MachineLearning-Lecture11

Instructor (Andrew Ng):Okay. Good morning. Welcome back.

What I want to do today is actually wrap up our discussion on learning theory and sort of on – and I'm gonna start by talking about Bayesian statistics and regularization, and then take a very brief digression to tell you about online learning. And most of today's lecture will actually be on various pieces of that, so applying machine learning algorithms to problems like, you know, like the project or other problems you may go work on after you graduate from this class.

But let's start the talk about Bayesian statistics and regularization. So you remember from last week, we started to talk about learning theory and we learned about bias and variance. And I guess in the previous lecture, we spent most of the previous lecture talking about algorithms for model selection and for feature selection. We talked about cross-validation. Right?

So most of the methods we talked about in the previous lecture were ways for you to try to simply the model. So for example, the feature selection algorithms we talked about gives you a way to eliminate a number of features, so as to reduce the number of parameters you need to fit and thereby reduce overfitting. Right? You remember that? So feature selection algorithms choose a subset of the features so that you have less parameters and you may be less likely to overfit. Right?

What I want to do today is to talk about a different way to prevent overfitting. And there's a method called regularization and there's a way that lets you keep all the parameters. So here's the idea, and I'm gonna illustrate this example with, say, linear regression. So you take the linear regression model, the very first model we learned about, right, we said that we would choose the parameters via maximum likelihood. Right? And that meant that, you know, you would choose the parameters theta that maximized the probability of the data, which is parameters theta that maximized the probability of the data we observe. Right?

And so to give this sort of procedure a name, this is one example of most common frequencies procedure, and frequency, you can think of sort of as maybe one school of statistics. And the philosophical view behind writing this down was we envisioned that there was some true parameter theta out there that generated, you know, the Xs and the Ys. There's some true parameter theta that govern housing prices, Y is a function of X, and we don't know what the value of theta is, and we'd like to come up with some procedure for estimating the value of theta. Okay? And so, maximum likelihood is just one possible procedure for estimating the unknown value for theta.

And the way you formulated this, you know, theta was not a random variable. Right? That's what why said, so theta is just some true value out there. It's not random or anything, we just don't know what it is, and we have a procedure called maximum

likelihood for estimating the value for theta. So this is one example of what's called a frequencies procedure.

The alternative to the, I guess, the frequency school of statistics is the Bayesian school, in which we're gonna say that we don't know what theta, and so we will put a prior on theta. Okay? So in the Bayesian school students would say, "Well don't know what the value of theta so let's represent our uncertainty over theta with a prior." So for example, our prior on theta may be a Gaussian distribution with mean zero and curvalence matrix given by tau squared I. Okay?

And so – actually, if I use S to denote my training set, well – right, so theta represents my beliefs about what the parameters are in the absence of any data. So not having seen any data, theta represents, you know, what I think theta – it probably represents what I think theta is most likely to be. And so given the training set, S, in the sort of Bayesian procedure, we would, well, calculate the probability, the posterior probability by parameters given my training sets, and – let's write this on the next board.

So my posterior on my parameters given my training set, by Bayes' rule, this will be proportional to, you know, this. Right? So by Bayes' rule. Let's call it posterior. And this distribution now represents my beliefs about what theta is after I've seen the training set.

And when you now want to make a new prediction on the price of a new house, on the input X, I would say that, well, the distribution over the possible housing prices for this new house I'm trying to estimate the price of, say, given the size of the house, the features of the house at X, and the training set I had previously, it is going to be given by an integral over my parameters theta of probably of Y given X comma theta and times the posterior distribution of theta given the training set. Okay?

And in particular, if you want your prediction to be the expected value of Y given the input X in training set, you would say integrate over Y times the posterior. Okay? You would take an expectation of Y with respect to your posterior distribution. Okay?

And you notice that when I was writing this down, so with the Bayesian formulation, and now started to write up here Y given X comma theta because this formula now is the property of Y conditioned on the values of the random variables X and theta. So I'm no longer writing semicolon theta, I'm writing comma theta because I'm now treating theta as a random variable.

So all of this is somewhat abstract but this is – and it turns out – actually let's check. Are there questions about this? No? Okay.

Let's try to make this more concrete. It turns out that for many problems, both of these steps in the computation are difficult because if, you know, theta is an N plus one-dimensional vector, is an N plus one-dimensional parameter vector, then this is one an integral over an N plus one-dimensional, you know, over RN plus one. And because

numerically it's very difficult to compute integrals over very high dimensional spaces, all right?

So usually this integral – actually usually it's hard to compute the posterior in theta and it's also hard to compute this integral if theta is very high dimensional. There are few exceptions for which this can be done in closed form, but for many learning algorithms, say, Bayesian logistic regression, this is hard to do.

And so what's commonly done is to take the posterior distribution and instead of actually computing a full posterior distribution, chi of theta given S, we'll instead take this quantity on the right-hand side and just maximize this quantity on the right-hand side. So let me write this down. So commonly, instead of computing the full posterior distribution, we will choose the following. Okay?

We will choose what's called the MAP estimate, or the maximum a posteriori estimate of theta, which is the most likely value of theta, most probable value of theta onto your posterior distribution. And that's just ont max chi of theta. And then when you need to make a prediction, you know, you would just predict, say, well, using your usual hypothesis and using this MAP value of theta in place of – as the parameter vector you'd choose. Okay?

And notice, the only difference between this and standard maximum likelihood estimation is that when you're choosing, you know, the – instead of choosing the maximum likelihood value for theta, you're instead maximizing this, which is what you have for maximum likelihood estimation, and then times this other quantity which is the prior. Right?

And let's see, when intuition is that if your prior is theta being Gaussian and with mean zero and some covariance, then for a distribution like this, most of the [inaudible] mass is close to zero. Right? So there's a Gaussian centered around the point zero, and so [inaudible] mass is close to zero. And so the prior distribution, instead of saying that you think most of the parameters should be close to zero, and if you remember our discussion on feature selection, if you eliminate a feature from consideration that's the same as setting the source and value of theta to be equal to zero.

All right? So if you set theta five to be equal to zero, that's the same as, you know, eliminating feature five from the your hypothesis. And so, this is the prior that drives most of the parameter values to zero – to values close to zero. And you'll think of this as doing something analogous, if – doing something reminiscent of feature selection. Okay? And it turns out that with this formulation, the parameters won't actually be exactly zero but many of the values will be close to zero.

And I guess in pictures, if you remember, I said that if you have, say, five data points and you fit a fourth-order polynomial – well I think that had too many bumps in it, but never mind. If you fit it a – if you fit very high polynomial to a very small dataset, you can get these very large oscillations if you use maximum likelihood estimation. All right?

In contrast, if you apply this sort of Bayesian regularization, you can actually fit a higher-order polynomial that still get sort of a smoother and smoother fit to the data as you decrease tau. So as you decrease tau, you're driving the parameters to be closer and closer to zero. And that in practice – it's sort of hard to see, but you can take my word for it. As tau becomes smaller and smaller, the curves you tend to fit your data also become smoother and smoother, and so you tend less and less overfit, even when you're fitting a large number of parameters. Okay?

Let's see, and one last piece of intuition that I would just toss out there. And you get to play more with this particular set of ideas more in Problem Set 3, which I'll post online later this week I guess. Is that whereas maximum likelihood tries to minimize, say, this, right?

Whereas maximum likelihood for, say, linear regression turns out to be minimizing this, it turns out that if you add this prior term there, it turns out that the authorization objective you end up optimizing turns out to be that. Where you add an extra term that, you know, penalizes your parameter theta as being large. And so this ends up being an algorithm that's very similar to maximum likelihood, expect that you tend to keep your parameters small. And this has the effect.

Again, it's kind of hard to see but just take my word for it. That strengthening the parameters has the effect of keeping the functions you fit to be smoother and less likely to overfit. Okay? Okay, hopefully this will make more sense when you play with these ideas a bit more in the next problem set. But let's check questions about all this.

Student: The smoothing behavior is it because [inaudible] actually get different [inaudible]?

Instructor (Andrew Ng):Let's see. Yeah. It depends on – well most priors with most of the mass close to zero will get this effect, I guess. And just by convention, the Gaussian prior is what's most – used the most common for models like logistic regression and linear regression, generalized in your models. There are a few other priors that I sometimes use, like the Laplace prior, but all of them will tend to have these sorts of smoothing effects.

All right. Cool.

And so it turns out that for problems like text classification, text classification is like 30,000 features or 50,000 features, where it seems like an algorithm like logistic regression would be very much prone to overfitting. Right? So imagine trying to build a spam classifier, maybe you have 100 training examples but you have 30,000 features or 50,000 features, that seems clearly to be prone to overfitting. Right? But it turns out that with this sort of Bayesian regularization, with [inaudible] Gaussian, logistic regression becomes a very effective text classification algorithm with this sort of Bayesian regularization.

Alex?

Student:[Inaudible]?

Instructor (Andrew Ng):Yeah, right, and so pick – and to pick either tau squared or lambda. I think the relation is lambda equals one over tau squared. But right, so pick either tau squared or lambda, you could use cross-validation, yeah. All right? Okay, cool.

So all right, that was all I want to say about methods for preventing overfitting. What I want to do next is just spend, you know, five minutes talking about online learning. And this is sort of a digression. And so, you know, when you're designing the syllabus of a class, I guess, sometimes there are just some ideas you want to talk about but can't find a very good place to fit in anywhere. So this is one of those ideas that may seem a bit disjointed from the rest of the class but I just want to tell you a little bit about it.

Okay. So here's the idea. So far, all the learning algorithms we've talked about are what's called batch learning algorithms, where you're given a training set and then you get to run your learning algorithm on the training set and then maybe you test it on some other test set. And there's another learning setting called online learning, in which you have to make predictions even while you are in the process of learning. So here's how the problem sees. All right?

I'm first gonna give you X one. Let's say there's a classification problem, so I'm first gonna give you X one and then gonna ask you, you know, "Can you make a prediction on X one? Is the label one or zero?" And you've not seen any data yet. And so, you make a guess. Right? You guess – we'll call your guess Y hat one. And after you've made your prediction, I will then reveal to you the true label Y one. Okay? And not having seen any data before, your odds of getting the first one right are only 50 percent, right, if you guess randomly.

And then I show you X two. And then I ask you, "Can you make a prediction on X two?" And so you now maybe are gonna make a slightly more educated guess and call that Y hat two. And after you've made your guess, I reveal the true label to you. And so, then I show you X three, and then you make your guess, and learning proceeds as follows.

So this is just a lot of machine learning and batch learning, and the model settings where you have to keep learning even as you're making predictions, okay? So I don't know, setting your website and you have users coming in. And as the first user comes in, you need to start making predictions already about what the user likes or dislikes. And there's only, you know, as you're making predictions you get to show more and more training examples.

So in online learning what you care about is the total online error, which is sum from I equals one to MC if you get the sequence of M examples all together, indicator Y hat I not equal to Y hi. Okay? So the total online error is the total number of mistakes you make on a sequence of examples like this.

And it turns out that, you know, many of the learning algorithms you have – when you finish all the learning algorithms, you've learned about and can apply to this setting. One thing you could do is when you're asked to make prediction on Y hat three, right, one simple thing to do is well you've seen some other training examples up to this point so you can just take your learning algorithm and run it on the examples, you know, leading up to Y hat three. So just run the learning algorithm on all the examples you've seen previous to being asked to make a prediction on certain example, and then use your learning algorithm to make a prediction on the next example.

And it turns out that there are also algorithms, especially the algorithms that we saw that you could use the stochastic gradient descent, that, you know, can be adapted very nicely to this. So as a concrete example, if you remember the perceptron algorithms, say, right, you would say initial the parameter theta to be equal to zero.

And then after seeing the Ith training example, you'd update the parameters, you know, using – you've see this reel a lot of times now, right, using the standard perceptron learning rule. And the same thing, if you were using logistic regression you can then, again, after seeing each training example, just run, you know, essentially run one-step stochastic gradient descent on just the example you saw. Okay?

And so the reason I've put this into the sort of "learning theory" section of this class was because it turns that sometimes you can prove fairly amazing results on your total online error using algorithms like these. I will actually – I don't actually want to spend the time in the main lecture to prove this, but, for example, you can prove that when you use the perceptron algorithm, then even when the features XI, maybe infinite dimensional feature vectors, like we saw for simple vector machines. And sometimes, infinite feature dimensional vectors may use kernel representations. Okay?

But so it turns out that you can prove that when you a perceptron algorithm, even when the data is maybe extremely high dimensional and it seems like you'd be prone to overfitting, right, you can prove that so as the long as the positive and negative examples are separated by a margin, right.

So in this infinite dimensional space, so long as, you know, there is some margin down there separating the positive and negative examples, you can prove that perceptron algorithm will converge to a hypothesis that perfectly separates the positive and negative examples. Okay? And then so after seeing only a finite number of examples, it'll converge to digital boundary that perfectly separates the positive and negative examples, even though you may in an infinite dimensional space. Okay?

So let's see. The proof itself would take me sort of almost an entire lecture to do, and there are sort of other things that I want to do more than that. So you want to see the proof of this yourself, it's actually written up in the lecture notes that I posted online. For the purposes of this class' syllabus, the proof of this result, you can treat this as optional reading. And by that, I mean, you know, it won't appear on the midterm and you won't be asked about this specifically in the problem sets, but I thought it'd be –

I know some of you are curious after the previous lecture so why you can prove that, you know, SVMs can have bounded VC dimension, even in these infinite dimensional spaces, and how do you prove things in these – how do you prove learning theory results in these infinite dimensional feature spaces. And so the perceptron bound that I just talked about was the simplest instance I know of that you can sort of read in like half an hour and understand it.

So if you're interested, there are lecture notes online for how this perceptron bound is actually proved. It's a very [inaudible], you can prove it in like a page or so, so go ahead and take a look at that if you're interested. Okay?

But regardless of the theoretical results, you know, the online learning setting is something that you – that comes reasonably often. And so, these algorithms based on stochastic gradient descent often go very well. Okay, any questions about this before I move on? All right. Cool.

So the last thing I want to do today, and was the majority of today's lecture, actually can I switch to PowerPoint slides, please, is I actually want to spend most of today's lecture sort of talking about advice for applying different machine learning algorithms.

And so, you know, right now, already you have a, I think, a good understanding of really the most powerful tools known to humankind in machine learning. Right? And what I want to do today is give you some advice on how to apply them really powerfully because, you know, the same tool – it turns out that you can take the same machine learning tool, say logistic regression, and you can ask two different people to apply it to the same problem. And sometimes one person will do an amazing job and it'll work amazingly well, and the second person will sort of not really get it to work, even though it was exactly the same algorithm. Right?

And so what I want to do today, in the rest of the time I have today, is try to convey to you, you know, some of the methods for how to make sure you're one of – you really know how to get these learning algorithms to work well in problems. So just some caveats on what I'm gonna, I guess, talk about in the rest of today's lecture.

Something I want to talk about is actually not very mathematical but is also some of the hardest, most conceptually most difficult material in this class to understand. All right? So this is not mathematical but this is not easy. And I want to say this caveat some of what I'll say today is debatable. I think most good machine learning people will agree with most of what I say but maybe not everything I say. And some of what I'll say is also not good advice for doing machine learning either, so I'll say more about this later.

What I'm focusing on today is advice for how to just get stuff to work. If you work in the company and you want to deliver a product or you're, you know, building a system and you just want your machine learning system to work. Okay? Some of what I'm about to say today isn't great advice if you goal is to invent a new machine learning algorithm, but

this is advice for how to make machine learning algorithm work and, you know, and deploy a working system.

So three key areas I'm gonna talk about. One: diagnostics for debugging learning algorithms. Second: sort of talk briefly about error analyses and ablative analysis. And third, I want to talk about just advice for how to get started on a machine-learning problem. And one theme that'll come up later is it turns out you've heard about premature optimization, right, in writing software. This is when someone over-designs from the start, when someone, you know, is writing piece of code and they choose a subroutine to optimize heavily. And maybe you write the subroutine as assembly or something.

And that's often – and many of us have been guilty of premature optimization, where we're trying to get a piece of code to run faster. And we choose probably a piece of code and we implement it an assembly, and really tune and get to run really quickly. And it turns out that wasn't the bottleneck in the code at all. Right? And we call that premature optimization. And in undergraduate programming classes, we warn people all the time not to do premature optimization and people still do it all the time. Right?

And turns out, a very similar thing happens in building machine-learning systems. That many people are often guilty of, what I call, premature statistical optimization, where they heavily optimize part of a machine learning system and that turns out not to be the important piece. Okay? So I'll talk about that later, as well.

So let's first talk about debugging learning algorithms. As a motivating example, let's say you want to build an anti-spam system. And let's say you've carefully chosen, you know, a small set of 100 words to use as features. All right? So instead of using 50,000 words, you're chosen a small set of 100 features to use for your anti-spam system. And let's say you implement Bayesian logistic regression, implement gradient descent, and you get 20 percent test error, which is unacceptably high. Right?

So this is Bayesian logistic regression, and so it's just like maximum likelihood but, you know, with that additional lambda squared term. And we're maximizing rather than minimizing as well, so there's a minus lambda theta square instead of plus lambda theta squared. So the question is, you implemented your Bayesian logistic regression algorithm, and you tested it on your test set and you got unacceptably high error, so what do you do next? Right?

So, you know, one thing you could do is think about the ways you could improve this algorithm. And this is probably what most people will do instead of, "Well let's sit down and think what could've gone wrong, and then we'll try to improve the algorithm." Well obviously having more training data could only help, so one thing you can do is try to get more training examples. Maybe you suspect, that even 100 features was too many, so you might try to get a smaller set of features.

What's more common is you might suspect your features aren't good enough, so you might spend some time, look at the email headers, see if you can figure out better features for, you know, finding spam emails or whatever. Right. And right, so and just sit around and come up with better features, such as for email headers. You may also suspect that gradient descent hasn't quite converged yet, and so let's try running gradient descent a bit longer to see if that works. And clearly, that can't hurt, right, just run gradient descent longer.

Or maybe you remember, you know, you remember hearing from class that maybe Newton's method converges better, so let's try that instead. You may want to tune the value for lambda, because not sure if that was the right thing, or maybe you even want to an SVM because maybe you think an SVM might work better than logistic regression.

So I only listed eight things here, but you can imagine if you were actually sitting down, building machine-learning system, the options to you are endless. You can think of, you know, hundreds of ways to improve a learning system. And some of these things like, well getting more training examples, surely that's gonna help, so that seems like it's a good use of your time. Right?

And it turns out that this [inaudible] of picking ways to improve the learning algorithm and picking one and going for it, it might work in the sense that it may eventually get you to a working system, but often it's very time-consuming. And I think it's often a largely – largely a matter of luck, whether you end up fixing what the problem is.

In particular, these eight improvements all fix very different problems. And some of them will be fixing problems that you don't have. And if you can rule out six of eight of these, say, you could – if by somehow looking at the problem more deeply, you can figure out which one of these eight things is actually the right thing to do, you can save yourself a lot of time. So let's see how we can go about doing that.

The people in industry and in research that I see that are really good, would not go and try to change a learning algorithm randomly. There are lots of things that obviously improve your learning algorithm, but the problem is there are so many of them it's hard to know what to do. So you find all the really good ones that run various diagnostics to figure out the problem is and they think where a problem is. Okay?

So for our motivating story, right, we said – let's say Bayesian logistic regression test error was 20 percent, which let's say is unacceptably high. And let's suppose you suspected the problem is either overfitting, so it's high bias, or you suspect that, you know, maybe you have two few features that classify as spam, so there's –

Oh excuse me; I think I wrote that wrong. Let's firstly – so let's forget – forget the tables.

Suppose you suspect the problem is either high bias or high variance, and some of the text here doesn't make sense. And you want to know if you're overfitting, which would

be high variance, or you have too few features classified as spam, it'd be high bias. I had those two reversed, sorry. Okay?

So how can you figure out whether the problem is one of high bias or high variance? Right? So it turns out there's a simple diagnostic you can look at that will tell you whether the problem is high bias or high variance. If you remember the cartoon we'd seen previously for high variance problems, when you have high variance the training error will be much lower than the test error. All right?

When you have a high variance problem, that's when you're fitting your training set very well. That's when you're fitting, you know, a tenth order polynomial to 11 data points. All right? And that's when you're just fitting the data set very well, and so your training error will be much lower than your test error.

And in contrast, if you have high bias, that's when your training error will also be high. Right? That's when your data is quadratic, say, but you're fitting a linear function to it and so you aren't even fitting your training set well.

So just in cartoons, I guess, this is a – this is what a typical learning curve for high variance looks like. On your horizontal axis, I'm plotting the training set size M, right, and on vertical axis, I'm plotting the error. And so, let's see, you know, as you increase – if you have a high variance problem, you'll notice as the training set size, M, increases, your test set error will keep on decreasing. And so this sort of suggests that, well, if you can increase the training set size even further, maybe if you extrapolate the green curve out, maybe that test set error will decrease even further. All right?

Another thing that's useful to plot here is – let's say the red horizontal line is the desired performance you're trying to reach, another useful thing to plot is actually the training error. Right? And it turns out that your training error will actually grow as a function of the training set size because the larger your training set, the harder it is to fit, you know, your training set perfectly. Right?

So this is just a cartoon, don't take it too seriously, but in general, your training error will actually grow as a function of your training set size. Because smart training sets, if you have one data point, it's really easy to fit that perfectly, but if you have 10,000 data points, it's much harder to fit that perfectly. All right?

And so another diagnostic for high variance, and the one that I tend to use more, is to just look at training versus test error. And if there's a large gap between them, then this suggests that, you know, getting more training data may allow you to help close that gap. Okay? So this is what the cartoon would look like when – in the case of high variance.

This is what the cartoon looks like for high bias. Right? If you look at the learning curve, you see that the curve for test error has flattened out already. And so this is a sign that, you know, if you get more training examples, if you extrapolate this curve further to the

right, it's maybe not likely to go down much further. And this is a property of high bias: that getting more training data won't necessarily help.

But again, to me the more useful diagnostic is if you plot training errors well, if you look at your training error as well as your, you know, hold out test set error. If you find that even your training error is high, then that's a sign that getting more training data is not going to help. Right?

In fact, you know, think about it, training error grows as a function of your training set size. And so if your training error is already above your level of desired performance, then getting even more training data is not going to reduce your training error down to the desired level of performance. Right? Because, you know, your training error sort of only gets worse as you get more and more training examples. So if you extrapolate further to the right, it's not like this blue line will come back down to the level of desired performance. Right? This will stay up there. Okay?

So for me personally, I actually, when looking at a curve like the green curve on test error, I actually personally tend to find it very difficult to tell if the curve is still going down or if it's [inaudible]. Sometimes you can tell, but very often, it's somewhat ambiguous. So for me personally, the diagnostic I tend to use the most often to tell if I have a bias problem or a variance problem is to look at training and test error and see if they're very close together or if they're relatively far apart. Okay?

And so, going back to the list of fixes, look at the first fix, getting more training examples is a way to fix high variance. Right? If you have a high variance problem, getting more training examples will help. Trying a smaller set of features: that also fixes high variance. All right? Trying a larger set of features or adding email features, these are solutions that fix high bias. Right? So high bias being if you're hypothesis was too simple, you didn't have enough features. Okay?

And so quite often you see people working on machine learning problems and they'll remember that getting more training examples helps. And so, they'll build a learning system, build an anti-spam system and it doesn't work. And then they go off and spend lots of time and money and effort collecting more training data because they'll say, "Oh well, getting more data's obviously got to help." But if they had a high bias problem in the first place, and not a high variance problem, it's entirely possible to spend three months or six months collecting more and more training data, not realizing that it couldn't possibly help. Right?

And so, this actually happens a lot in, you know, in Silicon Valley and companies, this happens a lot. There will often people building various machine learning systems, and they'll often – you often see people spending six months working on fixing a learning algorithm and you could've told them six months ago that, you know, that couldn't possibly have helped. But because they didn't know what the problem was, and they'd easily spend six months trying to invent new features or something.

And this is – you see this surprisingly often and this is somewhat depressing. You could've gone to them and told them, "I could've told you six months ago that this was not going to help." And the six months is not a joke, you actually see this. And in contrast, if you actually figure out the problem's one of high bias or high variance, then you can rule out two of these solutions and save yourself many months of fruitless effort. Okay?

I actually want to talk about these four at the bottom as well. But before I move on, let me just check if there were questions about what I've talked about so far. No? Okay, great.

So bias versus variance is one thing that comes up often. This bias versus variance is one common diagnostic. And so, for other machine learning problems, it's often up to your own ingenuity to figure out your own diagnostics to figure out what's wrong. All right? So if a machine-learning algorithm isn't working, very often it's up to you to figure out, you know, to construct your own tests. Like do you look at the difference training and test errors or do you look at something else? It's often up to your own ingenuity to construct your own diagnostics to figure out what's going on.

What I want to do is go through another example. All right? And this one is slightly more contrived but it'll illustrate another common question that comes up, another one of the most common issues that comes up in applying learning algorithms.

So in this example, it's slightly more contrived, let's say you implement Bayesian logistic regression and you get 2 percent error on spam mail and 2 percent error non-spam mail. Right? So it's rejecting, you know, 2 percent of – it's rejecting 98 percent of your spam mail, which is fine, so 2 percent of all spam gets through which is fine, but is also rejecting 2 percent of your good email, 2 percent of the email from your friends and that's unacceptably high, let's say.

And let's say that a simple vector machine using a linear kernel gets 10 percent error on spam and 0.01 percent error on non-spam, which is more of the acceptable performance you want. And let's say for the sake of this example, let's say you're trying to build an anti-spam system. Right? Let's say that you really want to deploy logistic regression to your customers because of computational efficiency or because you need retrain overnight every day, and because logistic regression just runs more easily and more quickly or something. Okay?

So let's say you want to deploy logistic regression, but it's just not working out well. So question is: What do you do next? So it turns out that this – the issue that comes up here, the one other common question that comes up is a question of is the algorithm converging. So you might suspect that maybe the problem with logistic regression is that it's just not converging. Maybe you need to run iterations.

And it turns out that, again if you look at the optimization objective, say, logistic regression is, let's say, optimizing J of theta, it actually turns out that if you look at

optimizing your objective as a function of the number of iterations, when you look at this curve, you know, it sort of looks like it's going up but it sort of looks like there's absentiles.

And when you look at these curves, it's often very hard to tell if the curve has already flattened out. All right? And you look at these curves a lot so you can ask: Well has the algorithm converged? When you look at the J of theta like this, it's often hard to tell. You can run this ten times as long and see if it's flattened out. And you can run this ten times as long and it'll often still look like maybe it's going up very slowly, or something. Right? So a better diagnostic for what logistic regression is converged than looking at this curve.

The other question you might wonder – the other thing you might suspect is a problem is are you optimizing the right function. So what you care about, right, in spam, say, is a weighted accuracy function like that. So A of theta is, you know, sum over your examples of some weights times whether you got it right. And so the weight may be higher for non-spam than for spam mail because you care about getting your predictions correct for spam email much more than non-spam mail, say.

So let's say A of theta is the optimization objective that you really care about, but Bayesian logistic regression is that it optimizes a quantity like that. Right? It's this sort of maximum likelihood thing and then with this two-nom, you know, penalty thing that we saw previously. And you might be wondering: Is this the right optimization function to be optimizing. Okay? And: Or do I maybe need to change the value for lambda to change this parameter? Or: Should I maybe really be switching to support vector machine optimization objective?

Okay? Does that make sense?

So the second diagnostic I'm gonna talk about is let's say you want to figure out is the algorithm converging, is the optimization algorithm converging, or is the problem with the optimization objective I chose in the first place? Okay?

So here's the diagnostic you can use. Let me let – right. So to just reiterate the story, right, let's say an SVM outperforms Bayesian logistic regression but you really want to deploy Bayesian logistic regression to your problem. Let me let theta subscript SVM, be the parameters learned by an SVM, and I'll let theta subscript BLR be the parameters learned by Bayesian logistic regression.

So the optimization objective you care about is this, you know, weighted accuracy criteria that I talked about just now. And the support vector machine outperforms Bayesian logistic regression. And so, you know, the weighted accuracy on the support-vector-machine parameters is better than the weighted accuracy for Bayesian logistic regression.

So further, Bayesian logistic regression tries to optimize an optimization objective like that, which I denoted J theta. And so, the diagnostic I choose to use is to see if J of SVM is bigger-than or less-than J of BLR. Okay? So I explain this on the next slide.

So we know two facts. We know that – well we know one fact. We know that a weighted accuracy of support vector machine, right, is bigger than this weighted accuracy of Bayesian logistic regression. So in order for me to figure out whether Bayesian logistic regression is converging, or whether I'm just optimizing the wrong objective function, the diagnostic I'm gonna use and I'm gonna check if this equality hold through. Okay?

So let me explain this, so in Case 1, right, it's just those two equations copied over. In Case 1, let's say that J of SVM is, indeed, is greater than J of BLR – or J of theta SVM is greater than J of theta BLR. But we know that Bayesian logistic regression was trying to maximize J of theta; that's the definition of Bayesian logistic regression.

So this means that theta – the value of theta output that Bayesian logistic regression actually fails to maximize J because the support back to machine actually returned the value of theta that, you know does a better job out-maximizing J. And so, this tells me that Bayesian logistic regression didn't actually maximize J correctly, and so the problem is with the optimization algorithm. The optimization algorithm hasn't converged.

The other case is as follows, where J of theta SVM is less-than/equal to J of theta BLR. Okay? In this case, what does that mean? This means that Bayesian logistic regression actually attains the higher value for the optimization objective J then doesn't support back to machine. The support back to machine, which does worse on your optimization problem, actually does better on the weighted accuracy measure.

So what this means is that something that does worse on your optimization objective, on J, can actually do better on the weighted accuracy objective. And this really means that maximizing J of theta, you know, doesn't really correspond that well to maximizing your weighted accuracy criteria. And therefore, this tells you that J of theta is maybe the wrong optimization objective to be maximizing. Right? That just maximizing J of theta just wasn't a good objective to be choosing if you care about the weighted accuracy. Okay?

Can you raise your hand if this made sense? Cool, good.

So that tells us whether the problem is with the optimization objective or whether it's with the objective function. And so going back to this slide, the eight fixes we had, you notice that if you run gradient descent for more iterations that fixes the optimization algorithm. You try and use this method fixes the optimization algorithm, whereas using a different value for lambda, in that lambda times norm of data squared, you know, in your objective, fixes the optimization objective. And changing to an SVM is also another way of trying to fix the optimization objective. Okay?

And so once again, you actually see this quite often that – actually, you see it very often, people will have a problem with the optimization objective and be working harder and harder to fix the optimization algorithm. That's another very common pattern that the problem is in the formula from your J of theta, that often you see people, you know, just running more and more iterations of gradient descent. Like trying Newton's method and trying conjugate and then trying more and more crazy optimization algorithms, whereas the problem was, you know, optimizing J of theta wasn't going to fix the problem at all. Okay?

So there's another example of when these sorts of diagnostics will help you figure out whether you should be fixing your optimization algorithm or fixing the optimization objective. Okay?

Let me think how much time I have. Hmm, let's see. Well okay, we have time. Let's do this. Show you one last example of a diagnostic. This is one that came up in, you know, my students' and my work on flying helicopters. This one actually, this example is the most complex of the three examples I'm gonna do today. I'm going to somewhat quickly, and this actually draws on reinforcement learning which is something that I'm not gonna talk about until towards – close to the end of the course here, but this just a more complicated example of a diagnostic we're gonna go over.

What I'll do is probably go over this fairly quickly, and then after we've talked about reinforcement learning in the class, I'll probably actually come back and redo this exact same example because you'll understand it more deeply. Okay?

So some of you know that my students and I fly autonomous helicopters, so how do you get a machine-learning algorithm to design the controller for helicopter? This is what we do. All right? This first step was you build a simulator for a helicopter, so, you know, there's a screenshot of our simulator. This is just like a – it's like a joystick simulator; you can fly a helicopter in simulation.

And then you choose a cost function, it's actually called a [inaudible] function, but for this actually I'll call it cost function. Say J of theta is, you know, the expected squared error in your helicopter's position. Okay? So this is J of theta is maybe it's expected square error or just the square error. And then we run a reinforcement-learning algorithm, you'll learn about RL algorithms in a few weeks. You run reinforcement learning algorithm in your simulator to try to minimize this cost function; try to minimize the squared error of how well you're controlling your helicopter's position. Okay?

The reinforcement learning algorithm will output some parameters, which I'm denoting theta subscript RL, and then you'll use that to fly your helicopter. So suppose you run this learning algorithm and you get out a set of controller parameters, theta subscript RL, that gives much worse performance than a human pilot. Then what do you do next? And in particular, you know, corresponding to the three steps above, there are three natural things you can try. Right?

You can try to – oh, the bottom of the slide got chopped off. You can try to improve the simulator. And maybe you think your simulator's isn't that accurate, you need to capture the aerodynamic effects more accurately. You need to capture the airflow and the turbulence affects around the helicopter more accurately. Maybe you need to modify the cost function. Maybe your square error isn't cutting it. Maybe what a human pilot does isn't just optimizing square area but it's something more subtle. Or maybe the reinforcement-learning algorithm isn't working; maybe it's not quite converging or something. Okay?

So these are the diagnostics that I actually used, and my students and I actually use to figure out what's going on. Actually, why don't you just think about this for a second and think what you'd do, and then I'll go on and tell you what we do. All right, so let me tell you what – how we do this and see whether it's the same as yours or not. And if you have a better idea than I do, let me know and I'll let you try it on my helicopter.

So here's a reasoning that I wanted to experiment, right. So, yeah, let's say the controller output by our reinforcement-learning algorithm does poorly. Well suppose the following three things hold true. Suppose the contrary, I guess. Suppose that the helicopter simulator is accurate, so let's assume we have an accurate model of our helicopter. And let's suppose that the reinforcement learning algorithm, you know, correctly controls the helicopter in simulation, so we tend to run a learning algorithm in simulation so that, you know, the learning algorithm can crash a helicopter and it's fine. Right?

So let's assume our reinforcement-learning algorithm correctly controls the helicopter so as to minimize the cost function J of theta. And let's suppose that minimizing J of theta does indeed correspond to accurate or the correct autonomous flight. If all of these things held true, then that means that the parameters, theta RL, should actually fly well on my real helicopter. Right? And so the fact that the learning control parameters, theta RL, does not fly well on my helicopter, that sort of means that ones of these three assumptions must be wrong and I'd like to figure out which of these three assumptions is wrong. Okay?

So these are the diagnostics we use. First one is we look at the controller and see if it even flies well in simulation. Right? So the simulator of the helicopter that we did the learning on, and so if the learning algorithm flies well in the simulator but it doesn't fly well on my real helicopter, then that tells me the problem is probably in the simulator. Right? My simulator predicts the helicopter's controller will fly well but it doesn't actually fly well in real life, so could be the problem's in the simulator and we should spend out efforts improving the accuracy of our simulator.

Otherwise, let me write theta subscript human, be the human control policy. All right? So let's go ahead and ask a human to fly the helicopter, it could be in the simulator, it could be in real life, and let's measure, you know, the means squared error of the human pilot's flight. And let's see if the human pilot does better or worse than the learned controller, in terms of optimizing this objective function J of theta. Okay?

So if the human does worse, if even a very good human pilot attains a worse value on my optimization objective, on my cost function, than my learning algorithm, then the problem is in the reinforcement-learning algorithm. Because my reinforcement-learning algorithm was trying to minimize J of theta, but a human actually attains a lower value for J of theta than does my algorithm. And so that tells me that clearly my algorithm's not managing to minimize J of theta and that tells me the problem's in the reinforcement learning algorithm.

And finally, if J of theta – if the human actually attains a larger value for theta – excuse me, if the human actually attains a larger value for J of theta, the human actually has, you know, larger mean squared error for the helicopter position than does my reinforcement learning algorithms, that's I like – but I like the way the human flies much better than my reinforcement learning algorithm.

So if that holds true, then clearly the problem's in the cost function, right, because the human does worse on my cost function but flies much better than my learning algorithm. And so that means the problem's in the cost function. It means – oh excuse me, I meant minimizing it, not maximizing it, there's a typo on the slide, because that means that minimizing the cost function – my learning algorithm does a better job minimizing the cost function but doesn't fly as well as a human pilot. So that tells you that minimizing the cost function doesn't correspond to good autonomous flight. And what you should do it go back and see if you can change J of theta. Okay?

And so for those reinforcement learning problems, you know, if something doesn't work – often reinforcement learning algorithms just work but when they don't work, these are the sorts of diagnostics you use to figure out should we be focusing on the simulator, on changing the cost function, or on changing the reinforcement learning algorithm.

And again, if you don't know which of your three problems it is, it's entirely possible, you know, to spend two years, whatever, changing, building a better simulator for your helicopter. But it turns out that modeling helicopter aerodynamics is an active area of research. There are people, you know, writing entire PhD theses on this still. So it's entirely possible to go out and spend six years and write a PhD thesis and build a much better helicopter simulator, but if you're fixing the wrong problem it's not gonna help.

So quite often, you need to come up with your own diagnostics to figure out what's happening in an algorithm when something is going wrong. And unfortunately I don't know of – what I've described are sort of maybe some of the most common diagnostics that I've used, that I've seen, you know, to be useful for many problems. But very often, you need to come up with your own for your own specific learning problem.

And I just want to point out that even when the learning algorithm is working well, it's often a good idea to run diagnostics, like the ones I talked about, to make sure you really understand what's going on. All right? And this is useful for a couple of reasons. One is that diagnostics like these will often help you to understand your application problem better. So some of you will, you know, graduate from Stanford and go on to get some

amazingly high-paying job to apply machine-learning algorithms to some application problem of, you know, significant economic interest. Right?

And you're gonna be working on one specific important machine learning application for many months, or even for years. One of the most valuable things for you personally will be for you to get in – for you personally to get in an intuitive understanding of what works and what doesn't work your problem. Sort of right now in the industry, in Silicon Valley or around the world, there are many companies with important machine learning problems and there are often people working on the same machine learning problem, you know, for many months or for years on end.

And when you're doing that, I mean solving a really important problem using learning algorithms, one of the most valuable things is just your own personal intuitive understanding of the problem. Okay? And diagnostics, like the sort I talked about, will be one way for you to get a better and better understanding of these problems.

It turns out, by the way, there are some of Silicon Valley companies that outsource their machine learning. So there's sometimes, you know, whatever. They're a company in Silicon Valley and they'll, you know, hire a firm in New York to run all their learning algorithms for them.

And I'm not a businessman, but I personally think that's often a terrible idea because if your expertise, if your understanding of your data is given, you know, to an outsource agency, then if you don't maintain that expertise, if there's a problem you really care about then it'll be your own, you know, understanding of the problem that you build up over months that'll be really valuable. And if that knowledge is outsourced, you don't get to keep that knowledge yourself. I personally think that's a terrible idea. But I'm not a businessman, but I just see people do that a lot, and just.

Let's see. Another reason for running diagnostics like these is actually in writing research papers, right? So diagnostics and error analyses, which I'll talk about in a minute, often help to convey insight about the problem and help justify your research claims.

So for example, rather than writing a research paper, say, that's says, you know, "Oh well here's an algorithm that works. I built this helicopter and it flies," or whatever, it's often much more interesting to say, "Here's an algorithm that works, and it works because of a specific component X. And moreover, here's the diagnostic that gives you justification that shows X was the thing that fixed this problem," and that's where you made it work. Okay?

So that leads me into a discussion on error analysis, which is often good machine learning practice, is a way for understanding what your sources of errors are. So what I call error analyses – and let's check questions about this. Yeah?

Student: What ended up being wrong with the helicopter?

Instructor (Andrew Ng):Oh, don't know. Let's see. We've flown so many times. The thing that is most difficult a helicopter is actually building a very – I don't know. It changes all the time. Quite often, it's actually the simulator. Building an accurate simulator of a helicopter is very hard. Yeah. Okay.

So for error analyses, this is a way for figuring out what is working in your algorithm and what isn't working. And we're gonna talk about two specific examples. So there are many learning – there are many sort of IA systems, many machine learning systems, that combine many different components into a pipeline. So here's sort of a contrived example for this, not dissimilar in many ways from the actual machine learning systems you see.

So let's say you want to recognize people from images. This is a picture of one of my friends. So you take this input in camera image, say, and you often run it through a long pipeline. So for example, the first thing you may do may be preprocess the image and remove the background, so you remove the background. And then you run a face detection algorithm, so a machine learning algorithm to detect people's faces. Right?

And then, you know, let's say you want to recognize the identity of the person, right, this is your application. You then segment of the eyes, segment of the nose, and have different learning algorithms to detect the mouth and so on. I know; she might not want to be friend after she sees this. And then having found all these features, based on, you know, what the nose looks like, what the eyes looks like, whatever, then you feed all the features into a logistic regression algorithm. And your logistic regression or soft match regression, or whatever, will tell you the identity of this person. Okay?

So this is what error analysis is. You have a long complicated pipeline combining many machine learning components. Many of these would be used in learning algorithms. And so, it's often very useful to figure out how much of your error can be attributed to each of these components. So what we'll do in a typical error analysis procedure is we'll repeatedly plug in the ground-truth for each component and see how the accuracy changes.

So what I mean by that is the figure on the bottom left – bottom right, let's say the overall accuracy of the system is 85 percent. Right? Then I want to know where my 15 percent of error comes from. And so what I'll do is I'll go to my test set and I'll actually code it and – oh, instead of – actually implement my correct background removal. So actually, go in and give it, give my algorithm what is the correct background versus foreground.

And if I do that, let's color that blue to denote that I'm giving that ground-truth data in the test set, let's assume our accuracy increases to 85.1 percent. Okay? And now I'll go in and, you know, give my algorithm the ground-truth, face detection output. So I'll go in and actually on my test set I'll just tell the algorithm where the face is. And if I do that, let's say my algorithm's accuracy increases to 91 percent, and so on.

And then I'll go for each of these components and just give it the ground-truth label for each of the components, because say, like, the nose segmentation algorithm's trying to figure out where the nose is. I just in and tell it where the nose is so that it doesn't have to figure that out. And as I do this, one component through the other, you know, I end up giving it the correct output label and end up with 100 percent accuracy.

And now you can look at this table – I'm sorry this is cut off on the bottom, it says logistic regression 100 percent. Now you can look at this table and see, you know, how much giving the ground-truth labels for each of these components could help boost your final performance. In particular, if you look at this table, you notice that when I added the face detection ground-truth, my performance jumped from 85.1 percent accuracy to 91 percent accuracy. Right?

So this tells me that if only I can get better face detection, maybe I can boost my accuracy by 6 percent. Whereas in contrast, when I, you know, say plugged in better, I don't know, background removal, my accuracy improved from 85 to 85.1 percent. And so, this sort of diagnostic also tells you that if your goal is to improve the system, it's probably a waste of your time to try to improve your background subtraction. Because if even if you got the ground-truth, this is gives you, at most, 0.1 percent accuracy, whereas if you do better face detection, maybe there's a much larger potential for gains there. Okay?

So this sort of diagnostic, again, is very useful because if your is to improve the system, there are so many different pieces you can easily choose to spend the next three months on. Right? And choosing the right piece is critical, and this sort of diagnostic tells you what's the piece that may actually be worth your time to work on.

There's sort of another type of analyses that's sort of the opposite of what I just talked about. The error analysis I just talked about tries to explain the difference between the current performance and perfect performance, whereas this sort of ablative analysis tries to explain the difference between some baselines, some really bad performance and your current performance.

So for this example, let's suppose you've built a very good anti-spam classifier for adding lots of clever features to your logistic regression algorithm. Right? So you added features for spam correction, for, you know, sender host features, for email header features, email text parser features, JavaScript parser features, features for embedded images, and so on.

So now let's say you preview the system and you want to figure out, you know, how well did each of these – how much did each of these components actually contribute? Maybe you want to write a research paper and claim this was the piece that made the big difference. Can you actually document that claim and justify it?

So in ablative analysis, here's what we do. So in this example, let's say that simple logistic regression without any of your clever improvements get 94 percent performance. And you want to figure out what accounts for your improvement from 94 to 99.9 percent

performance. So in ablative analysis and so instead of adding components one at a day, we'll instead remove components one at a time to see how it rates.

So start with your overall system, which is 99 percent accuracy. And then we remove spelling correction and see how much performance drops. Then we'll remove the sender host features and see how much performance drops, and so on. All right? And so, in this contrived example, you see that, I guess, the biggest drop occurred when you remove the text parser features. And so you can then make a credible case that, you know, the text parser features where what really made the biggest difference here. Okay?

And you can also tell, for instance, that, I don't know, removing the sender host features on this line, right, performance dropped from 99.9 to 98.9. And so this also means that in case you want to get rid of the sender host features to speed up computational something that would be a good candidate for elimination. Okay?

Student: Are there any guarantees that if you shuffle around the order in which you drop those features that you'll get the same –

Instructor (Andrew Ng):Yeah. Let's address the question: What if you shuffle in which you remove things? The answer is no. There's no guarantee you'd get the similar result. So in practice, sometimes there's a fairly natural of ordering for both types of analyses, the error analyses and ablative analysis, sometimes there's a fairly natural ordering in which you add things or remove things, sometimes there's isn't. And quite often, you either choose one ordering and just go for it or –

And don't think of these analyses as sort of formulas that are constants, though; I mean feel free to invent your own, as well. You know one of the things that's done quite often is take the overall system and just remove one and then put it back, then remove a different one then put it back until all of these things are done. Okay.

So the very last thing I want to talk about is sort of this general advice for how to get started on a learning problem. So here's a cartoon description on two broad to get started on learning problem. The first one is carefully design your system, so you spend a long time designing exactly the right features, collecting the right data set, and designing the right algorithmic structure, then you implement it and hope it works. All right?

The benefit of this sort of approach is you get maybe nicer, maybe more scalable algorithms, and maybe you come up with new elegant learning algorithms. And if your goal is to, you know, contribute to basic research in machine learning, if your goal is to invent new machine learning algorithms, this process of slowing down and thinking deeply about the problem, you know, that is sort of the right way to go about is think deeply about a problem and invent new solutions.

Second sort of approach is what I call build-and-fix, which is we input something quick and dirty and then you run error analyses and diagnostics to figure out what's wrong and you fix those errors.

The benefit of this second type of approach is that it'll often get your application working much more quickly. And especially with those of you, if you end up working in a company, and sometimes – if you end up working in a company, you know, very often it's not the best product that wins; it's the first product to market that wins. And so there's – especially in the industry. There's really something to be said for, you know, building a system quickly and getting it deployed quickly.

And the second approach of building a quick-and-dirty, I'm gonna say hack and then fixing the problems will actually get you to a system that works well much more quickly. And the reason is very often it's really not clear what parts of a system are easier to think of to build and therefore what you need to spends lot of time focusing on.

So there's that example I talked about just now. Right? For identifying people, say. And with a big complicated learning system like this, a big complicated pipeline like this, it's really not obvious at the outset which of these components you should spend lots of time working on. Right? And if you didn't know that preprocessing wasn't the right component, you could easily have spent three months working on better background subtraction, not knowing that it's just not gonna ultimately matter.

And so the only way to find out what really works was inputting something quickly and you find out what parts – and find out what parts are really the hard parts to implement, or what parts are hard parts that could make a difference in performance. In fact, say that if your goal is to build a people recognition system, a system like this is actually far too complicated as your initial system. Maybe after you're prototyped a few systems, and you converged a system like this. But if this is your first system you're designing, this is much too complicated.

Also, this is a very concrete piece of advice, and this applies to your projects as well. If your goal is to build a working application, Step 1 is actually probably not to design a system like this. Step 1 is where you would plot your data. And very often, and if you just take the data you're trying to predict and just plot your data, plot X, plot Y, plot your data everywhere you can think of, you know, half the time you look at it and go, "Gee, how come all those numbers are negative? I thought they should be positive. Something's wrong with this dataset."

And it's about half the time you find something obviously wrong with your data or something very surprising. And this is something you find out just by plotting your data, and that you won't find out be implementing these big complicated learning algorithms on it. Plotting the data sounds so simple, it was one of the pieces of advice that lots of us give but hardly anyone follows, so you can take that for what it's worth.

Let me just reiterate, what I just said here may be bad advice if your goal is to come up with new machine learning algorithms. All right? So for me personally, the learning algorithm I use the most often is probably logistic regression because I have code lying around. So give me a learning problem, I probably won't try anything more complicated

than logistic regression on it first. And it's only after trying something really simple and figure our what's easy, what's hard, then you know where to focus your efforts.

But again, if your goal is to invent new machine learning algorithms, then you sort of don't want to hack up something and then add another hack to fix it, and hack it even more to fix it. Right? So if your goal is to do novel machine learning research, then it pays to think more deeply about the problem and not gonna follow this specifically.

Shoot, you know what? All right, sorry if I'm late but I just have two more slides so I'm gonna go through these quickly. And so, this is what I think of as premature statistical optimization, where quite often, just like premature optimization of code, quite often people will prematurely optimize one component of a big complicated machine learning system. Okay?

Just two more slides. This was – this is a sort of cartoon that highly influenced my own thinking. It was based on a paper written by Christos Papadimitriou. This is how progress – this is how developmental progress of research often happens. Right? Let's say you want to build a mail delivery robot, so I've drawn a circle there that says mail delivery robot. And it seems like a useful thing to have. Right? You know free up people, don't have to deliver mail.

So what – to deliver mail, obviously you need a robot to wander around indoor environments and you need a robot to manipulate objects and pickup envelopes. And so, you need to build those two components in order to get a mail delivery robot. And so I've drawing those two components and little arrows to denote that, you know, obstacle avoidance is needed or would help build your mail delivery robot.

Well for obstacle for avoidance, clearly, you need a robot that can navigate and you need to detect objects so you can avoid the obstacles. Now we're gonna use computer vision to detect the objects. And so, we know that, you know, lighting sometimes changes, right, depending on whether it's the morning or noontime or evening. This is lighting causes the color of things to change, and so you need an object detection system that's invariant to the specific colors of an object. Right? Because lighting changes, say.

Well color, or RGB values, is represented by three-dimensional vectors. And so you need to learn when two colors might be the same thing, when two, you know, visual appearance of two colors may be the same thing as just the lighting change or something. And to understand that properly, we can go out and study differential geometry of 3d manifolds because that helps us build a sound theory on which to develop our 3d similarity learning algorithms.

And to really understand the fundamental aspects of this problem, we have to study the complexity of non-Riemannian geometries. And on and on it goes until eventually you're proving convergence bounds for sampled of non-monotonic logic. I don't even know what this is because I just made it up. Whereas in reality, you know, chances are that link isn't real. Color variance just barely helped object recognition maybe. I'm making this

up. Maybe differential geometry was hardly gonna help 3d similarity learning and that link's also gonna fail. Okay?

So, each of these circles can represent a person, or a research community, or a thought in your head. And there's a very real chance that maybe there are all these papers written on differential geometry of 3d manifolds, and they are written because some guy once told someone else that it'll help 3d similarity learning.

And, you know, it's like "A friend of mine told me that color invariance would help in object recognition, so I'm working on color invariance. And now I'm gonna tell a friend of mine that his thing will help my problem. And he'll tell a friend of his that his thing will help with his problem." And pretty soon, you're working on convergence bound for sampled non-monotonic logic, when in reality none of these will see the light of day of your mail delivery robot. Okay?

I'm not criticizing the role of theory. There are very powerful theories, like the theory of VC dimension, which is far, far, far to the right of this. So VC dimension is about as theoretical as it can get. And it's clearly had a huge impact on many applications. And there's, you know, dramatically advanced data machine learning. And another example is theory of NP-hardness as again, you know, is about theoretical as it can get. It's like a huge application on all of computer science, the theory of NP-hardness.

But when you are off working on highly theoretical things, I guess, to me personally it's important to keep in mind are you working on something like VC dimension, which is high impact, or are you working on something like convergence bound for sampled non-monotonic logic, which you're only hoping has some peripheral relevance to some application. Okay?

For me personally, I tend to work on an application only if I – excuse me. For me personally, and this is a personal choice, I tend to trust something only if I personally can see a link from the theory I'm working on all the way back to an application. And if I don't personally see a direct link from what I'm doing to an application then, you know, then that's fine. Then I can choose to work on theory, but I wouldn't necessarily trust that what the theory I'm working on will relate to an application, if I don't personally see a link all the way back.

Just to summarize. One lesson to take away from today is I think time spent coming up with diagnostics for learning algorithms is often time well spent. It's often up to your own ingenuity to come up with great diagnostics. And just when I personally, when I work on machine learning algorithm, it's not uncommon for me to be spending like between a third and often half of my time just writing diagnostics and trying to figure out what's going right and what's going on.

Sometimes it's tempting not to, right, because you want to be implementing learning algorithms and making progress. You don't want to be spending all this time, you know, implementing tests on your learning algorithms; it doesn't feel like when you're doing

anything. But when I implement learning algorithms, at least a third, and quite often half of my time, is actually spent implementing those tests and you can figure out what to work on. And I think it's actually one of the best uses of your time.

Talked about error analyses and ablative analyses, and lastly talked about, you know, different approaches and the risks of premature statistical optimization. Okay.

Sorry I ran you over. I'll be here for a few more minutes for your questions. That's [inaudible] today.

[End of Audio]

Duration: 82 minutes