MachineLearning-Lecture10

Instructor (Andrew Ng):So just a couple of quick announcements. One is, first, thanks again, for all of your Problem Set 1 submissions. They've all been graded, and we'll return them at the end of lecture today. If you're an SEPD student, and you submitted your Problem Set 1 by faxing it to us, then we'll return it to you via the SEPD career. And if you handed in a hard copy of your homework, instead of in person or in the late hand in box, then they'll be available in this classroom at the end of lecture today for you to pick up. And homework's that aren't picked up today, we'll leave at the late hand in box here in the basement of the Gates building; which, by the way, you can actually access after hours, in case any of you didn't already know that.

So you can pick them up at the end of class today or again, if you're an SEPD student, and we got your homework by fax, we'll return it to you via the SEPD career. And if you haven't seen it already, Problem Set 2 has also been posted on the web page. I think it was posted online last week. So do make sure you download that if you haven't already. I'd like to say, many of you actually did very well on Problem Set 1. As an instructor, it's actually sort of personally gratifying to me when I see your homework solutions, and I say, like, "Hey, these guys actually understood what I said," so that was actually cool. And I shall also say, thanks to all the people in this class for working as graders, and stayed up very late on Monday night to get all the grading done, so just a big thank you to the graders as well.

Let's see, one more announcement. Just a reminder, that the midterm for this class is scheduled for the 8th of November at 6:00 p.m. So the midterm will be – that's about two weeks from now, I guess. So the midterm will be open book, open notes, but please don't bring – but no laptops and computers. SEPD students, if you live in the Bay area, then I'll ask that you come in person to Stanford to take the midterm in person on the evening of the 8th of November. If you're an SEPD student, and you live outside the Bay area – so if you can't drive to Stanford to take your – then please email us at the usual class mailing address: cs229qa@cs.stanford.edu. This is the same address you see on our web pages.

If you can't physically come in to attend the midterm because you live outside the bay area, then please make sure you email us by next Wednesday, so they will make alternate arrangements for the midterm. And for regular Stanford students, as well; students that aren't taking this via SEPD, and if you have a conflict. So if you have some other event of sort of equal or greater importance than the 229 midterm, like another midterm of another class that conflicts, please also email us by next Wednesday at the usual staff mailing address to let us know; okay? And so if I don't hear from you by Wednesday, I'll assume that you'll be showing up in person for the midterm. Okay. Any questions about any of that?

Okay. So, welcome back. And before I get into this lecture's technical material, I'll just say this week's discussion section will be the TA's again talking about convex optimization. So at the last week's discussion section they discussed total convex

optimization. And this week they'll wrap up the material they have to present on convex optimization.

So what I want to do today in this lecture is talk a little bit more about learning theory. In particular, I'll talk about VC dimension and building on the issues of bias variance tradeoffs of under fitting and over fitting; that we've been seeing in the previous lecture, and then we'll see in this one. I then want to talk about model selection algorithms for automatically making decisions for this bias variance tradeoff, that we started to talk about in the previous lecture. And depending on how much time, I actually may not get to Bayesian, [inaudible]. But if I don't get to this today, I'll get to this in next week's lecture.

To recap: the result we proved at the previous lecture was that if you have a finite hypothesis class – if h is a set of k hypotheses, and suppose you have some fixed parameters, gamma and delta, then in order to guarantee that this holds, we're probability at least one minus delta. It suffices that n is greater and equal to that; okay? And using big-O notations, just learning dropped constants, I can also write this as that; okay? So just to quickly remind you of what all of the notation means, we talked about empirical risk minimization, which was the simplified modern machine learning that has a hypothesis class of script h.

And what the empirical risk minimization-learning algorithm does is it just chooses the hypothesis that attains the smallest error on the training set. And so this symbol, epsilon, just denoted generalization error; right? This is the probability of a hypothesis h [inaudible] misclassifying a new example drawn from the same distribution as the training set. And so this says that in order to guarantee that the generalization error of the hypothesis h [inaudible] output by empirical risk minimization – that this is less and equal to the best possible generalization error – use it in your hypothesis class plus two times gamma – two times this error threshold. We want to guarantee that this holds a probability at least one minus delta. We show that it suffices for your training set size m to be greater than equal to this; okay? One over two gamma square log two k over delta; where again, k is the size of your hypothesis class.

And so this is some complexity result because it gives us a bound in the number of training examples we need in order to give a guarantee on something – on the error; okay? So this is a sample complexity result. So what I want to do now is take this result, and try to generalize it to the case of infinite hypothesis classes. So here, we said that the set script h is sort of just k specific functions, when you want to use a model like logistic regression, which is actually parameterized by real numbers. So I'm actually first going to give an argument that's sort of formally broken – just sort of technically somewhat broken, but conveys useful intuition. And then I'll give the more correct argument, but without proving. It's as if, full proof is somewhat involved.

So here's a somewhat broken argument. Let's say I want to apply this result analyzing logistic regression. So let's say your hypothesis class is because of all linear division boundaries; right? So say script h is parameterized by d real numbers; okay? So for

example, if you're applying logistic regression with over [inaudible], then d would be endless one with logistic regression to find the linear position boundary, parameterized by endless one real numbers.

When you think about how your hypothesis class is really represented in a computer – computers use zero one bits to represent real numbers. And so if you use like a normal standard computer, it normally will represent real numbers by what's called double position floating point numbers. And what that means is that each real number is represented by or a 64-bit representation; right?

So really – you know what floating point is in a computer. So a 64-bit floating point is what almost all of us use routinely. And so this parameterized by d real numbers, that's really as if it's parameterized by 64 times d bits. Computers can't represent real numbers. They only represent – used to speed things. And so the size of your hypothesis class in your computer representation – you have 64 times d bits that you can flip. And so the number of possible values for your 62 to 64 d bits is really just to the power of 64 d; okay? Because that's the number of ways you can flip the 64 d bits. And so this is why it's important that we that we had log k there; right? So k is therefore, to the 64 d. And if I plug it into this equation over here, what you find is that in order to get this sort of guarantee, it suffices that m is great and equal to on the order of – one of the gamma square log – it's just a 64 d over delta, which is that; okay?

So just to be clear, in order to guarantee that there's only one, instead of the same complexity result as we had before – so the question is: suppose, you want a guarantee that a hypotheses returned by empirical risk minimization will have a generalization error that's within two gamma or the best hypotheses in your hypotheses class. Then what this result suggests is that, you know, in order to give that sort of error bound guarantee, it suffices that m is greater and equal to this. In other words, that your number of training examples has to be on the order of d over gamma square; 10, 12, 1 over delta. Okay? And the intuition that this conveys is actually, roughly right.

This says, that the number of training examples you need is roughly linear in the number of parameters of your hypothesis class. That m has [inaudible] on the order of something linear, [inaudible]. That intuition is actually, roughly right. I'll say more about this later. This result is clearly, slightly broken, in the sense that it relies on a 64-bit representation of 14-point numbers. So let me actually go ahead and outline the "right way" to show this more formally; all right? And it turns out the "right way" to show this more formally involves a much longer – because the proof is extremely involved, so I'm just actually going to state the result, and not prove it.

Farther proof – be a source of learning theory balance, infinite hypothesis classes. This definition – given a set of d points, we say, a hypothesis class h shatters the set s, if h can realize any labeling on it; okay? And what I mean by realizing any labeling on it – the informal way of thinking about this is: if a hypothesis class has shattered the set s, what that means is that I can take these d points, and I can associate these d points with any caught set of labels y; right? So choose any set of labeling y for each of these d points.

And if your hypothesis class shatters s, then that means that there will be a hypothesis that labels those d examples perfectly; okay? That's what shattering means.

So let me just illustrate those in an example. So let's say h is the class of all linear classifiers into e, and let's say that s is this [inaudible] comprising two points; okay? So there are four possible labelings that computes with these two points. You can choose to label both positive; one positive, one negative, one negative, one positive or you can label both of them negative. And if the hypothesis class h classed all linear classifiers into the – then, for each of these training sets, I can sort of find a linear classifier that attains zero training error on each of these. Then on all possible labelings of this set of two points. And so I'll say that the hypothesis class script h shatters this set s of two points; okay?

One more example – show you a larger example. Suppose my set s is now this set of three points; right? Then, I now have eight possible labelings for these three points; okay? And so for these three points, I now have eight possible labelings. And once again, I can – for each of these labelings, I can find the hypothesis in the hypothesis class that labels these examples correctly. And so once again, I see that – by definition, say, that my hypothesis class also shatters this set s.

Student: Right.

Instructor (Andrew Ng):And then that – that terminology – h can realize any labeling on s. That's obviously [inaudible]. Give it any set of labels and you can find a hypothesis that perfectly separates the positive and negative examples; okay? So how about this set? Suppose s is now this set of four points, then, you know, there are lots of labels. There are now 16 labelings we can choose on this; right? That's one for instance, and this is another one; right? And so I can realize some labelings. But there's no linear division boundary that can realize this labeling, and so h does not shatter this set of four points; okay? And I'm not really going to prove it here, but it turns out that you can show that in two dimensions, there is no set of four points that – the class of all linear classifiers can shatter; okay?

So here's another definition. When I say that the – well, it's called the VC dimension. These two people, Vapnik and Chervonenkis – so given a hypothesis class, the Vapnik and Chervonenkis dimension of h, which we usually write as VC of script h, is the size of the larger set that is shattered by this set – by h. And if a hypothesis class can shatter arbitrarily large sets, then the VC dimension is infinite. So just as a kind of good example: if h is the class of all linear classifiers into d, then the VC dimension of the set is equal to three because we saw just now that there is a size of – there was a set s of size three that it could shatter, and I don't really prove it. But it turns out there is no sets of size four that it can shatter. And therefore, the VC dimension of this is three. Yeah?

Student: But there are sets of size three that cannot shatter; right? [Inaudible] was your point.

Instructor (Andrew Ng):Yes, absolutely. So it turns out that if I choose a set like this – it's actually set s, then there are labelings on this they cannot realize. And so, h cannot shatter this set. But that's okay because – right – there definitely is – there exists some other set of size three being shattered. So the VC dimension is three. And then there is no set of size four that can shatter. Yeah?

Student:[Inaudible].

Instructor (Andrew Ng):Not according to this definition. No. Right. So again, let's see, I can choose my set s to be to be a set of three points that are all over lapping. Three points in exactly the same place. And clearly, I can't shatter this set, but that's okay. And I can't shatter this set, either, but that's okay because there are some other sets of size three that I can shatter. And it turns out this result holds true into the – more generally, in any dimensions – the VC dimension of the class of linear classifiers in any dimensions is equal to n plus one. Okay? So this is in [inaudible], and if you have linear classifiers over in any dimensional feature space, the VC dimension in any dimensions; whereas, n d is equal to n plus one.

So maybe you wanna write it down: what is arguably the best-known result in all of learning theory, I guess; which is that. Let a hypothesis class be given, and let the VC dimension of h be equal to d. Then we're in probability of one minus delta. We have that – the formula on the right looks a bit complicated, but don't worry about it. I'll point out the essential aspects of it later. But the key to this result is that if you have a hypothesis class with VC dimension d, and now this can be an infinite hypothesis class, what Vapnik and Chervonenkis show is that we're probability of at least one minus delta. You enjoy this sort of uniform conversions results; okay? We have that for all hypotheses h – that for all the hypotheses in your hypothesis class, you have that the generalization error of h minus the training error of h.

So the difference between these two things is bounded above by some complicated formula like this; okay? And thus, we're probably one minus delta. We also have that – have the same thing; okay? And going from this step to this step; right? Going from this step to this step is actually something that you saw yourself; that we actually proved earlier. Because – you remember, in the previous lecture we proved that if you have uniform conversions, then that implies that – it appears actually that we showed that if generalization error and training error are close to each other; within gamma of each other, then the generalization error of the hypotheses you pick will be within two gamma times the best generalization error.

So this is really generalization error of h [inaudible] best possible generalization error plus two times gamma. And just the two constants in front here that I've absorbed into the big-O notation. So that formula is slightly more complicated. Let me just rewrite this as a corollary, which is that in order to guarantee that this holds, we're probability of one minus delta. We're probably at least one minus delta, I should say. It suffices that – I'm gonna write this – this way: I'm gonna write m equals big-O of d, and I'm going to put gamma and delta in as a subscript error to denote that. Let's see, if we treat gamma and

delta as constants, so they allow me to absorb turns that depend on gamma and delta into the big-O notation, then in order to guarantee this holds, it suffices that m is on the order of the VC dimension and hypotheses class; okay?

So let's see. So what we conclude from this is that if you have a learning algorithm that tries to – for empirical risk minimization algorithms – in other words, less formally, for learning algorithms, they try to minimize training error. The intuition to take away from this is that the number of training examples you need is therefore, roughly, linear in the VC dimension of the hypotheses class. And more formally, this shows that sample complexity is upper bounded by the VC dimension; okay? It turns out that for most reasonable hypothesis classes, it turns out that the VC dimension is sort of very similar, I guess, to the number of parameters you model. So for example, you have model and logistic regression – linear classification.

In any dimensions – logistic regression in any dimensions is endless one parameters. And the VC dimension of which is the – of class of linear classifiers is always the endless one. So it turns out that for most reasonable hypothesis classes, the VC dimension is usually linear in the number of parameters of your model. Wherein, is most sense of low other polynomial; in the number of parameters of your model. And so this – the takeaway intuition from this is that the number of training examples you need to fit in those models is going to be let's say, roughly, linear in the number of parameters in your model; okay?

There are some – somewhat strange examples where what I just said is not true. There are some strange examples where you have very few parameters, but the VC dimension is enormous. But I actually know of – all of the examples I know of that fall into that regime are somewhat strange and degenerate. So somewhat unusual, and not the source of not learning algorithms you usually use.

Let's see, just other things. It turns out that – so this result shows the sample complexity is upper bounded by VC dimension. But if you have a number of training examples that are on the order of the VC dimension, then you find – it turns out that in the worse case some complexity is also lower bounded by VC dimension. And what that means is that if you have a perfectly nasty learning problem, say, then if the number of training examples you have is less than on the order of the VC dimension; then it is not possible to prove this bound. So I guess in the worse case, sample complexity in the number of training examples you need is upper bounded and lower bounded by the VC dimension.

Let's see, questions about this?

Student: Does the proof of this assume any sort of finites of, like, finite [inaudible] like you have to just [inaudible] real numbers and [inaudible]?

Instructor (Andrew Ng):Let's see. The proof is not, no. I've actually stated the entirety of the theorem. This is true. It turns out in the proof – well, somewhere, regardless of the proof there's a step reconstruction called an epsilon net, which is a very clever [inaudible]. It' sort of in regardless of the proof, it is not an assumption that you need. In

someway that sort of proof – that's one-step that uses a very clever [inaudible] to prove this. But that's not needed; it's an assumption.

I'd like to say, back when I was a Ph.D. student, when I was working through this proof, there was sort of a solid week where I would wake up, and go to the office at 9:00 a.m. Then I'd start reading the book that led up to this proof. And then I'd read from 9:00 a.m. to 6:00 p.m. And then I'd go home, and then the next day, I'd pick up where I left off. And it sort of took me a whole week that way, to understand this proof, so I thought I would inflict that on you.

Just to tie a couple of loose ends: what I'm about to do is, I'm about to just mention a few things that will maybe, feel a little bit like random facts. But I'm just gonna tie up just a couple of loose ends. And so let's see, it turns out that – just so it will be more strong with you – so this bound was proved for an algorithm that uses empirical risk minimization, for an algorithm that minimizes 0-1 training error. So one question that some of you ask is how about support vector machines; right? How come SVM's don't over fit? And in the sequel of – remember our discussion on support vector machines said that you use kernels, and map the features in infinite dimensional feature space. And so it seems like the VC dimension should be infinite; n plus one and n is infinite.

So it turns out that the class of linear separators with large margin actually has low VC dimension. I wanna say this very quickly, and informally. It's actually, not very important for you to understand the details, but I'm going to say it very informally. It turns out that I will give you a set of points. And if I ask you to consider only the course of lines that separate these points of a large margin [inaudible], so my hypothesis class will comprise only the linear position boundaries that separate the points of a large margin. Say with a margin, at least gamma; okay. And so I won't allow a point that comes closer. Like, I won't allow that line because it comes too close to one of my points.

It turns out that if I consider my data points all lie within some sphere of radius r, and if I consider only the course of linear separators is separate to data with a margin of at least gamma, then the VC dimension of this course is less than or equal to r squared over four gamma squared plus one; okay? So this funny symbol here, that just means rounding up. This is a ceiling symbol; it means rounding up x. And it turns out you prove – and there are some strange things about this result that I'm deliberately not gonna to talk about – but turns they can prove that the VC dimension of the class of linear classifiers with large margins is actually bounded. The surprising thing about this is that this is the bound on VC dimension that has no dependents on the dimension of the points x.

So in other words, your data points x combine an infinite dimensional space, but so long as you restrict attention to the class of your separators with large margin, the VC dimension is bounded. And so in trying to find a large margin separator – in trying to find the line that separates your positive and your negative examples with large margin, it turns out therefore, that the support vector machine is automatically trying to find a hypothesis class with small VC dimension. And therefore, it does not over fit. Alex?

Student: What is the [inaudible]?

Instructor (Andrew Ng):It is actually defined the same way as finite dimensional spaces. So you know, suppose you have infinite – actually, these are constantly infinite dimensional vectors; not [inaudible] to the infinite dimensional vectors. Normally, the 2 to 1 squared is equal to some [inaudible] equals 110 xi squared, so if x is infinite dimensional, you just appoint it like that. [Inaudible]. [Crosstalk]

Student:[Inaudible].

Instructor (Andrew Ng):Now, say that again.

Student:[Inaudible].

Instructor (Andrew Ng):Yes. Although, I assume that this is bounded by r.

Student: Oh.

Instructor (Andrew Ng):It's a – yeah – so this insures that conversions. So just something people sometimes wonder about. And last, the – actually – tie empirical risk minimization back a little more strongly to the source of algorithms we've talked about. It turns out that – so the theory was about, and so far, was really for empirical risk minimization. So that view's – so we focus on just one training example. Let me draw a function, you know, a zero here jumps to one, and it looks like that. And so this for once, this training example, this may be indicator h where [inaudible] is d equals data transpose x; okay? But one training example – your training example will be positive or negative. And depending on what the value of this data transpose x is, you either get it right or wrong. And so you know, I guess if your training example – if you have a positive example, then when z is positive, you get it right.

Suppose you have a negative example, so y equals 0; right? Then if z, which is data transpose x – if this is positive, then you will get this example wrong; whereas, if z is negative then you'd get this example right. And so this is a part of indicator h subscript [inaudible] x not equals y; okay? You know, it's equal to g of data transpose x; okay? And so it turns out that – so what you really like to do is choose parameters data so as to minimize this step function; right? You'd like to choose parameters data, so that you end up with a correct classification on setting your training example, and so you'd like indicator h of x not equal y. You'd like this indicator function to be 0. It turns out this step function is clearly a non-convex function.

And so it turns out that just the linear classifiers minimizing the training error is an empty heart problem. It turns out that both logistic regression, and support vector machines can be viewed as using a convex approximation for this problem. And in particular – and draw a function like that – it turns out that logistic regression is trying to maximize likelihood. And so it's tying to minimize the minus of the logged likelihood. And if you plot the minus of the logged likelihood, it actually turns out it'll be a function that looks

like this. And this line that I just drew, you can think of it as a rough approximation to this step function; which is maybe what you're really trying to minimize, so you want to minimize training error.

So you can actually think of logistic regression as trying to approximate empirical risk minimization. Where instead of using this step function, which is non-convex, and gives you a hard optimization problem, it uses this line above – this curve above. So approximate it, so you have a convex optimization problem you can find the – maximum likelihood it's in the parameters for logistic regression. And it turns out, support vector machine also can be viewed as approximated dysfunction to only a little bit different – let's see, support vector machine turns out, can be viewed as trying to approximate this step function two over different approximation that's linear, and then – that sort of [inaudible] linear that – our results goes this [inaudible] there, and then it goes up as a linear function there. And that's – that is called the hinge class.

And so you can think of logistic regression and the support vector machine as different approximations to try to minimize this step function; okay? And that's why I guess, all the theory we developed – even though SVM's and logistic regression aren't exactly due to empirical risk minimization, the theory we develop often gives the completely appropriate intuitions for SVM's, and logistic regression; okay. So that was the last of the loose ends. And if you didn't get this, don't worry too much about it. It's a high-level message. It's just that SVM's and logistic regression are reasonable to think of as approximations – empirical risk minimization algorithms. What I want to do next is move on to talk about model selection. Before I do that, let me just check for questions about this. Okay. Cool.

Okay. So in the theory that we started to develop in the previous lecture, and that we sort of wrapped up with a discussion on VC dimension, we saw that there's often a trade-off between bias and variance. And in particular, so it is important not to choose a hypothesis that's either too simple or too complex. So if your data has sort of a quadratic structure to it, then if you choose a linear function to try to approximate it, then you would under fit. So you have a hypothesis with high bias. And conversely, we choose a hypothesis that's too complex, and you have high variance. And you'll also fail to fit. Then you would over fit the data, and you'd also fail to generalize well. So model selection algorithms provide a class of methods to automatically trade – make these tradeoffs between bias and variance; right?

So remember the cartoon I drew last time of generalization error? I drew this last time. Where on the x-axis was model complexity, meaning the number of – the degree of the polynomial; the [inaudible] regression function or whatever. And if you have too simple a model, you have high generalization error, those under fitting. And you if have too complex a model, like 15 or 14-degree polynomial to five data points, then you also have high generalization error, and you're over fitting. So what I wanna do now is actually just talk about model selection in the abstract; all right? Some examples of model selection problems will include – well, I'll run the example of – let's say you're trying to choose the degree of a polynomial; right? What degree polynomial do you want to choose?

Another example of a model selection problem would be if you're trying to choose the parameter [inaudible], which was the bandwidth parameter in locally awaited linear regression or in some sort of local way to regression. Yet, another model selection problem is if you're trying to choose the parameter c [inaudible] and as the [inaudible]; right? And so one known soft margin is the – we had this optimization objective; right? And the parameter c controls the tradeoff between how much you want to set for your example. So a large margin versus how much you want to penalize in this class [inaudible] example. So these are three specific examples of model selection problems.

And let's come up with a method for semantically choosing them. Let's say you have some finite set of models, and let's write these as m1, m2, m3, and so on. For example, this may be the linear classifier or this may be the quadratic classifier, and so on; okay? Or this may also be – you may also take the bandwidth parameter [inaudible] and discretize it into a range of values, and you're trying to choose from the most – discrete of the values. So let's talk about how you would select an appropriate model; all right? Well, one thing you could do is you can pick all of these models, and train them on you're training set. And then see which model has the lowest training error. So that's a terrible idea, and why's that?

Student:[Inaudible].

Instructor (Andrew Ng):Right. Cool. Because of the over fit; right. And those – some of you are laughing that I asked that. So that'd be a terrible idea to choose a model by looking at your training set because well, obviously, you end up choosing the most complex model; right? And you choose a 10th degree polynomial because that's what fits the training set.

So we come to model selection in a training set – several standard procedures to do this. One is hold out cross validation, and in hold out cross validation, we teach a training set. And we randomly split the training set into two subsets. We call it subset – take all the data you have and randomly split it into two subsets. And we'll call it the training set, and the hold out cross validation subset. And then, you know, you train each model on just trading subset of it, and test it on your hold out cross validation set. And you pick the model with the lowest error on the hold out cross validation subset; okay?

So this is sort of a relatively straightforward procedure, and it's commonly used where you train on 70 percent of the data. Then test all of your models. And 30 percent, you can take whatever has the smallest hold out cross validation error. And after this – you actually have a chose. You can actually – having taken all of these hypothesis trained on 70 percent of the data, you can actually just output the hypothesis that has the lowest error on your hold out cross validation set. And optionally, you can actually take the model that you selected and go back, and retrain it on all 100 percent of the data; okay?

So both versions are actually done and used really often. You can either, you know, just take the best hypothesis that was trained on 70 percent of the data, and just output that as you find the hypothesis or you can use this to – say, having chosen the degree of the

polynomial you want to fit, you can then go back and retrain the model on the entire 100 percent of your data. And both of these are commonly done. How about a cross validation does – sort of work straight? And sometimes we're working with a company or application or something. The many machine-learning applications we have very little data or where, you know, every training example you have was painfully acquired at great cost; right?

Sometimes your data is acquired by medical experiments, and each of these – each training example represents a sick man in amounts of physical human pain or something. So we talk and say, "Well, I'm going to hold out 30 percent of your data set, just to select my model." If people were who – sometimes that causes unhappiness, and so maybe you wanna use – not have to leave out 30 percent of your data just to do model selection.

So there are a couple of other variations on hold out cross validation that makes sometimes, slightly more efficient use of the data. And one is called k-fold cross validation. And here's the idea: I'm gonna take all of my data s; so imagine, I'm gonna draw this box s, as to note the entirety of all the data I have. And I'll then divide it into k pieces, and this is five pieces in what I've drawn. Then what'll I'll do is I will repeatedly train on k minus one pieces. Test on the remaining one – test on the remaining piece, I guess; right? And then you average over the k result.

So another way, we'll just hold out – I will hold out say, just 1/5 of my data and I'll train on the remaining 4/5, and I'll test on the first one. And then I'll then go and hold out the second 1/5 from my [inaudible] for the remaining pieces – test on this, you remove the third piece, train on the 4/5; I'm gonna do this five times. And then I'll take the five error measures I have and I'll average them. And this then gives me an estimate of the generalization error of my model; okay? And then, again, when you do k-fold cross validation, usually you then go back and retrain the model you selected on the entirety of your training set. So I drew five pieces here because that was easier for me to draw, but k equals 10 is very common; okay? I should say k equals 10 is the fairly common choice to do 10 fold cross validation.

And the advantage of the over hold out cross option is that you switch the data into ten pieces. Then each time you're only holding out 1/10 of your data, rather than, you know, say, 30 percent of your data. I must say, in standard hold out – in simple hold out cross validation, a 30 – 70 split is fairly common. Sometimes like 2/3 – 1/3 or a 70 – 30 split is fairly common. And if you use k-fold cross validation, k equals 5 or more commonly k equals 10, and is the most common choice. The disadvantage of k-fold cross validation is that it can be much more computationally expensive. In particular, to validate your model, you now need to train your model ten times, instead of just once. And so you need to: from logistic regression, ten times per model, rather than just once. And so this is computationally more expensive. But k equals ten works great.

And then, finally, in – there's actually a version of this that you can take even further, which is when your set k equals m. And so that's when you take your training set, and you split it into as many pieces as you have training examples. And this procedure is

called leave one out cross validation. And what you do is you then take out the first training example, train on the rest, and test on the first example. Then you take out the second training example, train on the rest, and test on the second example. Then you take out the third example, train on everything, but your third example. Test on the third example, and so on. And so with this many pieces you are now making, maybe even more effective use of your data than k-fold cross validation.

But you could leave – leave one out cross validation is computationally very expensive because now you need to repeatedly leave one example out, and then run your learning algorithm on m minus one training examples. You need to do this a lot of times, and so this is computationally very expensive. And typically, this is done only when you're extremely data scarce. So if you have a learning problem where you have, say, 15 training examples or something, then if you have very few training examples, leave one out cross validation is maybe preferred. Yeah?

Student: You know, that time you proved that the difference between the generalized [inaudible] by number of examples in your training set and VC dimension. So maybe [inaudible] examples into different groups, we can use that for [inaudible].

Instructor (Andrew Ng):Yeah, I mean –

Student:- compute the training error, and use that for computing [inaudible] for a generalized error.

Instructor (Andrew Ng):Yeah, that's done, but – yeah, in practice, I personally tend not to do that. It tends not to be – the VC dimension bounds are somewhat loose bounds. And so there are people – in structure risk minimization that propose what you do, but I personally tend not to do that, though. Questions for cross validation? Yeah.

Student: This is kind of far from there because when we spend all this time [inaudible] but how many data points do you sort of need to go into your certain marginal [inaudible]?

Instructor (Andrew Ng):Right.

Student: So it seems like when I'd be able to use that instead of do this; more analytically, I guess. I mean –

Instructor (Andrew Ng):Yeah.

Student:[Inaudible].

Instructor (Andrew Ng):No – okay. So it turns out that when you're proving learning theory bounds, very often the bounds will be extremely loose because you're sort of proving the worse case upper bound that holds true even for very bad – what is it – so the bounds that I proved just now; right? That holds true for absolutely any probability

distribution over training examples; right? So just assume the training examples we've drawn, iid from some distribution script d, and the bounds I proved hold true for absolutely any probability distribution over script d. And chances are whatever real life distribution you get over, you know, houses and their prices or whatever, is probably not as bad as the very worse one you could've gotten; okay? And so it turns out that if you actually plug in the constants of learning theory bounds, you often get extremely large numbers.

Take logistic regression – logistic regression you have ten parameters and 0.01 error, and with 95 percent probability. How many training examples do I need? If you actually plug in actual constants into the text for learning theory bounds, you often get extremely pessimistic estimates with the number of examples you need. You end up with some ridiculously large numbers. You would need 10,000 training examples to fit ten parameters. So a good way to think of these learning theory bounds is – and this is why, also, when I write papers on learning theory bounds, I quite often use big-O notation to just absolutely just ignore the constant factors because the bounds seem to be very loose.

There are some attempts to use these bounds to give guidelines as to what model to choose, and so on. But I personally tend to use the bounds – again, intuition about – for example, what are the number of training examples you need gross linearly in the number of parameters or what are your gross x dimension in number of parameters; whether it goes quadratic – parameters? So it's quite often the shape of the bounds. The fact that the number of training examples – the fact that some complexity is linear in the VC dimension, that's sort of a useful intuition you can get from these theories. But the actual magnitude of the bound will tend to be much looser than will hold true for a particular problem you are working on. So did that answer your question?

Student: Uh-huh.

Instructor (Andrew Ng):Yeah. And it turns out, by the way, for myself, a rule of thumb that I often use is if you're trying to fit a logistic regression model, if you have n parameters or n plus one parameters; if the number of training examples is ten times your number of parameters, then you're probably in good shape. And if your number of training examples is like tiny times the number of parameters, then you're probably perfectly fine fitting that model. So those are the sorts of intuitions that you can get from these bounds.

Student: In cross validation do we assume these examples randomly?

Instructor (Andrew Ng):Yes. So by convention we usually split the train testers randomly. One more thing I want to talk about for model selection is – there's actually a special case of model selections, called the feature selection problem. And so here's the intuition: for many machine-learning problems you may have a very high dimensional feature space with very high dimensional – you have x's – [inaudible] feature x's. So for example, for text classification – and I wanna talk about this text classification example that spam versus non-spam. You may easily have on the order of 30,000 or 50,000

features. I think I used 50,000 in my early examples. So if you have so many features – you have 50,000 features, depending on what learning algorithm you use, there may be a real risk of over fitting.

And so if you can reduce the number of features, maybe you can reduce the variance of your learning algorithm, and reduce the risk of over fitting. And for the specific case of text classification, if you imagine that maybe there's a small number of "relevant features," so there are all these English words. And many of these English words probably don't tell you anything at all about whether the email is spam or non-spam. If it were, you know, English function words like, the, of, a, and; these are probably words that don't tell you anything about whether the email is spam or non-spam. So words in contrast will be a much smaller number of features that are truly "relevant" to the learning problem.

So for example, you see the word buy or Viagra, those are words that are very useful. So you – words, some you spam and non-spam. You see the word Stanford or machine-learning or your own personal name. These are other words that are useful for telling you whether something is spam or non-spam. So in feature selection, we would like to select a subset of the features that may be or hopefully the most relevant ones for a specific learning problem, so as to give ourselves a simpler learning – a simpler hypothesis class to choose from. And then therefore, reduce the risk of over fitting. Even when we may have had 50,000 features originally.

So how do you do this? Well, if you have n features, then there are two to the n possible subsets; right? Because, you know, each of your n features can either be included or excluded. So there are two to the n possibilities. And this is a huge space. So in feature selection, what we most commonly do is use various searcheristics – sort of simple search algorithms to try to search through this space of two to the n possible subsets of features; to try to find a good subset of features. This is too large a number to enumerate all possible feature subsets.

And as a complete example, this is the forward search algorithm; it's also called the forward selection algorithm. It's actually pretty simple, but I'll just write it out. My writing it out will make it look more complicated than it really is, but it starts with – initialize the sets script f to be the empty set, and then repeat for i equals one to n; try adding feature i to the set scripts f, and evaluate the model using cross validation. And by cross validation, I mean any of the three flavors, be it simple hold out cross validation or k-fold cross validation or leave one out cross validation. And then, you know, set f to be equal to f union, I guess. And then the best feature found is f 1, I guess; okay? And finally, you would – okay?

So forward selections, procedure is: follow through the empty set of features. And then on each generation, take each of your features that isn't already in your set script f and you try adding that feature to your set. Then you train them all, though, and evaluate them all, though, using cross validation. And basically, figure out what is the best single feature to add to your set script f. In step two here, you go ahead and add that feature to

your set script f, and you get it right. And when I say best feature or best model here – by best, I really mean the best model according to hold out cross validation. By best, I really mean the single feature addition that results in the lowest hold out cross validation error or the lowest cross validation error. So you do this adding one feature at a time.

When you terminate this a little bit, as if you've added all the features to f, so f is now the entire set of features; you can terminate this. Or if by some rule of thumb, you know that you probably don't ever want more than k features, you can also terminate this if f is already exceeded some threshold number of features. So maybe if you have 100 training examples, and you're fitting logistic regression, you probably know you won't want more than 100 features. And so you stop after you have 100 features added to set f; okay? And then finally, having done this, output of best hypothesis found; again, by best, I mean, when learning this algorithm, you'd be seeing lots of hypothesis. You'd be training lots of hypothesis, and testing them using cross validation.

So when I say output best hypothesis found, I mean of all of the hypothesis you've seen during this entire procedure, pick the one with the lowest cross validation error that you saw; okay? So that's forward selection. So let's see, just to give this a name, this is an incidence of what's called wrapper feature selection. And the term wrapper comes from the fact that this feature selection algorithm that I just described is a forward selection or forward search. It's a piece of software that you write that wraps around your learning algorithm. In the sense that to perform forward selection, you need to repeatedly make cause to your learning algorithm to train your model, using different subsets of features; okay? So this is called wrapper model feature selection.

And it tends to be somewhat computationally expensive because as you're performing the search process, you're repeatedly training your learning algorithm over and over and over on all of these different subsets of features. Let's just mention also, there is a variation of this called backward search or backward selection, which is where you start with f equals the entire set of features, and you delete features one at a time; okay? So that's backward search or backward selection. And this is another feature selection algorithm that you might use. Part of whether this makes sense is really – there will be problems where it really doesn't even make sense to initialize f to be the set of all features.

So if you have 100 training examples and 10,000 features, which may well happen – 100 emails and 10,000 training – 10,000 features in email, then 100 training examples – then depending on the learning algorithm you're using, it may or may not make sense to initialize the set f to be all features, and train them all by using all features. And if it doesn't make sense, then you can train them all by using all features; then forward selection would be more common.

So let's see. Wrapper model feature selection algorithms tend to work well. And in particular, they actually often work better than a different class of algorithms I'm gonna talk about now. But their main disadvantage is that they're computationally very expensive. Do you have any questions about this before I talk about the other? Yeah?

Student:[Inaudible].

Instructor (Andrew Ng):Yeah – yes, you're actually right. So forward search and backward search, both of these are searcheristics, and you cannot – but for either of these you cannot guarantee they'll find the best subset of features. It actually turns out that under many formulizations of the feature selection problems – it actually turns out to be an empty heart problem, to find the best subset of features. But in practice, forward selection backward selection work fine, and you can also envision other search algorithms where you sort of have other methods to search through the space up to the end possible feature subsets.

So let's see. Wrapper feature selection tends to work well when you can afford to do it computationally. But for problems such as text classification – it turns out for text classification specifically because you have so many features, and easily have 50,000 features. Forward selection would be very, very expensive. So there's a different class of algorithms that will give you – that tends not to do as well in the sense of generalization error. So you tend to learn the hypothesis that works less well, but is computationally much less expensive. And these are called the filter feature selection methods. And the basic idea is that for each feature i will compute some measure of how informative xi is about y; okay?

And to do this, we'll use some simple heuristics; for every feature we'll just try to compute some rough estimate or compute some measure of how informative xi is about y. So there are many ways you can do this. One way you can choose is to just compute the correlation between xi and y. And just for each of your features just see how correlated this is with your class label y. And then you just pick the top k most correlated features. Another way to do this – for the case of text classification, there's one other method, which especially for this k features I guess – there's one other informative measure that's used very commonly, which is called major information.

I'm going to tell you some of these ideas in problem sets, but I'll just say this very briefly. So the major information between feature xi and y – I'll just write out the definition, I guess. Let's say this is text classification, so x can take on two values, 0, 1; the major information between xi and y is to find out some overall possible values of x; some overall possible values of y times the distribution times that. Where all of these distributions – where so the joint distribution over xi and y, you would estimate from your training data all of these things you would use, as well. You would estimate from the training data what is the probability that x is 0, what's the probability that x is one, what's the probability that x is 0, y is 0, x is one; y is 0, and so on.

So it turns out there's a standard information theoretic measure of how different probability distributions are. And I'm not gonna prove this here. But it turns out that this major information is actually – so the standard measure of how different distributions are; called the K-L divergence. When you take a class in information theory, you have seen concepts of mutual information in the K-L divergence, but if you haven't, don't worry about it. Just the intuition is there's something called K-L divergence that's a formal

measure of how different two probability distributions are. And mutual information is a measure for how different – the joint distribution is of x and y; from the distribution you would get – if you were to assume they were independent; okay?

So if x and y were independent, then p of x, y would be equal to p of x times p of y. And so you know, this distribution and this distribution would be identical, and the K-L divergence would be 0. In contrast, if x and y were very non-independent – in other words, if x and y are very informative about each other, then this K-L divergence will be large. And so mutual information is a formal measure of how non-independent x and y are. And if x and y are highly non-independent then that means that x will presumably tell you something about y, and so they'll have large mutual information. And this measure of information will tell you x might be a good feature. And you get to play with some of these ideas more in the problem sets. So I won't say much more about it.

And what you do then is – having chosen some measure like correlation or major information or something else, you then pick the top k features; meaning that you compute correlation between xi and y for all the features of mutual information – xi and y for all the features. And then you include in your learning algorithm the k features of the largest correlation with the label or the largest mutual information label, whatever. And to choose k, you can actually use cross validation, as well; okay? So you would take all your features, and sort them in decreasing order of mutual information. And then you'd try using just the top one feature, the top two features, the top three features, and so on. And you decide how many features includes using cross validation; okay? Or you can – sometimes you can just choose this by hand, as well.

Okay. Questions about this? Okay. Cool. Great. So next lecture I'll continue – I'll wrap up the Bayesian model selection, but less close to the end.

[End of Audio]

Duration: 77 minutes