector machine, I'm actually going to start with logistic regression, and show how we can modify it a bit, and get what is essentially the support vector machine. So in logistic regression, we have our familiar form of the hypothesis there and the sigmoid activation function shown on the right. And in order to explain some of the math, I'm going to use z to denote theta transpose axiom. Now let's think about what we would like logistic regression to do. If we have an example with y equals one and by this I mean an example in either the training set or the test set or the cross-validation set, but when y is equal to one then we're sort of hoping that h of x will be close to one. Right, we're hoping to correctly classify that example. And what having x subscript 1, what that means is that theta transpose x must be must larger than 0. So there's greater than, greater than sign that means much, much greater than 0. And that's because it is z, the theta of transpose x is when z is much bigger than 0 is far to the right of the sphere. That the outputs of logistic progression becomes close to one. Conversely, if we have an example where y is equal to zero, then what we're hoping for is that the hypothesis will output a value close to zero. And that corresponds to theta transpose x of z being much less than zero because that corresponds to a hypothesis of putting a value close to zero.

Alternative view of logistic regression

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

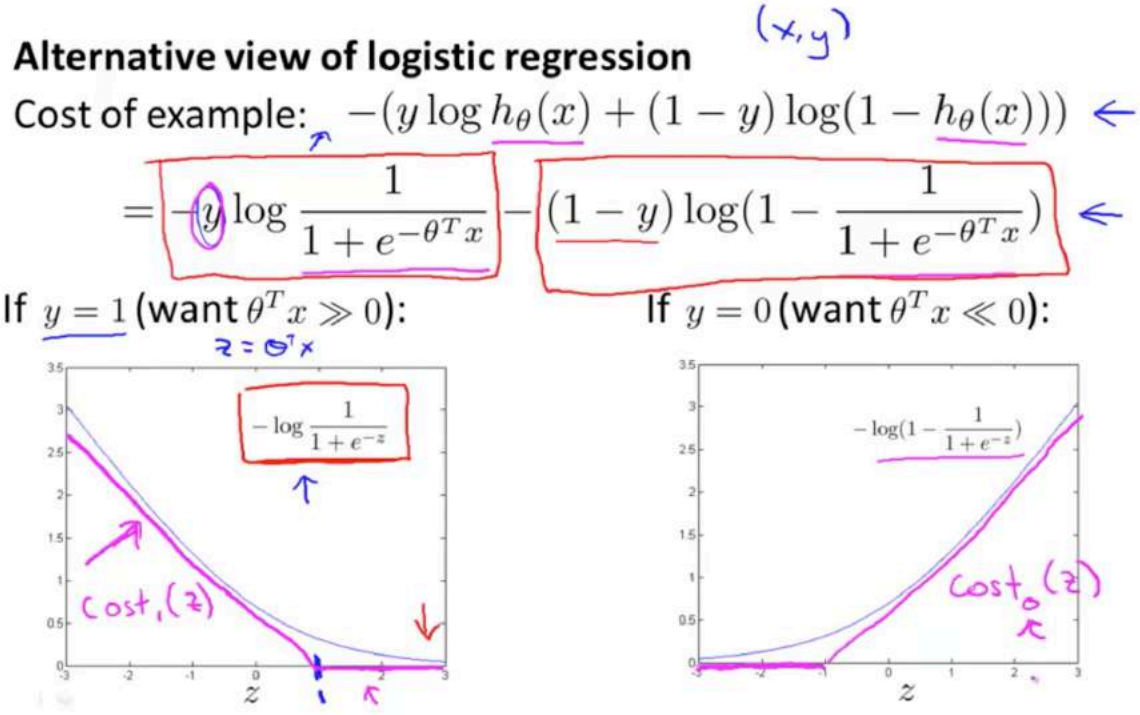


If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

If you look at the cost function of logistic regression, what you'll find is that each example (x,y) contributes a term like this to the overall cost function, right? So for the overall cost function, we will also have a sum over all the chain examples and the 1 over m term, that this expression here, that's the term that a single training example contributes to the overall objective function so we can just rush them. Now if I take the definition for the fall of my hypothesis and plug it in over here, then what I get is that each training example contributes this term, ignoring the one over M but it contributes that term to my overall cost function for logistic regression. Now let's consider two cases of when y is equal to one and when y is equal to zero. In the first case, let's suppose that y is equal to 1. In that case, only this first term in the objective matters, because this one minus y term would be equal to zero if y is equal to one. So when y is equal to one, when in our example x comma y, when y is equal to 1 what we get is this term.. Minus log one over one, plus E to the negative Z where as similar to the last line I'm using Z to denote data transposed X and of course in a cost I should have this minus line that we just had if Y is equal to one so that's equal to one I just simplify in a way in the expression that I have written down here. And if we plot this function as a function of z, what you find is that you get this curve shown on the lower left of the slide. And thus, we also see that when z is equal to large, that is, when theta transpose x is large, that corresponds to a value of z that gives us a fairly small value, a very, very small contribution to the consumption. And this kinda explains why, when logistic regression sees a positive example, with y=1, it tries to set theta transport x to be very large because that corresponds to this term, in the cross function, being small. Now, to fill the support vector machine, here's what we're going to do. We're gonna take this cross function, this minus log 1 over 1 plus e to negative z, and modify it a little bit. Let me take

this point 1 over here, and let me draw the cross functions you're going to use. The new pass functions can be flat from here on out, and then we draw something that grows as a straight line, similar to logistic regression. But this is going to be a straight line at this portion. So the curve that I just drew in magenta, and the curve I just drew purple and magenta, so if it's pretty close approximation to the cross function used by logistic regression. Except it is now made up of two line segments, there's this flat portion on the right, and then there's this straight line portion on the left. And don't worry too much about the slope of the straight line portion. It doesn't matter that much. But that's the new cost function we're going to use for when y is equal to one, and you can imagine it should do something pretty similar to logistic regression. But turns out, that this will give the support vector machine computational advantages and give us, later on, an easier optimization problem that would be easier for software to solve. We just talked about the case of y equals one. The other case is if y is equal to zero. In that case, if you look at the cost, then only the second term will apply because the first term goes away, right? If y is equal to zero, then you have a zero here, so you're left only with the second term of the expression above. And so the cost of an example, or the contribution of the cost function, is going to be given by this term over here. And if you plot that as a function of z, to have pure z on the horizontal axis, you end up with this one. And for the support vector machine, once again, we're going to replace this blue line with something similar and at the same time we replace it with a new cost, this flat out here, this 0 out here. And that then grows as a straight line, like so. So let me give these two functions names. This function on the left I'm going to call cost subscript 1 of z, and this function of the right I'm gonna call cost subscript 0 of z. And the subscript just refers to the cost corresponding to when y is equal to 1, versus when y Is equal to zero. Armed with these definitions, we're now ready to build a support vector machine.



Here's the cost function, j of theta, that we have for logistic regression. In case this equation looks a bit unfamiliar, it's because previously we had a minus sign outside, but here what I did was I instead moved the minus signs inside these expressions, so it just makes it look a little different. For the support vector machine what we're going to do is essentially take this and replace this with cost1 of z, that is cost1 of theta transpose x. And we're going to take this and replace it with cost0 of z, that is cost0 of theta transpose x. Where the cost one function is what we had on the previous slide that looks like this. And the cost zero function, again what we had on the previous slide, and it looks like this. So what we have for the support vector machine is a minimization problem of one over M, the sum of Y I times cost one, theta transpose X I, plus one minus Y I times cause zero of theta transpose X I, and then plus my usual regularization parameter. Like so. Now, by convention, for the support of vector machine, we're actually write things slightly different. We re-parameterize this just very slightly differently. First, we're going to get rid of the 1 over m terms, and this just this happens to be a slightly different convention that people use for support vector machines compared to or just a progression. But here's what I mean. You're one way to do this, we're just gonna get rid of these one over m terms and this should give you me the same optimal value of beta right? Because one over m is just as constant so whether I solved this minimization problem with one over n in front or not. I should end up with the same optimal value for theta. Here's what I mean, to give you an example, suppose I had a minimization problem. Minimize over a long number U of U minus five squared plus one. Well, the minimum of this happens to be U equals five. Now if I were to take this objective function and multiply it by 10. So here my minimization problem is min over U, 10 U minus five squared plus 10. Well the value of U that minimizes this is still U equals five right? So multiply something that you're minimizing over, by some constant, 10 in this case, it does not change the value of U that gives us, that minimizes this function. So the same way, what I've done is by crossing out the M is all I'm doing is multiplying my objective function by some constant M and it doesn't change the value of theta. That achieves the minimum. The second bit of notational change, which is just, again, the more standard convention when using SVMs instead of logistic regression, is the following. So for logistic regression, we add two terms to the objective function. The first is this term, which is the cost that comes from the training set and the second is this row, which is the regularization term. And what we had was we had a, we control the trade-off between these by saying, what we want is A plus, and then my regularization parameter lambda. And then times some other term B, where I guess I'm using your A to denote this first term, and I'm using B to denote the second term, maybe without the lambda. And instead of prioritizing this as A plus lambda B, and so what we did was by setting different values for this regularization parameter lambda, we could trade off the relative weight between how much

we wanted the training set well, that is, minimizing A, versus how much we care about keeping the values of the parameter small, so that will be, the parameter is B for the support vector machine, just by convention, we're going to use a different parameter. So instead of using lambda here to control the relative waiting between the first and second terms. We're instead going to use a different parameter which by convention is called C and is set to minimize C times a + B. So for logistic regression, if we set a very large value of lambda, that means you will give B a very high weight. Here is that if we set C to be a very small value, then that responds to giving B a much larger rate than C, than A. So this is just a different way of controlling the trade off, it's just a different way of prioritizing how much we care about optimizing the first term, versus how much we care about optimizing the second term. And if you want you can think of this as the parameter C playing a role similar to 1 over lambda. And it's not that it's two equations or these two expressions will be equal. This equals 1 over lambda, that's not the case. It's rather that if C is equal to 1 over lambda, then these two optimization objectives should give you the same value the same optimal value for theta so we just filling that in I'm gonna cross out lambda here and write in the constant C there. So that gives us our overall optimization objective function for the support vector machine. And if you minimize that function, then what you have is the parameters learned by the SVM.

Consider the following minimization problems:

1. $\min_\theta \ \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

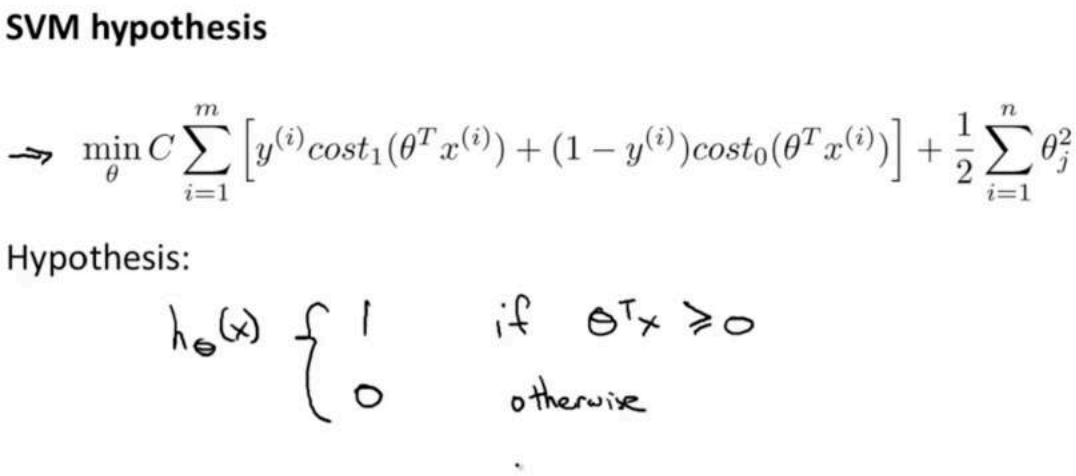
2. $\min_\theta \ C\left[\sum_{i=1}^{m} y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$

These two optimization problems will give the same value of θ (i.e., the same value of θ gives the optimal solution to both problems) if:

- ○ $C = \lambda$
- ○ $C = -\lambda$
- ● $C = \frac{1}{\lambda}$
- ○ $C = \frac{2}{\lambda}$

Finally unlike logistic regression, the support vector machine doesn't output the probability is that what we have is we have this cost function, that we minimize to get the parameter's data, and what a support vector machine does is it just makes a prediction of y being equal to one or zero, directly. So the hypothesis will predict one if theta transpose x is greater or equal to zero, and

it will predict zero otherwise and so having learned the parameters theta, this is the form of the hypothesis for the support vector machine.

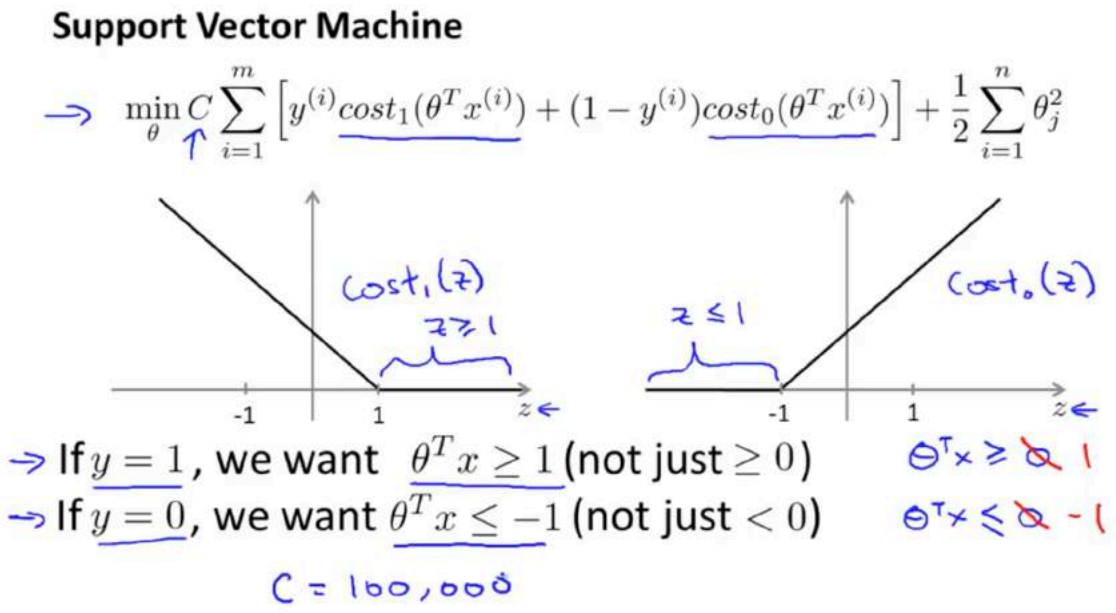


So that was a mathematical definition of what a support vector machine does. In the next few videos, let's try to get back to intuition about what this optimization objective leads to and whether the source of the hypotheses SVM will learn and we'll also talk about how to modify this just a little bit to the complex nonlinear functions.

Large Margin Intuition

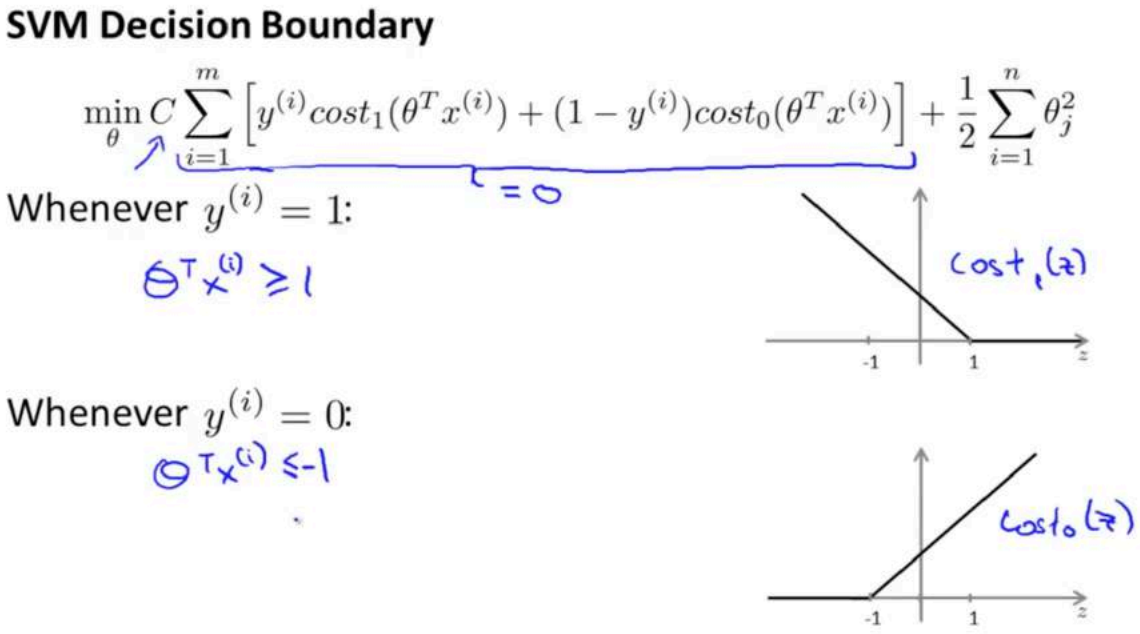
Sometimes people talk about support vector machines, as large margin classifiers, in this video I'd like to tell you what that means, and this will also give us a useful picture of what an SVM hypothesis may look like. Here's my cost function for the support vector machine where here on the left I've plotted my cost 1 of z function that I used for positive examples and on the right I've plotted my zero of 'Z' function, where I have 'Z' here on the horizontal axis. Now, let's think about what it takes to make these cost functions small. If you have a positive example, so if y is equal to 1, then cost 1 of Z is zero only when Z is greater than or equal to 1. So in other words, if you have a positive example, we really want theta transpose x to be greater than or equal to 1 and conversely if y is equal to zero, look this cost zero of z function, then it's only in this region where z is less than equal to 1 we have the cost is zero as z is equals to zero, and this is an interesting property of the support vector machine right, which is that, if you have a positive example so if y is equal to one, then all we really need is that theta transpose x is greater than equal to

zero. And that would mean that we classify correctly because if theta transpose x is greater than zero our hypothesis will predict zero. And similarly, if you have a negative example, then really all you want is that theta transpose x is less than zero and that will make sure we got the example right. But the support vector machine wants a bit more than that. It says, you know, don't just barely get the example right. So then don't just have it just a little bit bigger than zero. What i really want is for this to be quite a lot bigger than zero say maybe bit greater or equal to one and I want this to be much less than zero. Maybe I want it less than or equal to -1. And so this builds in an extra safety factor or safety margin factor into the support vector machine. Logistic regression does something similar too of course, but let's see what happens or let's see what the consequences of this are, in the context of the support vector machine. Concretely, what I'd like to do next is consider a case case where we set this constant C to be a very large value, so let's imagine we set C to a very large value, may be a hundred thousand, some huge number. Let's see what the support vector machine will do.



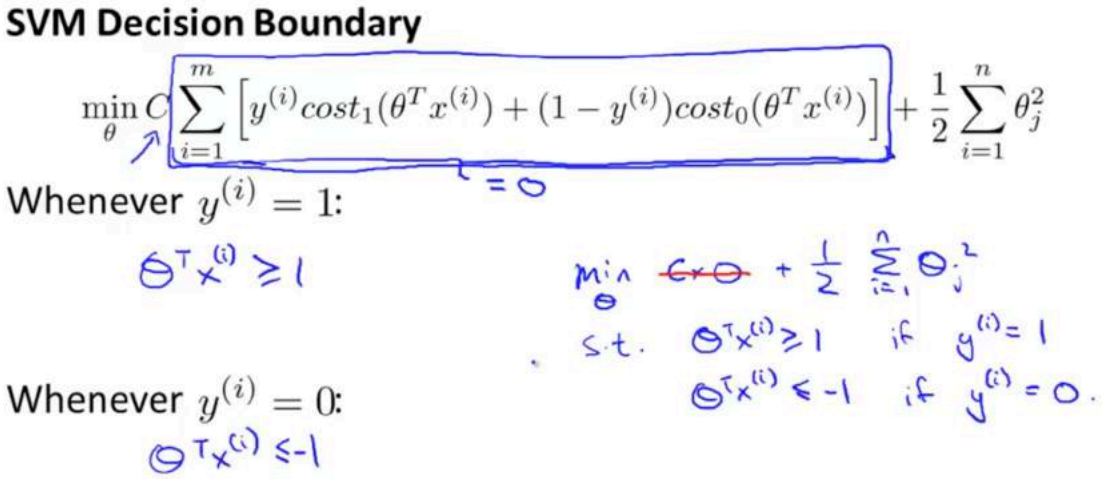
If C is very, very large, then when minimizing this optimization objective, we're going to be highly motivated to choose a value, so that this first term is equal to zero. So let's try to understand the optimization problem in the context of, what would it take to make this first term in the objective equal to zero, because you know, maybe we'll set C to some huge constant, and this will hope, this should give us additional intuition about what sort of hypotheses a support vector machine learns. So we saw already that whenever you have a training example with a label of y=1 if you want to make that first term zero, what you need is is to find a value of theta so that theta transpose x i is greater than or equal to 1. And similarly, whenever we have an example, with label zero, in order to make sure that the cost, cost zero of Z, in order to make

sure that cost is zero we need that theta transpose x i is less than or equal to -1.



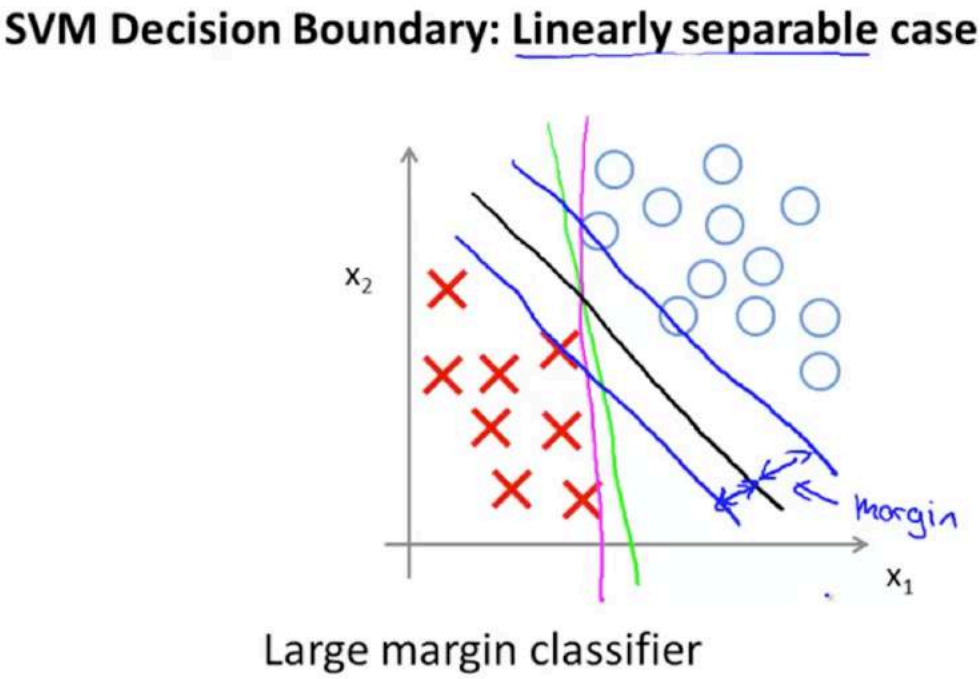
And so this builds in an extra safety factor or safety margin factor into the support vector machine. Logistic regression does something similar too of course, but let's see what happens or let's see what the consequences of this are, in the context of the support vector machine. Concretely, what I'd like to do next is consider a case case where we set this constant C to be a very large value, so let's imagine we set C to a very large value, may be a hundred thousand, some huge number. Let's see what the support vector machine will do. If C is very, very large, then when minimizing this optimization objective, we're going to be highly motivated to choose a value, so that this first term is equal to zero. So let's try to understand the optimization problem in the context of, what would it take to make this first term in the objective equal to zero, because you know, maybe we'll set C to some huge constant, and this will hope, this should give us additional intuition about what sort of hypotheses a support vector machine learns. So we saw already that whenever you have a training example with a label of y=1 if you want to make that first term zero, what you need is is to find a value of theta so that theta transpose x i is greater than or equal to 1. And similarly, whenever we have an example, with label zero, in order to make sure that the cost, cost zero of Z, in order to make sure that cost is zero we need that theta transpose x i is less than or equal to -1. So, if we think of our optimization problem as now, really choosing parameters and show that this first term is equal to zero, what we're left with is the following optimization problem. We're going to minimize that first term zero, so C times zero, because we're going to choose parameters so that's equal to zero, plus one half and then you know that second term and this first term is 'C' times zero, so let's just cross that out because I know that's going to

be zero. And this will be subject to the constraint that theta transpose x(i) is greater than or equal to one, if y(i) Is equal to one and theta transpose x(i) is less than or equal to minus one whenever you have a negative example and it turns out that when you solve this optimization problem, when you minimize this as a function of the parameters theta you get a very interesting decision boundary.



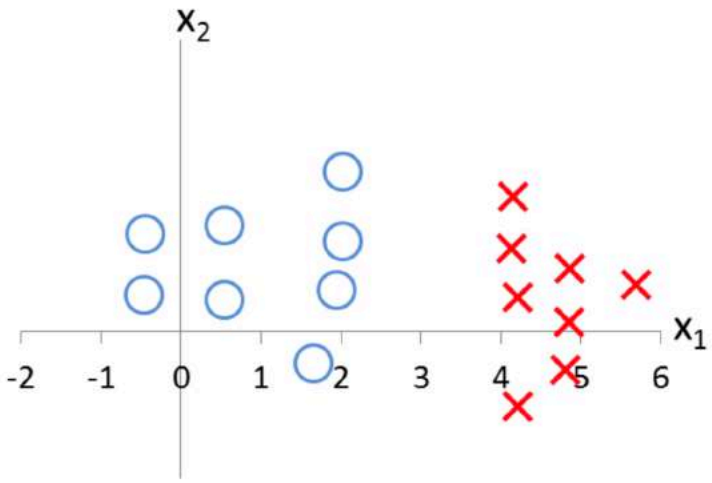
Concretely, if you look at a data set like this with positive and negative examples, this data is linearly separable and by that, I mean that there exists, you know, a straight line, although there is many a different straight lines, they can separate the positive and negative examples perfectly. For example, here is one decision boundary that separates the positive and negative examples, but somehow that doesn't look like a very natural one, right? Or by drawing an even worse one, you know here's another decision boundary that separates the positive and negative examples but just barely. But neither of those seem like particularly good choices. The Support Vector Machines will instead choose this decision boundary, which I'm drawing in black. And that seems like a much better decision boundary than either of the ones that I drew in magenta or in green. The black line seems like a more robust separator, it does a better job of separating the positive and negative examples. And mathematically, what that does is, this black decision boundary has a larger distance. That distance is called the margin, when I draw up this two extra blue lines, we see that the black decision boundary has some larger minimum distance from any of my training examples, whereas the magenta and the green lines they come awfully close to the training examples. and then that seems to do a less a good job separating the positive and negative classes than my black line. And so this distance is called the margin of the support vector machine and this gives the SVM a certain robustness, because it tries to separate the data with as a large a margin as possible. So the support vector machine is sometimes also called a large margin classifier and this is actually a consequence of the optimization problem we wrote down on the previous

slide. I know that you might be wondering how is it that the optimization problem I wrote down in the previous slide, how does that lead to this large margin classifier. I know I haven't explained that yet. And in the next video I'm going to sketch a little bit of the intuition about why that optimization problem gives us this large margin classifier. But this is a useful feature to keep in mind if you are trying to understand what are the sorts of hypothesis that an SVM will choose. That is, trying to separate the positive and negative examples with as big a margin as possible.



Question

Consider the training set to the right, where "x" denotes positive examples ($y = 1$) and "o" denotes negative examples ($y = 0$). Suppose you train an SVM (which will predict 1 when $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$). What values might the SVM give for θ_0, θ_1, and θ_2?



$\theta_0 = 3, \theta_1 = 1, \theta_2 = 0$

$\theta_0 = -3, \theta_1 = 1, \theta_2 = 0$

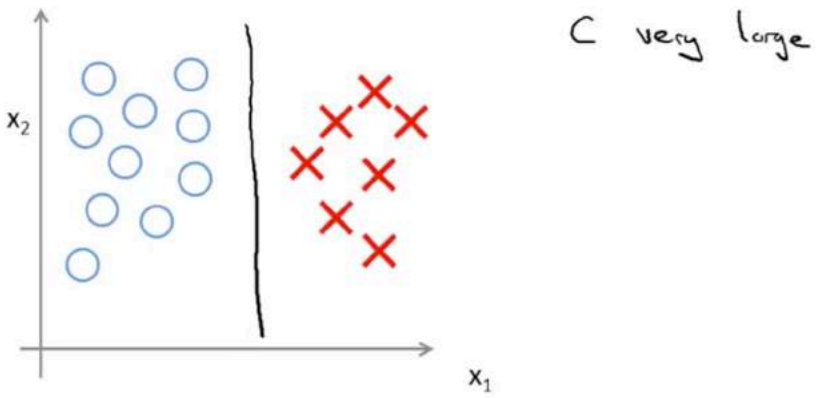
Correct

$\theta_0 = 3, \theta_1 = 0, \theta_2 = 1$

$\theta_0 = -3, \theta_1 = 0, \theta_2 = 1$

I want to say one last thing about large margin classifiers in this intuition, so we wrote out this large margin classification setting in the case of when C, that regularization concept, was very large, I think I set that to a hundred thousand or something. So given a dataset like this, maybe we'll choose that decision boundary that separate the positive and negative examples on large margin.

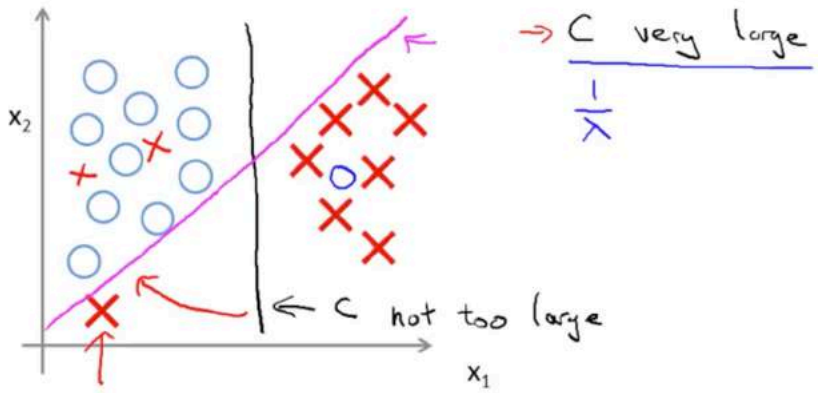
Large margin classifier in presence of outliers



Now, the SVM is actually slightly more sophisticated than this large margin view might suggest. And in particular, if all you're doing is use a large margin classifier then your learning algorithms can be sensitive to outliers, so lets just add an extra positive example like that shown on the screen. If he had one example then it seems as if to separate data with a large margin, maybe I'll end up learning a decision boundary like that, right? that is the magenta line and it's really not clear that based on the single outlier based on a single example and it's really not clear that it's actually a good idea to change my decision boundary from the black one over to the magenta one. So, if C, if the regularization parameter C were very large, then this is actually what SVM will do, it will change the decision boundary from the black to the magenta one but if C were reasonably small if you were to use the C, not too large then you still

end up with this black decision boundary. And of course if the data were not linearly separable so if you had some positive examples in here, or if you had some negative examples in here then the SVM will also do the right thing. And so this picture of a large margin classifier that's really, that's really the picture that gives better intuition only for the case of when the regulations parameter C is very large, and just to remind you this corresponds C plays a role similar to one over Lambda, where Lambda is the regularization parameter we had previously. And so it's only of one over Lambda is very large or equivalently if Lambda is very small that you end up with things like this Magenta decision boundary, but in practice when applying support vector machines, when C is not very very large like that, it can do a better job ignoring the few outliers like here. And also do fine and do reasonable things even if your data is not linearly separable. But when we talk about bias and variance in the context of support vector machines which will do a little bit later, hopefully all of of this trade-offs involving the regularization parameter will become clearer at that time.

Large margin classifier in presence of outliers

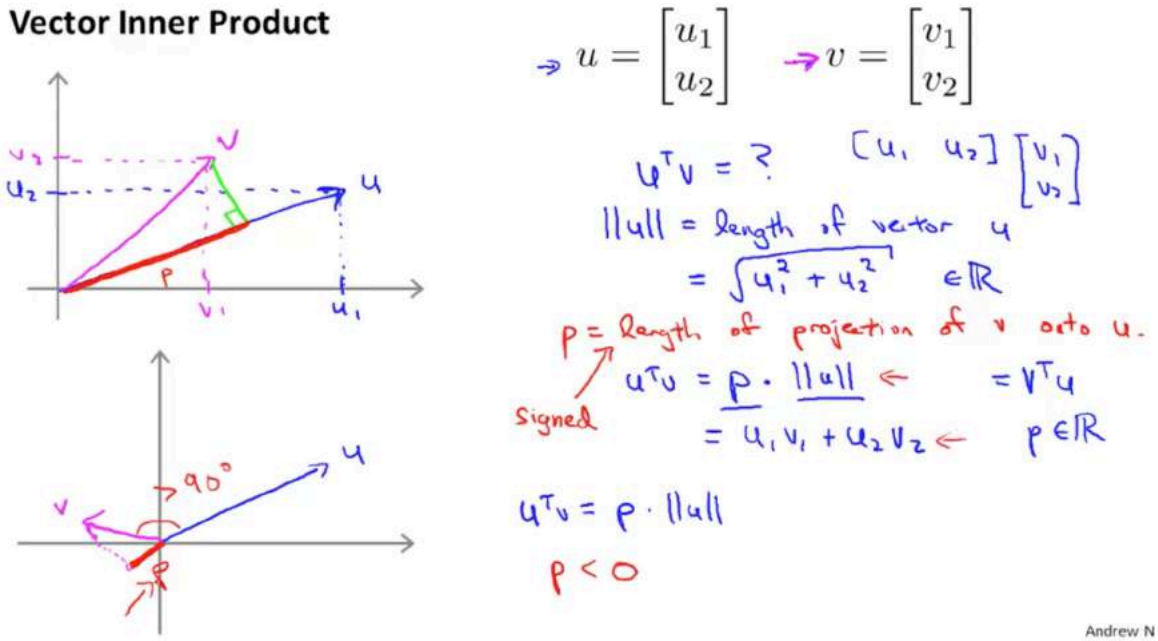


So I hope that gives some intuition about how this support vector machine functions as a large margin classifier that tries to separate the data with a large margin, technically this picture of this view is true only when the parameter C is very large, which is a useful way to think about support vector machines. There was one missing step in this video which is, why is it that the optimization problem we wrote down on these slides, how does that actually lead to the large margin classifier, I didn't do that in this video, in the next video I will sketch a little bit more of the math behind that to explain that separate reasoning of how the optimization problem we wrote out results in a large margin classifier.

Mathematics Behind Large Margin Classification

In this video, I'd like to tell you a bit about the math behind large margin classification. This video is optional, so please feel free to skip it. It may also give you better intuition about how the optimization problem of the support vex machine, how that leads to large margin classifiers. In order to get started, let me first remind you of a couple of properties of what vector inner products look like. Let's say I have two vectors U and V, that look like this. So both two dimensional vectors. Then let's see what U transpose V looks like. And U transpose V is also called the inner products between the vectors U and V. Use a two dimensional vector, so I can on plot it on this figure. So let's say that's the vector U. And what I mean by that is if on the horizontal axis that value takes whatever value U1 is and on the vertical axis the height of that is whatever U2 is the second component of the vector U. Now, one quantity that will be nice to have is the norm of the vector U. So, these are, you know, double bars on the left and right that denotes the norm or length of U. So this just means; really the euclidean length of the vector U. And this is Pythagoras theorem is just equal to U1 squared plus U2 squared square root, right? And this is the length of the vector U. That's a real number. Just say you know, what is the length of this, what is the length of this vector down here. What is the length of this arrow that I just drew, is the normal view? Now let's go back and look at the vector V because we want to compute the inner product. So V will be some other vector with, you know, some value V1, V2. And so, the vector V will look like that, towards V like so. Now let's go back and look at how to compute the inner product between U and V. Here's how you can do it. Let me take the vector V and project it down onto the vector U. So I'm going to take a orthogonal projection or a 90 degree projection, and project it down onto U like so. And what I'm going to do measure length of this red line that I just drew here. So, I'm going to call the length of that red line P. So, P is the length or is the magnitude of the projection of the vector V onto the vector U. Let me just write that down. So, P is the length of the projection of the vector V onto the vector U. And it is possible to show that unit product U transpose V, that this is going to be equal to P times the norm or the length of the vector U. So, this is one way to compute the inner product. And if you actually do the geometry figure out what P is and figure out what the norm of U is. This should give you the same way, the same answer as the other way of computing unit product. Right. Which is if you take U transpose V then U transposes this U1 U2, its a one by two matrix, 1 times V. And so this should actually give you U1, V1 plus U2, V2. And so the theorem of linear algebra that these two formulas give you the same answer. And by the way, U transpose V is also equal to V transpose U. So if you were to do the same process in reverse, instead of projecting V onto U, you could project U onto V. Then, you know, do the same process, but

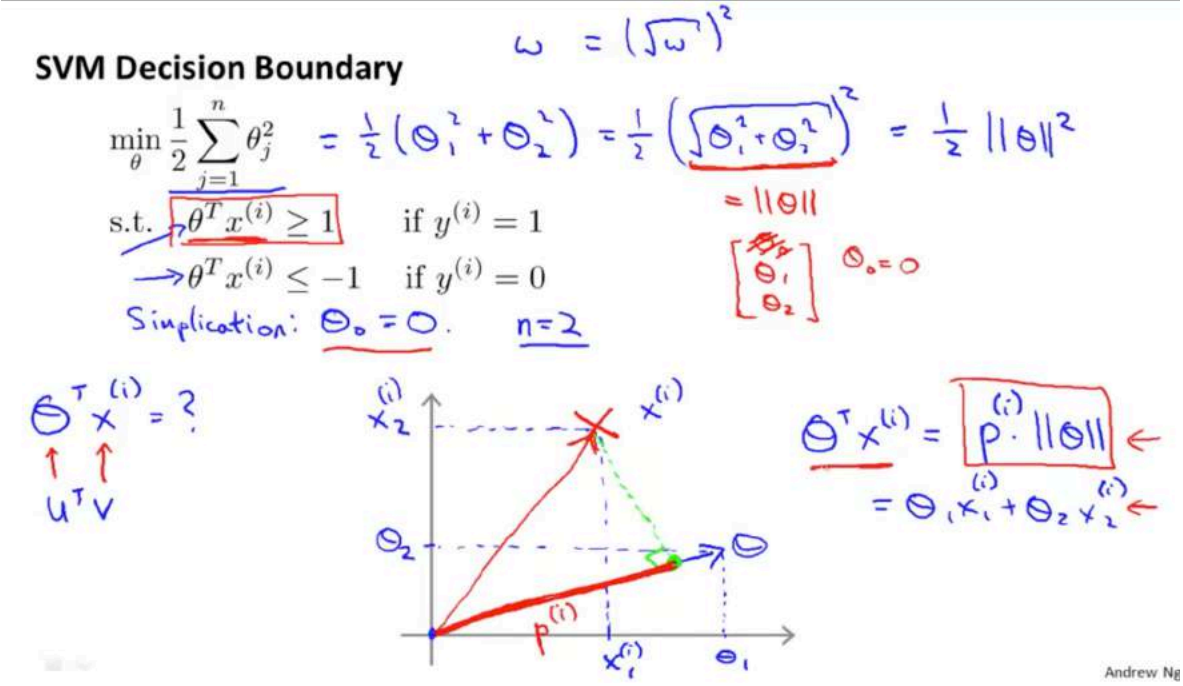
with the rows of U and V reversed. And you would actually, you should actually get the same number whatever that number is. And just to clarify what's going on in this equation the norm of U is a real number and P is also a real number. And so U transpose V is the regular multiplication as two real numbers of the length of P times the normal view. Just one last detail, which is if you look at the norm of P, P is actually signed so to the right. And it can either be positive or negative. So let me say what I mean by that, if U is a vector that looks like this and V is a vector that looks like this. So if the angle between U and V is greater than ninety degrees. Then if I project V onto U, what I get is a projection it looks like this and so that length P. And in this case, I will still have that U transpose V is equal to P times the norm of U. Except in this example P will be negative. So, you know, in inner products if the angle between U and V is less than ninety degrees, then P is the positive length for that red line whereas if the angle of this angle of here is greater than 90 degrees then P here will be negative of the length of the super line of that little line segment right over there. So the inner product between two vectors can also be negative if the angle between them is greater than 90 degrees. So that's how vector inner products work. We're going to use these properties of vector inner product to try to understand the support vector machine optimization objective over there.



Here is the optimization objective for the support vector machine that we worked out earlier. Just for the purpose of this slide I am going to make one simplification or once just to make the objective easy to analyze and what I'm going to do is ignore the indeceptrums. So, we'll just ignore theta 0 and set that to be equal to 0. To make things easier to plot, I'm also going to set N the number of features to be equal to 2. So, we have only 2 features, X1 and X2.

Now, let's look at the objective function. The optimization objective of the SVM. What we have only two features. When N is equal to 2. This can be written, one half of theta one squared plus theta two squared. Because we only have two parameters, theta one and thetaa two. What I'm going to do is rewrite this a bit. I'm going to write this as one half of theta one squared plus theta two squared and the square root squared. And the reason I can do that, is because for any number, you know, W, right, the square roots of W and then squared, that's just equal to W. So square roots and squared should give you the same thing. What you may notice is that this term inside is that's equal to the norm or the length of the vector theta and what I mean by that is that if we write out the vector theta like this, as you know theta one, theta two. Then this term that I've just underlined in red, that's exactly the length, or the norm, of the vector theta. We are calling the definition of the norm of the vector that we have on the previous line. And in fact this is actually equal to the length of the vector theta, whether you write it as theta zero, theta 1, theta 2. That is, if theta zero is equal to zero, as I assume here. Or just the length of theta 1, theta 2; but for this line I am going to ignore theta 0. So let me just, you know, treat theta as this, let me just write theta, the normal theta as this theta 1, theta 2 only, but the math works out either way, whether we include theta zero here or not. So it's not going to matter for the rest of our derivation. And so finally this means that my optimization objective is equal to one half of the norm of theta squared. So all the support vector machine is doing in the optimization objective is it's minimizing the squared norm of the square length of the parameter vector theta. Now what I'd like to do is look at these terms, theta transpose X and understand better what they're doing. So given the parameter vector theta and given and example x, what is this is equal to? And on the previous slide, we figured out what U transpose V looks like, with different vectors U and V. And so we're going to take those definitions, you know, with theta and X(i) playing the roles of U and V. And let's see what that picture looks like. So, let's say I plot. Let's say I look at just a single training example. Let's say I have a positive example the drawing was across there and let's say that is my example X(i), what that really means is plotted on the horizontal axis some value X(i) 1 and on the vertical axis X(i) 2. That's how I plot my training examples. And although we haven't been really thinking of this as a vector, what this really is, this is a vector from the origin from 0, 0 out to the location of this training example. And now let's say we have a parameter vector and I'm going to plot that as vector, as well. What I mean by that is if I plot theta 1 here and theta 2 there so what is the inner product theta transpose X(i). While using our earlier method, the way we compute that is we take my example and project it onto my parameter vector theta. And then I'm going to look at the length of this segment that I'm coloring in, in red. And I'm going to call that P superscript I to denote that this is a projection of the i-th training example onto the parameter vector theta. And so what we have is that theta transpose X(i) is equal to following what we have on the previous slide, this is going to be equal to P times the length of the norm of the vector theta. And this is of course also equal to theta 1 x1 plus theta 2 x2. So each of these is, you know, an equally

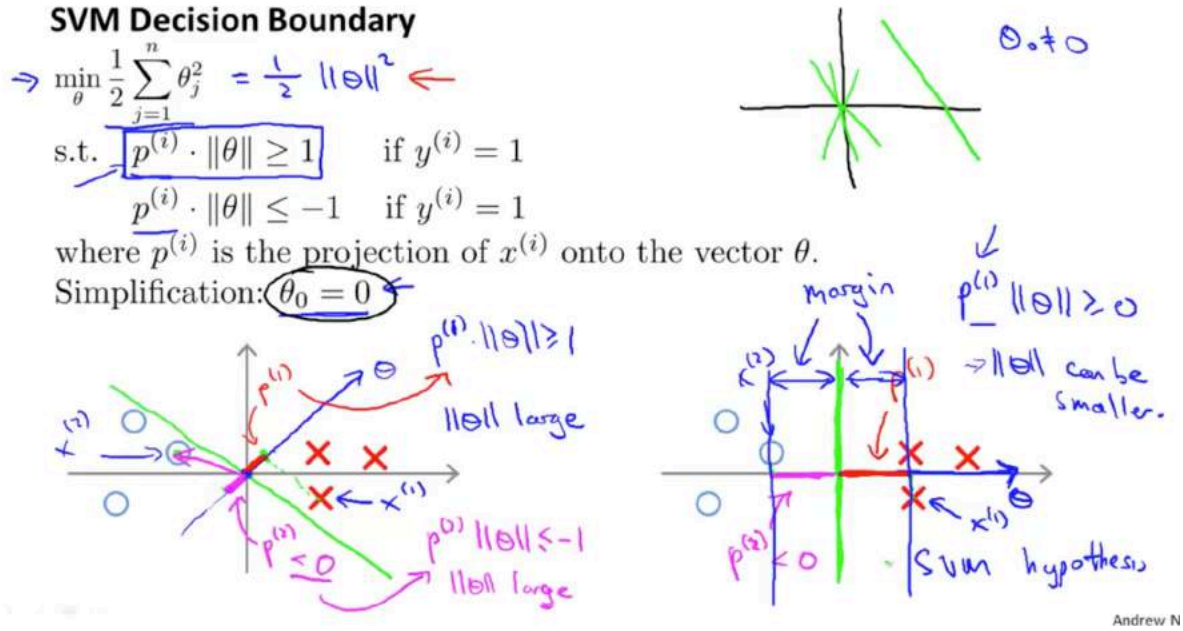
valid way of computing the inner product between theta and X(i). Okay. So where does this leave us? What this means is that, this constrains that theta transpose X(i) be greater than or equal to one or less than minus one. What this means is that it can replace the use of constraints that P(i) times X be greater than or equal to one. Because theta transpose X(i) is equal to P(i) times the norm of theta.



So writing that into our optimization objective. This is what we get where I have, instead of theta transpose X(i), I now have this P(i) times the norm of theta. And just to remind you we worked out earlier too that this optimization objective can be written as one half times the norm of theta squared. So, now let's consider the training example that we have at the bottom and for now, continuing to use the simplification that theta 0 is equal to 0. Let's see what decision boundary the support vector machine will choose. Here's one option, let's say the support vector machine were to choose this decision boundary. This is not a very good choice because it has very small margins. This decision boundary comes very close to the training examples. Let's see why the support vector machine will not do this. For this choice of parameters it's possible to show that the parameter vector theta is actually at 90 degrees to the decision boundary. And so, that green decision boundary corresponds to a parameter vector theta that points in that direction. And by the way, the simplification that theta 0 equals 0 that just means that the decision boundary must pass through the origin, (0,0) over there. So now, let's look at what this implies for the optimization objective. Let's say that this example here. Let's say that's my first example, you know, X1. If we look at the projection of this example onto my parameters theta. That's the projection. And so that little red line segment. That is equal to P1. And that is going to be pretty small, right. And similarly, if

this example here, if this happens to be X2, that's my second example. Then, if I look at the projection of this this example onto theta. You know. Then, let me draw this one in magenta. This little magenta line segment, that's going to be P2. That's the projection of the second example onto my, onto the direction of my parameter vector theta which goes like this. And so, this little projection line segment is getting pretty small. P2 will actually be a negative number, right so P2 is in the opposite direction. This vector has greater than 90 degree angle with my parameter vector theta, it's going to be less than 0. And so what we're finding is that these terms P(i) are going to be pretty small numbers. So if we look at the optimization objective and see, well, for positive examples we need P(i) times the norm of theta to be bigger than either one. But if P(i) over here, if P1 over here is pretty small, that means that we need the norm of theta to be pretty large, right? If P1 of theta is small and we want P1 you know times in all of theta to be bigger than either one, well the only way for that to be true for the profit that these two numbers to be large if P1 is small, as we said we want the norm of theta to be large. And similarly for our negative example, we need P2 times the norm of theta to be less than or equal to minus one. And we saw in this example already that P2 is going pretty small negative number, and so the only way for that to happen as well is for the norm of theta to be large, but what we are doing in the optimization objective is we are trying to find a setting of parameters where the norm of theta is small, and so you know, so this doesn't seem like such a good direction for the parameter vector and theta. In contrast, just look at a different decision boundary. Here, let's say, this SVM chooses that decision boundary. Now the is going to be very different. If that is the decision boundary, here is the corresponding direction for theta. So, with the direction boundary you know, that vertical line that corresponds to it is possible to show using linear algebra that the way to get that green decision boundary is have the vector of theta be at 90 degrees to it, and now if you look at the projection of your data onto the vector x, lets say its before this example is my example of x1. So when I project this on to x, or onto theta, what I find is that this is P1. That length there is P1. The other example, that example is and I do the same projection and what I find is that this length here is a P2 really that is going to be less than 0. And you notice that now P1 and P2, these lengths of the projections are going to be much bigger, and so if we still need to enforce these constraints that P1 of the norm of theta is phase number one because P1 is so much bigger now. The normal can be smaller. And so, what this means is that by choosing the decision boundary shown on the right instead of on the left, the SVM can make the norm of the parameters theta much smaller. So, if we can make the norm of theta smaller and therefore make the squared norm of theta smaller, which is why the SVM would choose this hypothesis on the right instead. And this is how the SVM gives rise to this large margin certification effect. Mainly, if you look at this green line, if you look at this green hypothesis we want the projections of my positive and negative examples onto theta to be large, and the only way for that to hold true this is if surrounding the green line. There's this large margin, there's this large gap that separates positive and negative examples is really the

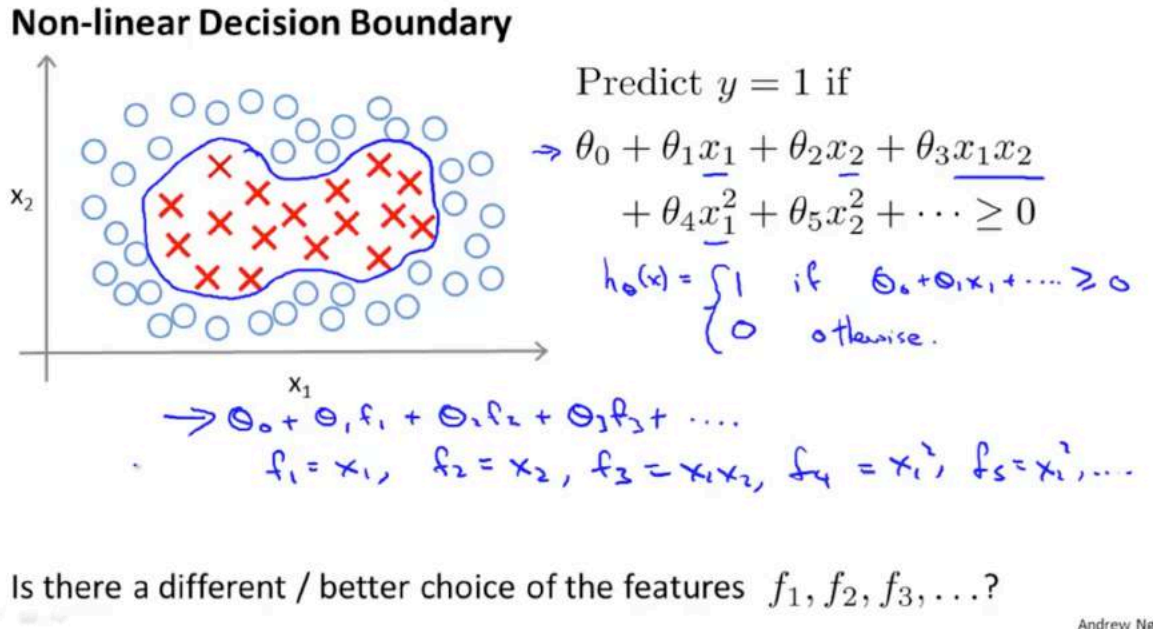
magnitude of this gap. The magnitude of this margin is exactly the values of P1, P2, P3 and so on. And so by making the margin large, by these tyros P1, P2, P3 and so on that's the SVM can end up with a smaller value for the norm of theta which is what it is trying to do in the objective. And this is why this machine ends up with enlarge margin classifiers because itss trying to maximize the norm of these P1 which is the distance from the training examples to the decision boundary. Finally, we did this whole derivation using this simplification that the parameter theta 0 must be equal to 0. The effect of that as I mentioned briefly, is that if theta 0 is equal to 0 what that means is that we are entertaining decision boundaries that pass through the origins of decision boundaries pass through the origin like that, if you allow theta zero to be non 0 then what that means is that you entertain the decision boundaries that did not cross through the origin, like that one I just drew. And I'm not going to do the full derivation that. It turns out that this same large margin proof works in pretty much in exactly the same way. And there's a generalization of this argument that we just went through them long ago through that shows that even when theta 0 is non 0, what the SVM is trying to do when you have this optimization objective. Which again corresponds to the case of when C is very large. But it is possible to show that, you know, when theta is not equal to 0 this support vector machine is still finding is really trying to find the large margin separator that between the positive and negative examples. So that explains how this support vector machine is a large margin classifier. In the next video we will start to talk about how to take some of these SVM ideas and start to apply them to build a complex nonlinear classifiers.



Kernels

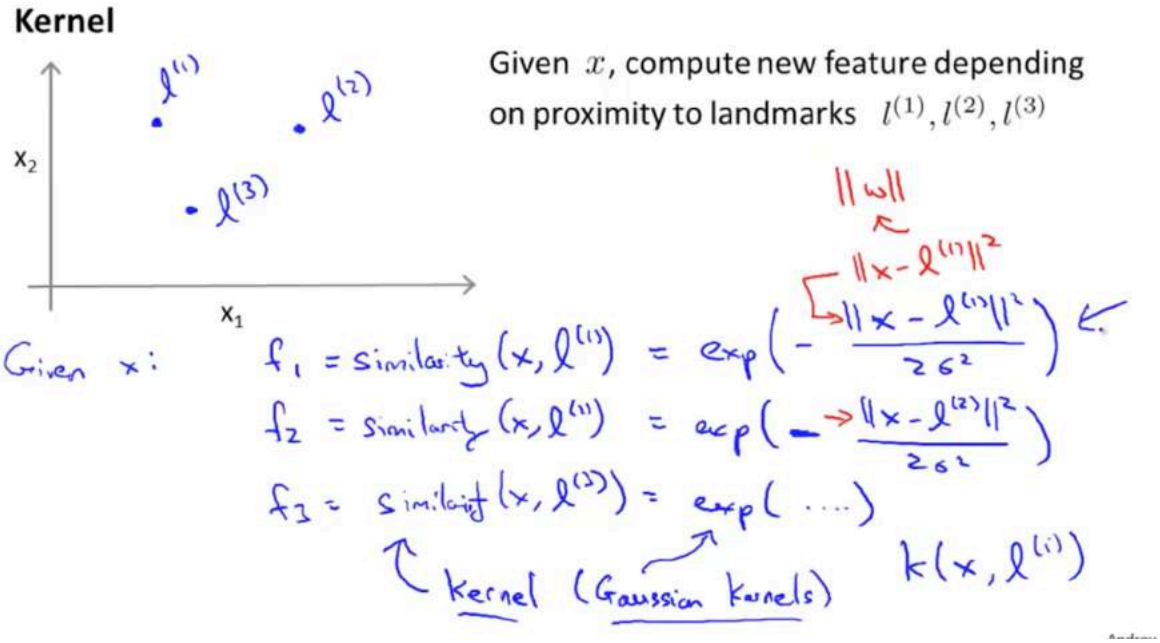
Kernels 1

In this video, I'd like to start adapting support vector machines in order to develop complex nonlinear classifiers. The main technique for doing that is something called kernels. Let's see what this kernels are and how to use them. If you have a training set that looks like this, and you want to find a nonlinear decision boundary to distinguish the positive and negative examples, maybe a decision boundary that looks like that. One way to do so is to come up with a set of complex polynomial features, right? So, set of features that looks like this, so that you end up with a hypothesis X that predicts 1 if you know that theta 0 and plus theta 1 X1 plus dot dot dot all those polynomial features is greater than 0, and predict 0, otherwise. And another way of writing this, to introduce a level of new notation that I'll use later, is that we can think of a hypothesis as computing a decision boundary using this. So, theta 0 plus theta 1 f1 plus theta 2, f2 plus theta 3, f3 plus and so on. Where I'm going to use this new denotation f1, f2, f3 and so on to denote these new sort of features that I'm computing, so f1 is just X1, f2 is equal to X2, f3 is equal to this one here. So, X1X2. So, f4 is equal to X1 squared where f5 is to be x2 squared and so on and we seen previously that coming up with these high order polynomials is one way to come up with lots more features, the question is, is there a different choice of features or is there better sort of features than this high order polynomials because you know it's not clear that this high order polynomial is what we want, and what we talked about computer vision talk about when the input is an image with lots of pixels. We also saw how using high order polynomials becomes very computationally expensive because there are a lot of these higher order polynomial terms.



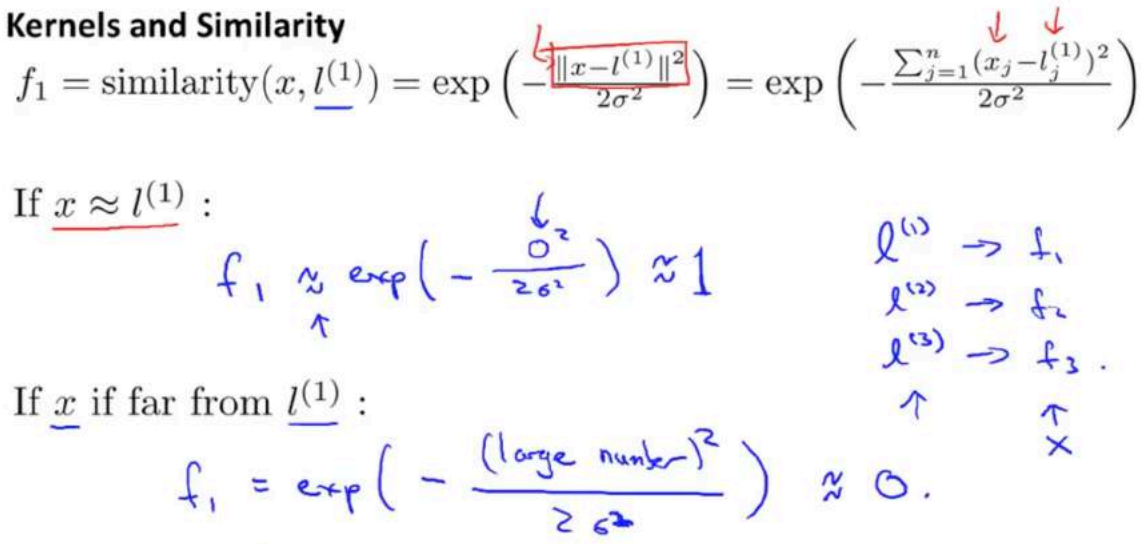
So, is there a different or a better choice of the features that we can use to plug into this sort of hypothesis form. So, here is one idea for how to define new features f1, f2, f3. On this line I am going to define only three new features, but for real problems we can get to define a much larger number. But here's what I'm going to do in this phase of features X1, X2, and I'm going to leave X0 out of this, the interceptor X0, but in this phase X1 X2, I'm going to just, you know, manually pick a few points, and then call these points l1, we are going to pick a different point, let's call that l2 and let's pick the third one and call this one l3, and for now let's just say that I'm going to choose these three points manually. I'm going to call these three points line ups, so line up one, two, three. What I'm going to do is define my new features as follows, given an example X, let me define my first feature f1 to be some measure of the similarity between my training example X and my first landmark and this specific formula that I'm going to use to measure similarity is going to be this is E to the minus the length of X minus l1, squared, divided by two sigma squared. So, depending on whether or not you watched the previous optional video, this notation, you know, this is the length of the vector W. And so, this thing here, this X minus l1, this is actually just the euclidean distance squared, is the euclidean distance between the point x and the landmark l1. We will see more about this later. But that's my first feature, and my second feature f2 is going to be, you know, similarity function that measures how similar X is to l2 and the game is going to be defined as the following function. This is E to the minus of the square of the euclidean distance between X and the second landmark, that is what the enumerator is and then divided by 2 sigma squared and similarly f3 is, you know, similarity between X and l3, which is equal to, again, similar formula. And what this similarity function is, the mathematical term for this, is that this is going to be a kernel function. And the specific kernel I'm using here,

this is actually called a Gaussian kernel. And so this formula, this particular choice of similarity function is called a Gaussian kernel. But the way the terminology goes is that, you know, in the abstract these different similarity functions are called kernels and we can have different similarity functions and the specific example I'm giving here is called the Gaussian kernel. We'll see other examples of other kernels. But for now just think of these as similarity functions. And so, instead of writing similarity between X and l, sometimes we also write this a kernel denoted you know, lower case k between x and one of my landmarks all right. So let's see what a criminals actually do and why these sorts of similarity functions, why these expressions might make sense.



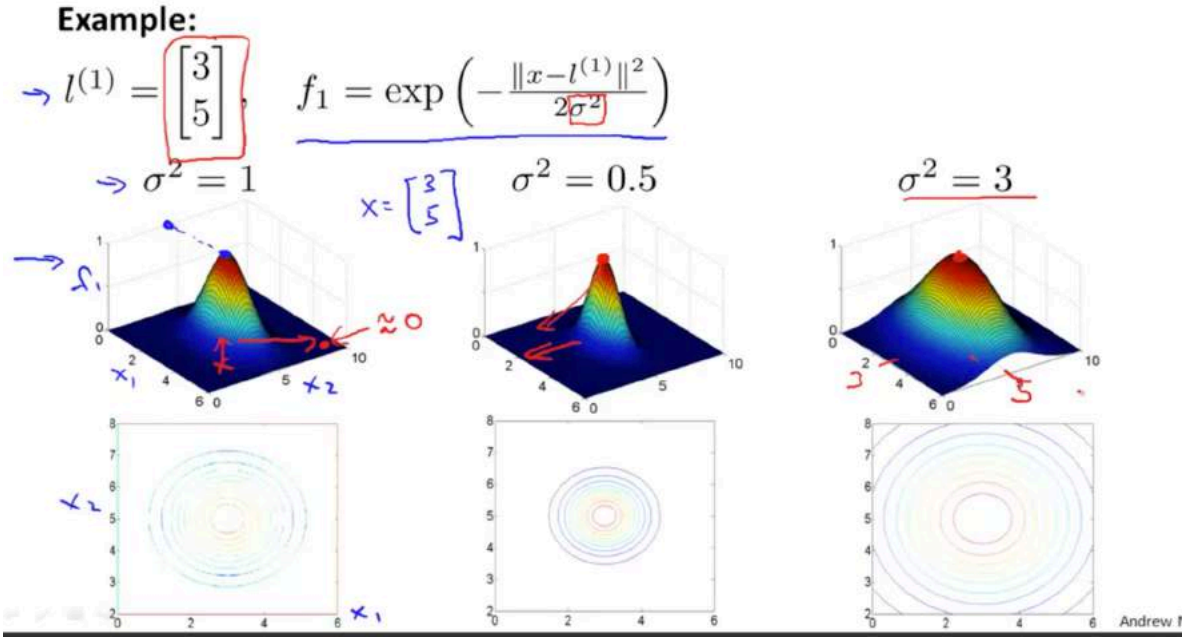
So let's take my first landmark. My landmark l1, which is one of those points I chose on my figure just now. So the similarity of the kernel between x and l1 is given by this expression. Just to make sure, you know, we are on the same page about what the numerator term is, the numerator can also be written as a sum from J equals 1 through N on sort of the distance. So this is the component wise distance between the vector X and the vector l. And again for the purpose of these slides I'm ignoring X0. So just ignoring the intercept term X0, which is always equal to 1. So, you know, this is how you compute the kernel with similarity between X and a landmark. So let's see what this function does. Suppose X is close to one of the landmarks. Then this euclidean distance formula and the numerator will be close to 0, right. So, that is this term here, the distance was great, the distance using X and 0 will be close to zero, and so f1, this is a simple feature, will be approximately E to the minus 0 and then the numerator squared over 2 is equal to squared so that E to the 0, E to minus 0, E to 0 is going to be close to one. And I'll put the approximation symbol here because the distance may not be exactly 0, but if X is closer to landmark this

term will be close to 0 and so f1 would be close 1. Conversely, if X is far from 01 then this first feature f1 will be E to the minus of some large number squared, divided divided by two sigma squared and E to the minus of a large number is going to be close to 0. So what these features do is they measure how similar X is from one of your landmarks and the feature f is going to be close to one when X is close to your landmark and is going to be 0 or close to zero when X is far from your landmark. Each of these landmarks. On the previous line, I drew three landmarks, l1, l2,l3. Each of these landmarks, defines a new feature f1, f2 and f3. That is, given the the training example X, we can now compute three new features: f1, f2, and f3, given, you know, the three landmarks that I wrote just now. But first, let's look at this exponentiation function, let's look at this similarity function and plot in some figures and just, you know, understand better what this really looks like.



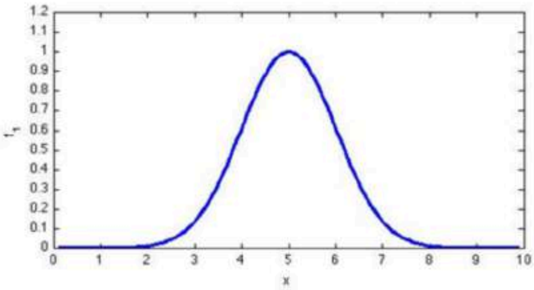
For this example, let's say I have two features X1 and X2. And let's say my first landmark, l1 is at a location, 3 5. So and let's say I set sigma squared equals one for now. If I plot what this feature looks like, what I get is this figure. So the vertical axis, the height of the surface is the value of f1 and down here on the horizontal axis are, if I have some training example, and there is x1 and there is x2. Given a certain training example, the training example here which shows the value of x1 and x2 at a height above the surface, shows the corresponding value of f1 and down below this is the same figure I had showed, using a quantifiable plot, with x1 on horizontal axis, x2 on horizontal axis and so, this figure on the bottom is just a contour plot of the 3D surface. You notice that when X is equal to 3 5 exactly, then we the f1 takes on the value 1, because that's at the maximum and X moves away as X goes further away then this feature takes on values that are close to 0. And so, this is really a feature, f1 measures, you know, how close X is to the first landmark and if varies between 0 and one depending on how close X is to the first landmark l1. Now the other

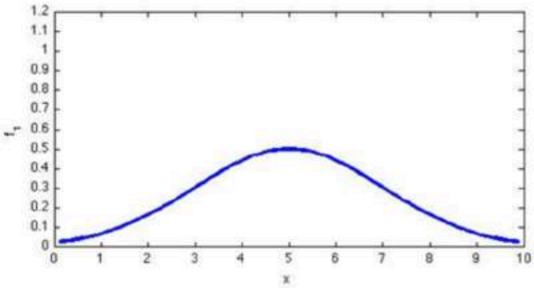
was due on this slide is show the effects of varying this parameter sigma squared. So, sigma squared is the parameter of the Gaussian kernel and as you vary it, you get slightly different effects. Let's set sigma squared to be equal to 0.5 and see what we get. We set sigma square to 0.5, what you find is that the kernel looks similar, except for the width of the bump becomes narrower. The contours shrink a bit too. So if sigma squared equals to 0.5 then as you start from X equals 3 5 and as you move away, then the feature f1 falls to zero much more rapidly and conversely, if you has increase since where three in that case and as I move away from, you know I. So this point here is really I, right, that's I1 is at location 3 5, right. So it's shown up here. And if sigma squared is large, then as you move away from I1, the value of the feature falls away much more slowly.

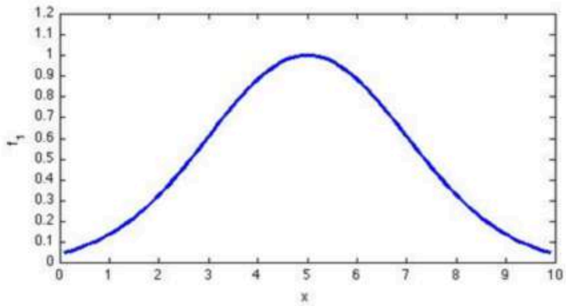


Question

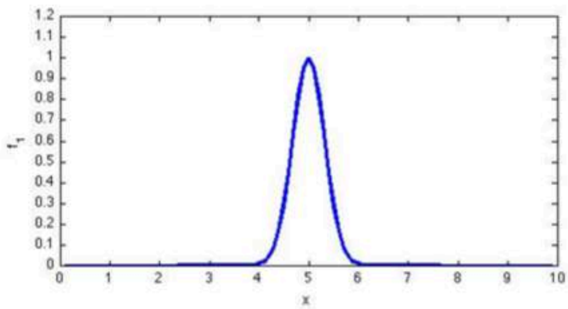
Consider a 1-D example with one feature x_1. Suppose $l^{(1)} = 5$. To the right is a plot of $f_1 = \exp(-\frac{\|x_1 - l^{(1)}\|}{2\sigma^2})$ when $\sigma^2 = 1$. Suppose we now change $\sigma^2 = 4$. Which of the following is a plot of f_1 with the new value of σ^2?





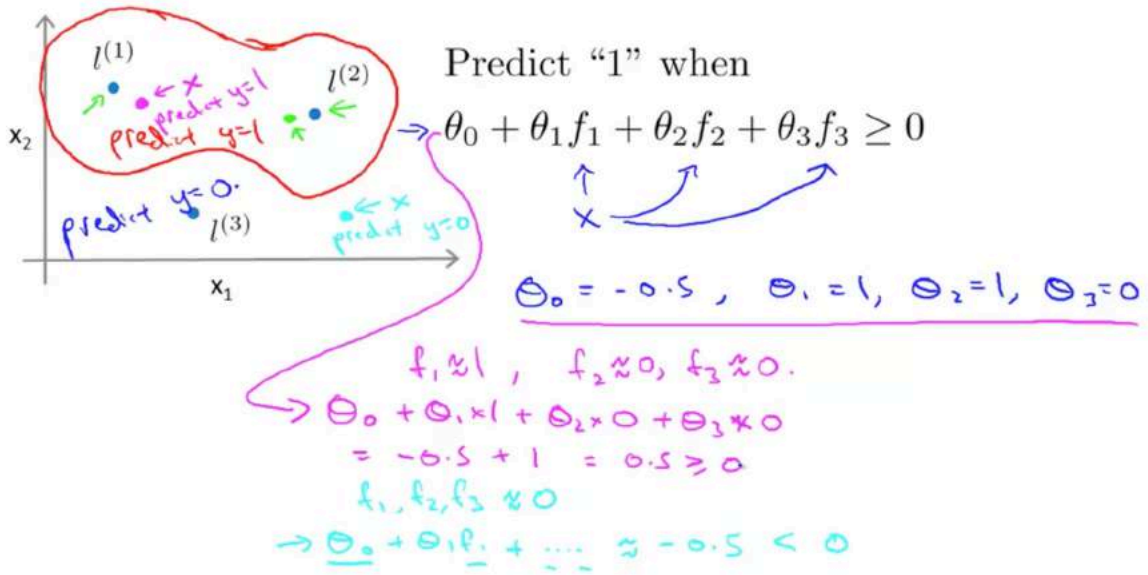


Correct



So, given this definition of the features, let's see what source of hypothesis we can learn. Given the training example X, we are going to compute these features f1, f2, f3 and a hypothesis is going to predict one when theta 0 plus theta 1 f1 plus theta 2 f2, and so on is greater than or equal to 0. For this

particular example, let's say that I've already found a learning algorithm and let's say that, you know, somehow I ended up with these values of the parameter. So if theta 0 equals minus 0.5, theta 1 equals 1, theta 2 equals 1, and theta 3 equals 0 And what I want to do is consider what happens if we have a training example that takes has location at this magenta dot, right where I just drew this dot over here. So let's say I have a training example X, what would my hypothesis predict? Well, If I look at this formula. Because my training example X is close to l1, we have that f1 is going to be close to 1 the because my training example X is far from l2 and l3 I have that, you know, f2 would be close to 0 and f3 will be close to 0. So, if I look at that formula, I have theta 0 plus theta 1 times 1 plus theta 2 times some value. Not exactly 0, but let's say close to 0. Then plus theta 3 times something close to 0. And this is going to be equal to plugging in these values now. So, that gives minus 0.5 plus 1 times 1 which is 1, and so on. Which is equal to 0.5 which is greater than or equal to 0. So, at this point, we're going to predict Y equals 1, because that's greater than or equal to zero. Now let's take a different point. Now lets' say I take a different point, I'm going to draw this one in a different color, in cyan say, for a point out there, if that were my training example X, then if you make a similar computation, you find that f1, f2, Ff3 are all going to be close to 0. And so, we have theta 0 plus theta 1, f1, plus so on and this will be about equal to minus 0.5, because theta 0 is minus 0.5 and f1, f2, f3 are all zero. So this will be minus 0.5, this is less than zero. And so, at this point out there, we're going to predict Y equals zero. And if you do this yourself for a range of different points, be sure to convince yourself that if you have a training example that's close to L2, say, then at this point we'll also predict Y equals one. And in fact, what you end up doing is, you know, if you look around this boundary, this space, what we'll find is that for points near l1 and l2 we end up predicting positive. And for points far away from l1 and l2, that's for points far away from these two landmarks, we end up predicting that the class is equal to 0. As so, what we end up doing,is that the decision boundary of this hypothesis would end up looking something like this where inside this red decision boundary would predict Y equals 1 and outside we predict Y equals 0. And so this is how with this definition of the landmarks and of the kernel function. We can learn pretty complex non-linear decision boundary, like what I just drew where we predict positive when we're close to either one of the two landmarks. And we predict negative when we're very far away from any of the landmarks. And so this is part of the idea of kernels of and how we use them with the support vector machine, which is that we define these extra features using landmarks and similarity functions to learn more complex nonlinear classifiers.

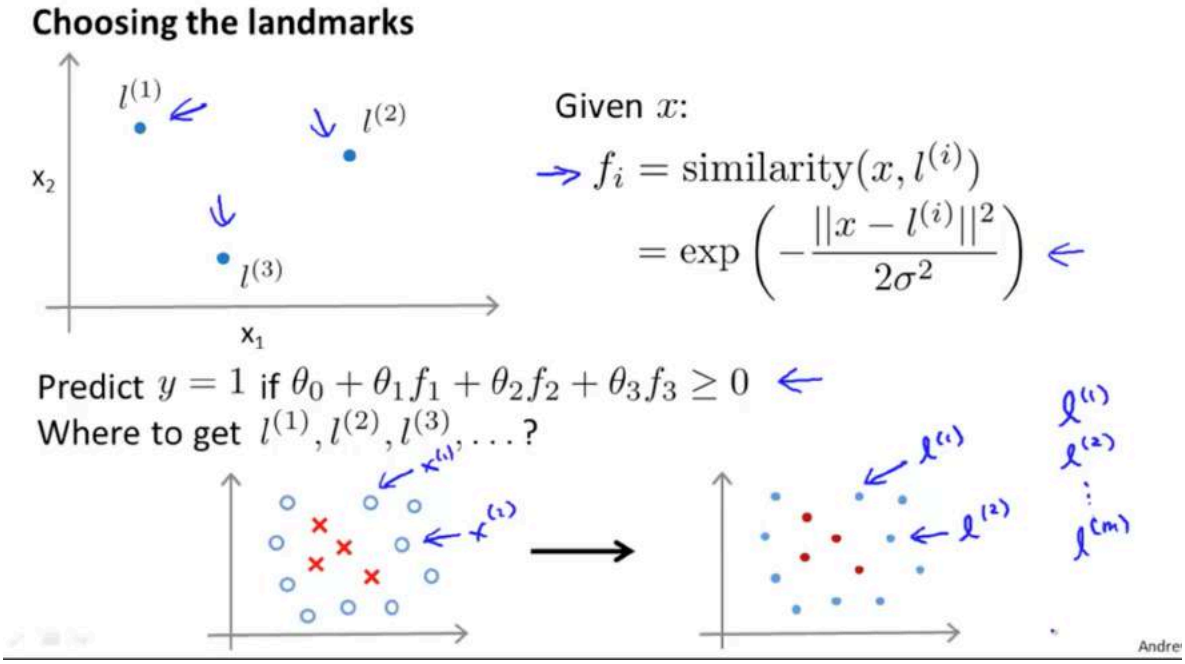


So hopefully that gives you a sense of the idea of kernels and how we could use it to define new features for the Support Vector Machine. But there are a couple of questions that we haven't answered yet. One is, how do we get these landmarks? How do we choose these landmarks? And another is, what other similarity functions, if any, can we use other than the one we talked about, which is called the Gaussian kernel. In the next video we give answers to these questions and put everything together to show how support vector machines with kernels can be a powerful way to learn complex nonlinear functions.

Kernels 2

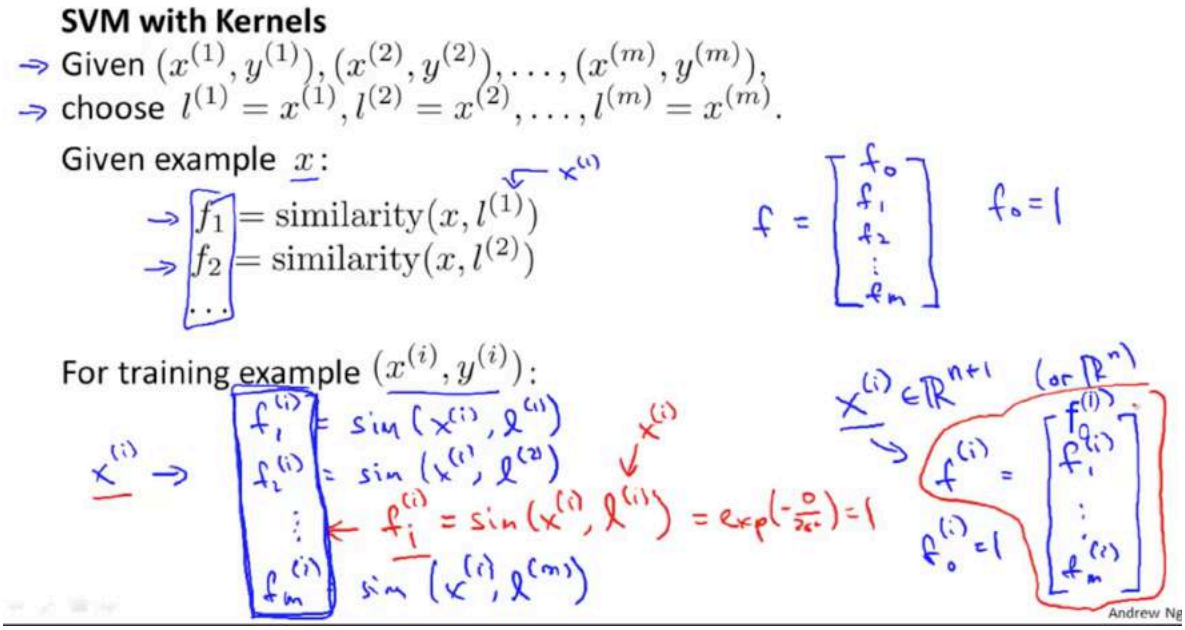
In the last video, we started to talk about the kernels idea and how it can be used to define new features for the support vector machine. In this video, I'd like to throw in some of the missing details and, also, say a few words about how to use these ideas in practice. Such as, how they pertain to, for example, the bias variance trade-off in support vector machines. In the last video, I talked about the process of picking a few landmarks. You know, l1, l2, l3 and that allowed us to define the similarity function also called the kernel or in this example if you have this similarity function this is a Gaussian kernel. And that allowed us to build this form of a hypothesis function. But where do we get these landmarks from? Where do we get l1, l2, l3 from? And it seems, also, that

for complex learning problems, maybe we want a lot more landmarks than just three of them that we might choose by hand. So in practice this is how the landmarks are chosen which is that given the machine learning problem. We have some data set of some some positive and negative examples. So, this is the idea here which is that we're gonna take the examples and for every training example that we have, we are just going to call it. We're just going to put landmarks as exactly the same locations as the training examples. So if I have one training example if that is x1, well then I'm going to choose this is my first landmark to be at exactly the same location as my first training example. And if I have a different training example x2. Well we're going to set the second landmark to be the location of my second training example. On the figure on the right, I used red and blue dots just as illustration, the color of this figure, the color of the dots on the figure on the right is not significant. But what I'm going to end up with using this method is I'm going to end up with m landmarks of l1, l2 down to l(m) if I have m training examples with one landmark per location of my per location of each of my training examples. And this is nice because it is saying that my features are basically going to measure how close an example is to one of the things I saw in my training set.



So, just to write this outline a little more concretely, given m training examples, I'm going to choose the the location of my landmarks to be exactly near the locations of my m training examples. When you are given example x, and in this example x can be something in the training set, it can be something in the cross validation set, or it can be something in the test set. Given an example x we are going to compute, you know, these features as so f1, f2, and so on. Where l1 is actually equal to x1 and so on. And these then give me a feature vector. So let me write f as the feature vector. I'm going to take these f1, f2 and

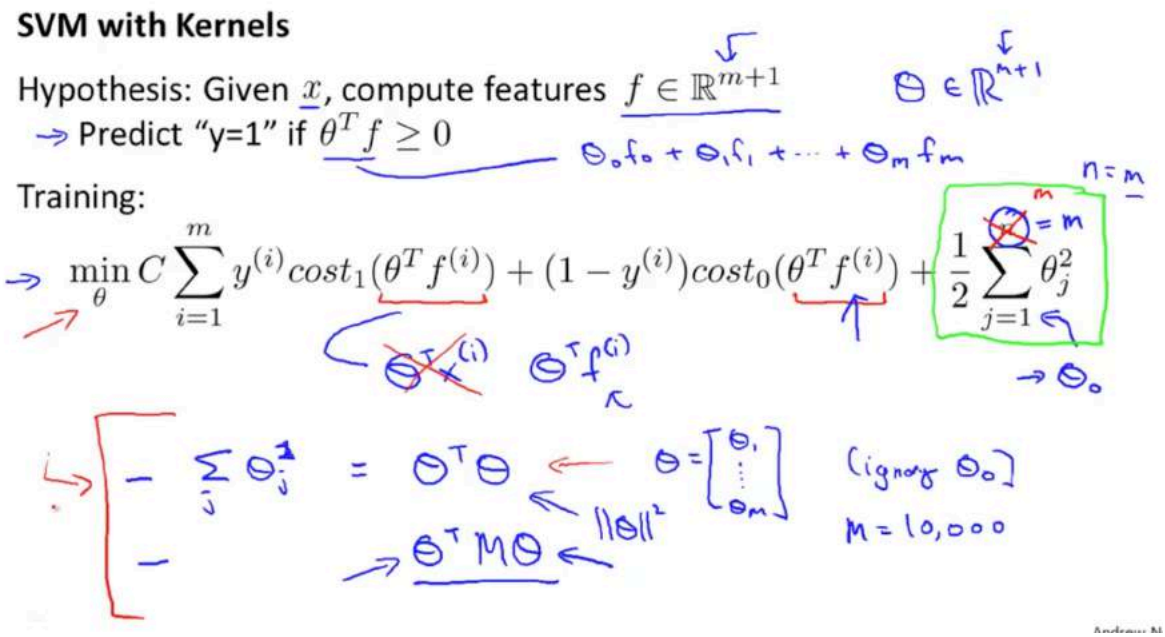
so on, and just group them into feature vector. Take those down to fm. And, you know, just by convention. If we want, we can add an extra feature f0, which is always equal to 1. So this plays a role similar to what we had previously. For x0, which was our interceptor. So, for example, if we have a training example x(i), y(i), the features we would compute for this training example will be as follows: given x(i), we will then map it to, you know, f1(i). Which is the similarity. I'm going to abbreviate as SIM instead of writing out the whole word similarity, right? And f2(i) equals the similarity between x(i) and l2, and so on, down to fm(i) equals the similarity between x(i) and l(m). And somewhere in the middle. Somewhere in this list, you know, at the i-th component, I will actually have one feature component which is f subscript i(i), which is going to be the similarity between x and l(Where l(i) is equal to x(i), and so you know fi(i) is just going to be the similarity between x and itself. And if you're using the Gaussian kernel this is actually e to the minus 0 over 2 sigma squared and so, this will be equal to 1 and that's okay. So one of my features for this training example is going to be equal to 1. And then similar to what I have above. I can take all of these m features and group them into a feature vector. So instead of representing my example, using, you know, x(i) which is this what R(n) plus R(n) one dimensional vector. Depending on whether you can set terms, is either R(n) or R(n) plus 1. We can now instead represent my training example using this feature vector f. I am going to write this f superscript i. Which is going to be taking all of these things and stacking them into a vector. So, f1(i) down to fm(i) and if you want and well, usually we'll also add this f0(i), where f0(i) is equal to 1. And so this vector here gives me my new feature vector with which to represent my training example.



So given these kernels and similarity functions, here's how we use a simple vector machine. If you already have a learning set of parameters theta, then if

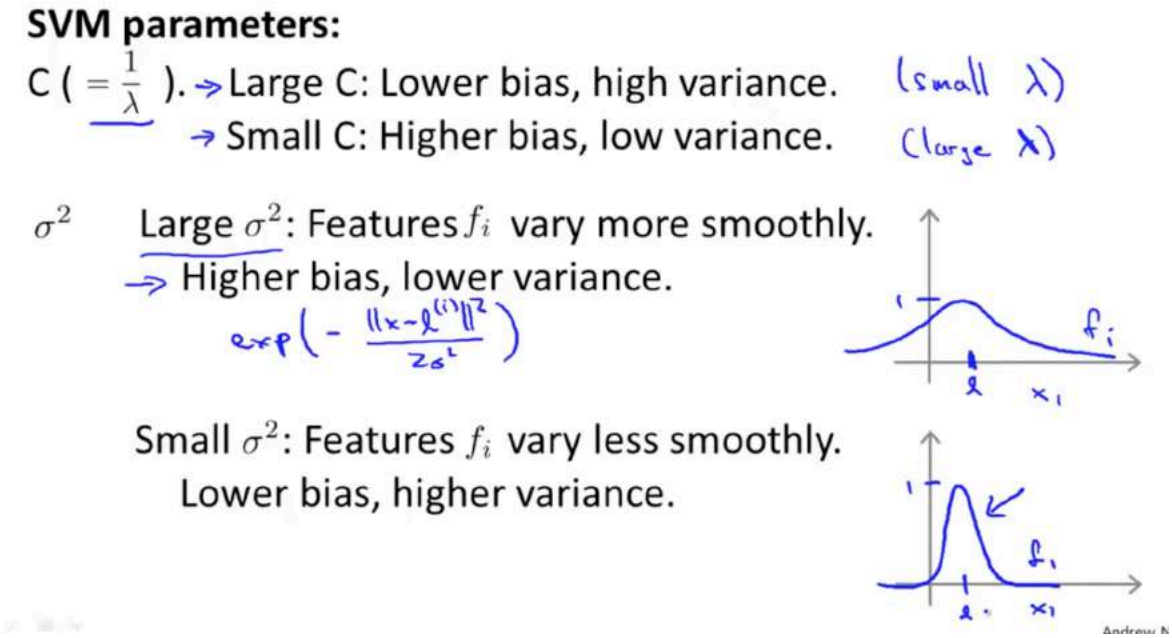
you given a value of x and you want to make a prediction. What we do is we compute the features f, which is now an R(m) plus 1 dimensional feature vector. And we have m here because we have m training examples and thus m landmarks and what we do is we predict 1 if theta transpose f is greater than or equal to 0. Right. So, if theta transpose f, of course, that's just equal to theta 0, f0 plus theta 1, f1 plus dot dot dot, plus theta m f(m). And so my parameter vector theta is also now going to be an m plus 1 dimensional vector. And we have m here because where the number of landmarks is equal to the training set size. So m was the training set size and now, the parameter vector theta is going to be m plus one dimensional. So that's how you make a prediction if you already have a setting for the parameter's theta. How do you get the parameter's theta? Well you do that using the SVM learning algorithm, and specifically what you do is you would solve this minimization problem. You've minimized the parameter's theta of C times this cost function which we had before. Only now, instead of looking there instead of making predictions using theta transpose x(i) using our original features, x(i). Instead we've taken the features x(i) and replace them with a new features so we are using theta transpose f(i) to make a prediction on the i'f training examples and we see that, you know, in both places here and it's by solving this minimization problem that you get the parameters for your Support Vector Machine. And one last detail is because this optimization problem we really have n equals m features. That is here. The number of features we have. Really, the effective number of features we have is dimension of f. So that n is actually going to be equal to m. So, if you want to, you can think of this as a sum, this really is a sum from j equals 1 through m. And then one way to think about this, is you can think of it as n being equal to m, because if f isn't a new feature, then we have m plus 1 features, with the plus 1 coming from the interceptor. And here, we still do sum from j equal 1 through n, because similar to our earlier videos on regularization, we still do not regularize the parameter theta zero, which is why this is a sum for j equals 1 through m instead of j equals zero though m. So that's the support vector machine learning algorithm. That's one sort of, mathematical detail aside that I should mention, which is that in the way the support vector machine is implemented, this last term is actually done a little bit differently. So you don't really need to know about this last detail in order to use support vector machines, and in fact the equations that are written down here should give you all the intuitions that should need. But in the way the support vector machine is implemented, you know, that term, the sum of j of theta j squared right? Another way to write this is this can be written as theta transpose theta if we ignore the parameter theta 0. So theta 1 down to theta m. Ignoring theta 0. Then this sum of j of theta j squared that this can also be written theta transpose theta. And what most support vector machine implementations do is actually replace this theta transpose theta, will instead, theta transpose times some matrix inside, that depends on the kernel you use, times theta. And so this gives us a slightly different distance metric. We'll use a slightly different measure instead of minimizing exactly the norm of theta squared means that minimize something slightly similar to it. That's like a

rescale version of the parameter vector theta that depends on the kernel. But this is kind of a mathematical detail. That allows the support vector machine software to run much more efficiently. And the reason the support vector machine does this is with this modification. It allows it to scale to much bigger training sets. Because for example, if you have a training set with 10,000 training examples. Then, you know, the way we define landmarks, we end up with 10,000 landmarks. And so theta becomes 10,000 dimensional. And maybe that works, but when m becomes really, really big then solving for all of these parameters, you know, if m were 50,000 or a 100,000 then solving for all of these parameters can become expensive for the support vector machine optimization software, thus solving the minimization problem that I drew here. So kind of as mathematical detail, which again you really don't need to know about. It actually modifies that last term a little bit to optimize something slightly different than just minimizing the norm squared of theta squared, of theta. But if you want, you can feel free to think of this as an kind of a n implementational detail that does change the objective a bit, but is done primarily for reasons of computational efficiency, so usually you don't really have to worry about this. And by the way, in case your wondering why we don't apply the kernel's idea to other algorithms as well like logistic regression, it turns out that if you want, you can actually apply the kernel's idea and define the source of features using landmarks and so on for logistic regression. But the computational tricks that apply for support vector machines don't generalize well to other algorithms like logistic regression. And so, using kernels with logistic regression is going too very slow, whereas, because of computational tricks, like that embodied and how it modifies this and the details of how the support vector machine software is implemented, support vector machines and kernels tend go particularly well together. Whereas, logistic regression and kernels, you know, you can do it, but this would run very slowly. And it won't be able to take advantage of advanced optimization techniques that people have figured out for the particular case of running a support vector machine with a kernel. But all this pertains only to how you actually implement software to minimize the cost function. I will say more about that in the next video, but you really don't need to know about how to write software to minimize this cost function because you can find very good off the shelf software for doing so. And just as, you know, I wouldn't recommend writing code to invert a matrix or to compute a square root, I actually do not recommend writing software to minimize this cost function yourself, but instead to use off the shelf software packages that people have developed and so those software packages already embody these numerical optimization tricks, so you don't really have to worry about them.

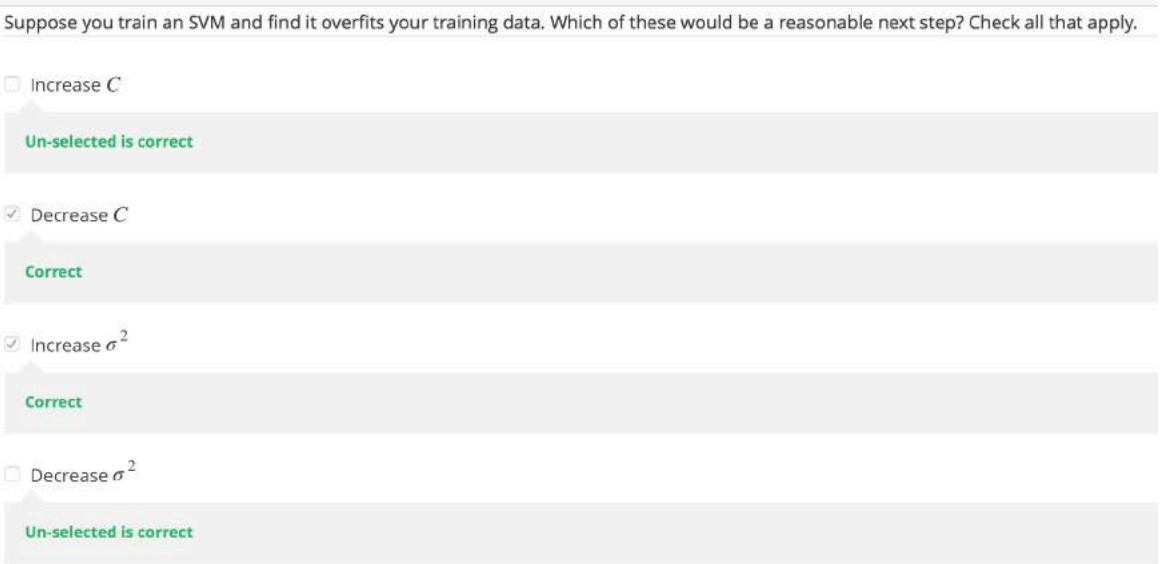


But one other thing that is worth knowing about is when you're applying a support vector machine, how do you choose the parameters of the support vector machine? And the last thing I want to do in this video is say a little word about the bias and variance trade offs when using a support vector machine. When using an SVM, one of the things you need to choose is the parameter C which was in the optimization objective, and you recall that C played a role similar to 1 over lambda, where lambda was the regularization parameter we had for logistic regression. So, if you have a large value of C, this corresponds to what we have back in logistic regression, of a small value of lambda meaning of not using much regularization. And if you do that, you tend to have a hypothesis with lower bias and higher variance. Whereas if you use a smaller value of C then this corresponds to when we are using logistic regression with a large value of lambda and that corresponds to a hypothesis with higher bias and lower variance. And so, hypothesis with large C has a higher variance, and is more prone to overfitting, whereas hypothesis with small C has higher bias and is thus more prone to underfitting. So this parameter C is one of the parameters we need to choose. The other one is the parameter sigma squared, which appeared in the Gaussian kernel. So if the Gaussian kernel sigma squared is large, then in the similarity function, which was this you know E to the minus x minus landmark varies squared over 2 sigma squared. In this one of the example; If I have only one feature, x1, if I have a landmark there at that location, if sigma squared is large, then, you know, the Gaussian kernel would tend to fall off relatively slowly and so this would be my feature f(i), and so this would be smoother function that varies more smoothly, and so this will give you a hypothesis with higher bias and lower variance, because the Gaussian kernel that falls off smoothly, you tend to get a hypothesis that varies slowly, or varies

smoothly as you change the input x. Whereas in contrast, if sigma squared was small and if that's my landmark given my 1 feature x1, you know, my Gaussian kernel, my similarity function, will vary more abruptly. And in both cases I'd pick out 1, and so if sigma squared is small, then my features vary less smoothly. So if it's just higher slopes or higher derivatives here. And using this, you end up fitting hypotheses of lower bias and you can have higher variance. And if you look at this week's points exercise, you actually get to play around with some of these ideas yourself and see these effects yourself.



Question



So, that was the support vector machine with kernels algorithm. And hopefully

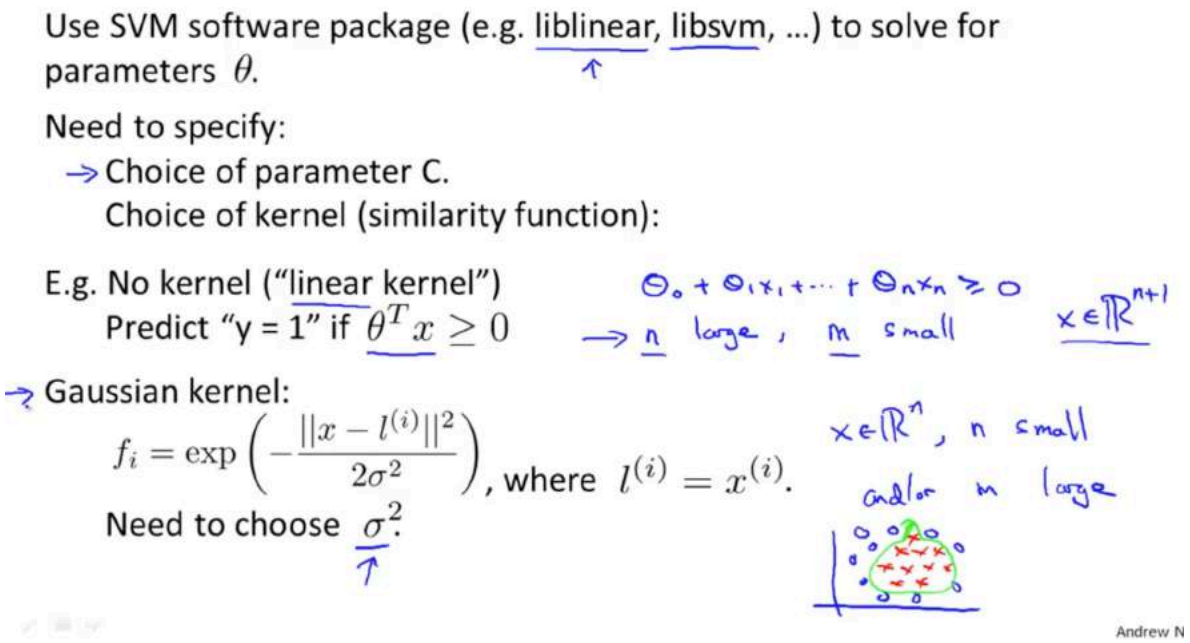
this discussion of bias and variance will give you some sense of how you can expect this algorithm to behave as well.

SVMs in Practice

Using An SVM

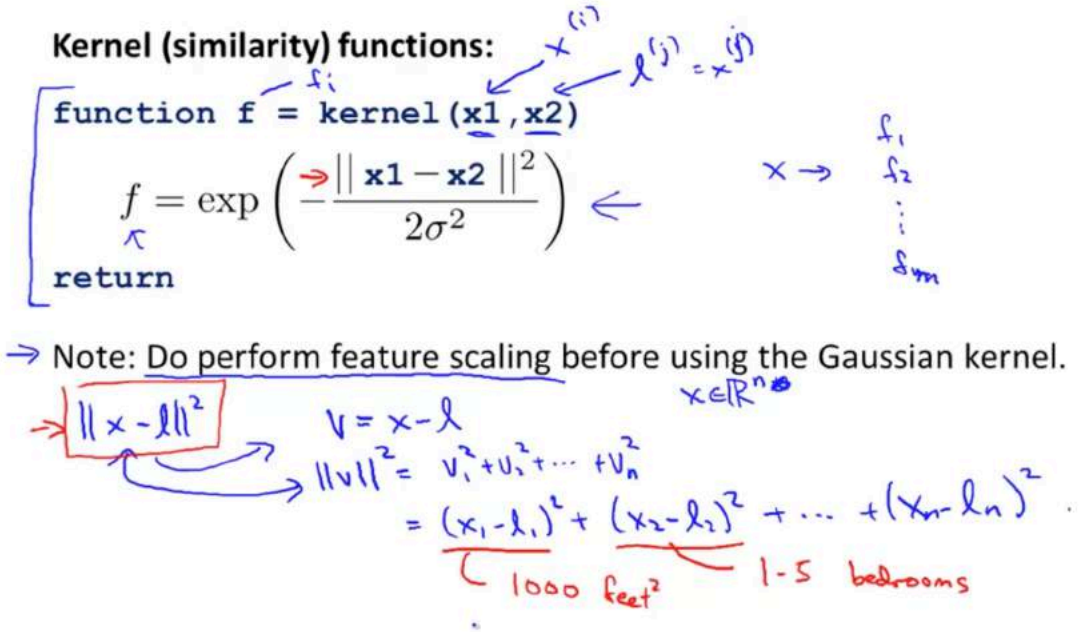
So far we've been talking about SVMs in a fairly abstract level. In this video I'd like to talk about what you actually need to do in order to run or to use an SVM. The support vector machine algorithm poses a particular optimization problem. But as I briefly mentioned in an earlier video, I really do not recommend writing your own software to solve for the parameter's theta yourself. So just as today, very few of us, or maybe almost essentially none of us would think of writing code ourselves to invert a matrix or take a square root of a number, and so on. We just, you know, call some library function to do that. In the same way, the software for solving the SVM optimization problem is very complex, and there have been researchers that have been doing essentially numerical optimization research for many years. So you come up with good software libraries and good software packages to do this. And then strongly recommend just using one of the highly optimized software libraries rather than trying to implement something yourself. And there are lots of good software libraries out there. The two that I happen to use the most often are the linear SVM but there are really lots of good software libraries for doing this that you know, you can link to many of the major programming languages that you may be using to code up learning algorithm. Even though you shouldn't be writing your own SVM optimization software, there are a few things you need to do, though. First is to come up with with some choice of the parameter's C. We talked a little bit of the bias/variance properties of this in the earlier video. Second, you also need to choose the kernel or the similarity function that you want to use. So one choice might be if we decide not to use any kernel. And the idea of no kernel is also called a linear kernel. So if someone says, I use an SVM with a linear kernel, what that means is you know, they use an SVM without using without using a kernel and it was a version of the SVM that just uses theta transpose X, right, that predicts 1 theta 0 plus theta 1 X1 plus so on

plus theta N, X N is greater than equals 0. This term linear kernel, you can think of this as you know this is the version of the SVM that just gives you a standard linear classifier. So that would be one reasonable choice for some problems, and you know, there would be many software libraries, like linear, was one example, out of many, one example of a software library that can train an SVM without using a kernel, also called a linear kernel. So, why would you want to do this? If you have a large number of features, if N is large, and M the number of training examples is small, then you know you have a huge number of features that if X, this is an X is an Rn, Rn +1. So if you have a huge number of features already, with a small training set, you know, maybe you want to just fit a linear decision boundary and not try to fit a very complicated nonlinear function, because might not have enough data. And you might risk overfitting, if you're trying to fit a very complicated function in a very high dimensional feature space, but if your training set sample is small. So this would be one reasonable setting where you might decide to just not use a kernel, or equivalents to use what's called a linear kernel. A second choice for the kernel that you might make, is this Gaussian kernel, and this is what we had previously. And if you do this, then the other choice you need to make is to choose this parameter sigma squared when we also talk a little bit about the bias variance tradeoffs of how, if sigma squared is large, then you tend to have a higher bias, lower variance classifier, but if sigma squared is small, then you have a higher variance, lower bias classifier. So when would you choose a Gaussian kernel? Well, if your omission of features X, I mean Rn, and if N is small, and, ideally, you know, if n is large, right, so that's if, you know, we have say, a two-dimensional training set, like the example I drew earlier. So n is equal to 2, but we have a pretty large training set. So, you know, I've drawn in a fairly large number of training examples, then maybe you want to use a kernel to fit a more complex nonlinear decision boundary, and the Gaussian kernel would be a fine way to do this. I'll say more towards the end of the video, a little bit more about when you might choose a linear kernel, a Gaussian kernel and so on. But if concretely, if you decide to use a Gaussian kernel, then here's what you need to do.



Depending on what support vector machine software package you use, it may ask you to implement a kernel function, or to implement the similarity function. So if you're using an octave or MATLAB implementation of an SVM, it may ask you to provide a function to compute a particular feature of the kernel. So this is really computing f subscript i for one particular value of i, where f here is just a single real number, so maybe I should move this better written f(i), but what you need to do is to write a kernel function that takes this input, you know, a training example or a test example whatever it takes in some vector X and takes as input one of the landmarks and but only I've come down X1 and X2 here, because the landmarks are really training examples as well. But what you need to do is write software that takes this input, you know, X1, X2 and computes this sort of similarity function between them and return a real number. And so what some support vector machine packages do is expect you to provide this kernel function that take this input you know, X1, X2 and returns a real number. And then it will take it from there and it will automatically generate all the features, and so automatically take X and map it to f1, f2, down to f(m) using this function that you write, and generate all the features and train the support vector machine from there. But sometimes you do need to provide this function yourself. Other if you are using the Gaussian kernel, some SVM implementations will also include the Gaussian kernel and a few other kernels as well, since the Gaussian kernel is probably the most common kernel. Gaussian and linear kernels are really the two most popular kernels by far. Just one implementational note. If you have features of very different scales, it is important to perform feature scaling before using the Gaussian kernel. And here's why. If you imagine the computing the norm between X and l, right, so this term here, and the numerator term over there. What this is

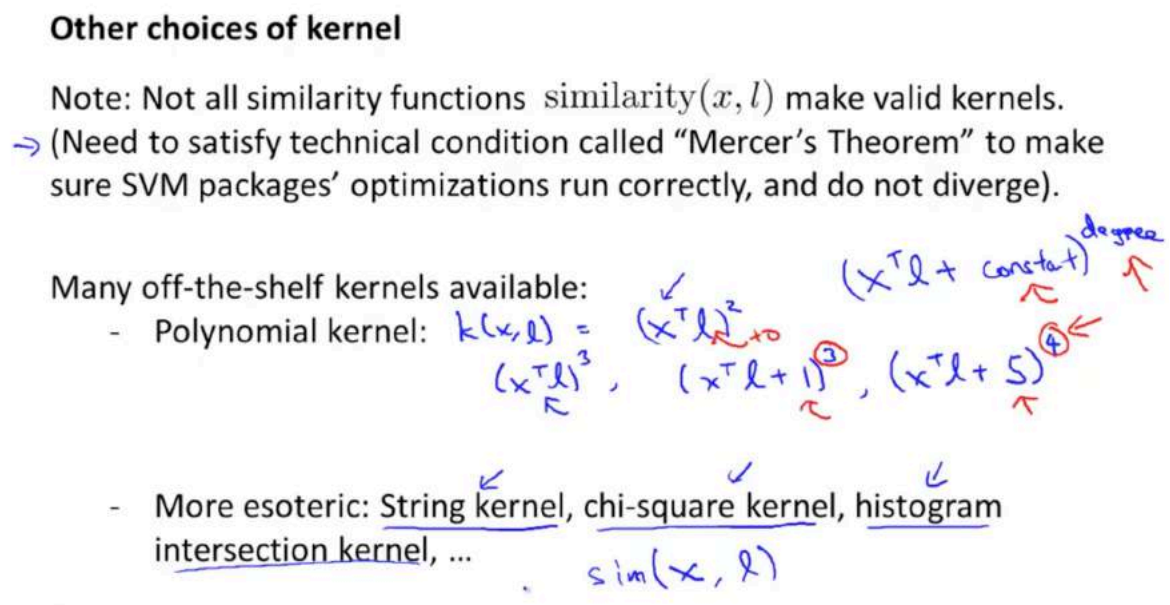
doing, the norm between X and l, that's really saying, you know, let's compute the vector V, which is equal to X minus l. And then let's compute the norm does vector V, which is the difference between X. So the norm of V is really equal to V1 squared plus V2 squared plus dot dot dot, plus Vn squared. Because here X is in Rn, or Rn plus 1, but I'm going to ignore, you know, X0. So, let's pretend X is an Rn, square on the left side is what makes this correct. So this is equal to that, right? And so written differently, this is going to be X1 minus l1 squared, plus x2 minus l2 squared, plus dot dot dot plus Xn minus ln squared. And now if your features take on very different ranges of value. So take a housing prediction, for example, if your data is some data about houses. And if X is in the range of thousands of square feet, for the first feature, X1. But if your second feature, X2 is the number of bedrooms. So if this is in the range of one to five bedrooms, then X1 minus l1 is going to be huge. This could be like a thousand squared, whereas X2 minus l2 is going to be much smaller and if that's the case, then in this term, those distances will be almost essentially dominated by the sizes of the houses and the number of bathrooms would be largely ignored. As so as, to avoid this in order to make a machine work well, do perform future scaling. And that will sure that the SVM gives, you know, comparable amount of attention to all of your different features, and not just to in this example to size of houses were big movement here the features.



When you try a support vector machines chances are by far the two most common kernels you use will be the linear kernel, meaning no kernel, or the Gaussian kernel that we talked about. And just one note of warning which is that not all similarity functions you might come up with are valid kernels. And the Gaussian kernel and the linear kernel and other kernels that you sometimes others will use, all of them need to satisfy a technical condition. It's called Mercer's Theorem and the reason you need to this is because support vector

machine algorithms or implementations of the SVM have lots of clever numerical optimization tricks. In order to solve for the parameter's theta efficiently and in the original design envisaged, those are decision made to restrict our attention only to kernels that satisfy this technical condition called Mercer's Theorem. And what that does is, that makes sure that all of these SVM packages, all of these SVM software packages can use the large class of optimizations and get the parameter theta very quickly. So, what most people end up doing is using either the linear or Gaussian kernel, but there are a few other kernels that also satisfy Mercer's theorem and that you may run across other people using, although I personally end up using other kernels you know, very, very rarely, if at all. Just to mention some of the other kernels that you may run across. One is the polynomial kernel. And for that the similarity between X and l is defined as, there are a lot of options, you can take X transpose l squared. So, here's one measure of how similar X and l are. If X and l are very close with each other, then the inner product will tend to be large. And so, you know, this is a slightly unusual kernel. That is not used that often, but you may run across some people using it. This is one version of a polynomial kernel. Another is X transpose l cubed. These are all examples of the polynomial kernel. X transpose l plus 1 cubed. X transpose l plus maybe a number different then one 5 and, you know, to the power of 4 and so the polynomial kernel actually has two parameters. One is, what number do you add over here? It could be 0. This is really plus 0 over there, as well as what's the degree of the polynomial over there. So the degree power and these numbers. And the more general form of the polynomial kernel is X transpose l, plus some constant and then to some degree in the X1 and so both of these are parameters for the polynomial kernel. So the polynomial kernel almost always or usually performs worse. And the Gaussian kernel does not use that much, but this is just something that you may run across. Usually it is used only for data where X and l are all strictly non negative, and so that ensures that these inner products are never negative. And this captures the intuition that X and l are very similar to each other, then maybe the inter product between them will be large. They have some other properties as well but people tend not to use it much. And then, depending on what you're doing, there are other, sort of more esoteric kernels as well, that you may come across. You know, there's a string kernel, this is sometimes used if your input data is text strings or other types of strings. There are things like the chi-square kernel, the histogram intersection kernel, and so on. There are sort of more esoteric kernels that you can use to measure similarity between different objects. So for example, if you're trying to do some sort of text classification problem, where the input x is a string then maybe we want to find the similarity between two strings using the string kernel, but I personally you know end up very rarely, if at all, using these more esoteric kernels. I think I might have use the chi-square kernel, may be once in my life and the histogram kernel, may be once or twice in my life. I've actually never used the string kernel myself. But in case you've run across this in other applications. You know, if you do a quick web search we do a quick Google search or quick Bing search you should have

found definitions that these are the kernels as well.



Question

Suppose you are trying to decide among a few different choices of kernel and are also choosing parameters such as C, σ^2, etc. How should you make the choice?

- Choose whatever performs best on the training data.
- Choose whatever performs best on the cross-validation data.

Correct

- Choose whatever performs best on the test data.
- Choose whatever gives the largest SVM margin.