MachineLearning-Lecture03

Instructor (Andrew Ng):Okay. Good morning and welcome back to the third lecture of this class. So here's what I want to do today, and some of the topics I do today may seem a little bit like I'm jumping, sort of, from topic to topic, but here's, sort of, the outline for today and the illogical flow of ideas. In the last lecture, we talked about linear regression and today I want to talk about sort of an adaptation of that called locally weighted regression. It's very a popular algorithm that's actually one of my former mentors probably favorite machine learning algorithm.

We'll then talk about a probable second interpretation of linear regression and use that to move onto our first classification algorithm, which is logistic regression; take a brief digression to tell you about something called the perceptron algorithm, which is something we'll come back to, again, later this quarter; and time allowing I hope to get to Newton's method, which is an algorithm for fitting logistic regression models.

So this is recap where we're talking about in the previous lecture, remember the notation I defined was that I used this X superscript I, Y superscript I to denote the I training example. And when we're talking about linear regression or linear least squares, we use this to denote the predicted value of "by my hypothesis H" on the input XI. And my hypothesis was franchised by the vector of grams as theta and so we said that this was equal to some from theta J, si J, and more theta transpose X. And we had the convention that X subscript Z is equal to one so this accounts for the intercept term in our linear regression model. And lowercase n here was the notation I was using for the number of features in my training set. Okay? So in the example when trying to predict housing prices, we had two features, the size of the house and the number of bedrooms. We had two features and there was – little n was equal to two. So just to finish recapping the previous lecture, we defined this quadratic cos function J of theta equals one-half, something I equals one to m, theta of XI minus YI squared where this is the sum over our m training examples and my training set. So lowercase m was the notation I've been using to denote the number of training examples I have and the size of my training set. And at the end of the last lecture, we derive the value of theta that minimizes this enclosed form, which was X transpose X inverse X transpose Y. Okay?

So as we move on in today's lecture, I'll continue to use this notation and, again, I realize this is a fair amount of notation to all remember, so if partway through this lecture you forgot – if you're having trouble remembering what lowercase m is or what lowercase n is or something please raise your hand and ask. When we talked about linear regression last time we used two features. One of the features was the size of the houses in square feet, so the living area of the house, and the other feature was the number of bedrooms in the house. In general, we apply a machine-learning algorithm to some problem that you care about. The choice of the features will very much be up to you, right? And the way you choose your features to give the learning algorithm will often have a large impact on how it actually does. So just for example, the choice we made last time was X1 equal this size, and let's leave this idea of the feature of the number of bedrooms for now, let's say we don't have data that tells us how many bedrooms are in these houses. One thing you

could do is actually define – oh, let's draw this out. And so, right? So say that was the size of the house and that's the price of the house. So if you use this as a feature maybe you get theta zero plus theta 1, X1, this, sort of, linear model. If you choose – let me just copy the same data set over, right? You can define the set of features where X1 is equal to the size of the house and X2 is the square of the size of the house. Okay? So X1 is the size of the house in say square footage and X2 is just take whatever the square footage of the house is and just square that number, and this would be another way to come up with a feature, and if you do that then the same algorithm will end up fitting a quadratic function for you. Theta 2, XM squared. Okay? Because this is actually X2. And depending on what the data looks like, maybe this is a slightly better fit to the data.

You can actually take this even further, right? Which is – let's see. I have seven training examples here, so you can actually maybe fit up to six for the polynomial. You can actually fill a model theta zero plus theta one, X1 plus theta two, X squared plus up to theta six. X to the power of six and theta six are the polynomial to these seven data points. And if you do that you find that you come up with a model that fits your data exactly. This is where, I guess, in this example I drew, we have seven data points, so if you fit a six model polynomial you can, sort of, fit a line that passes through these seven points perfectly. And you probably find that the curve you get will look something like that. And on the one hand, this is a great model in a sense that it fits your training data perfectly. On the other hand, this is probably not a very good model in the sense that none of us seriously think that this is a very good predictor of housing prices as a function of the size of the house, right?

So we'll actually come back to this later. It turns out of the models we have here; I feel like maybe the quadratic model fits the data best. Whereas the linear model looks like there's actually a bit of a quadratic component in this data that the linear function is not capturing. So we'll actually come back to this a little bit later and talk about the problems associated with fitting models that are either too simple, use two small a set of features, or on the models that are too complex and maybe use too large a set of features. Just to give these a name, we call this the problem of underfitting and, very informally, this refers to a setting where there are obvious patterns that – where there are patterns in the data that the algorithm is just failing to fit. And this problem here we refer to as overfitting and, again, very informally, this is when the algorithm is fitting the idiosyncrasies of this specific data set, right? It just so happens that of the seven houses we sampled in Portland, or wherever you collect data from, that house happens to be a bit more expensive, that house happened on the less expensive, and by fitting six for the polynomial we're, sort of, fitting the idiosyncratic properties of this data set, rather than the true underlying trends of how housing prices vary as the function of the size of house. Okay?

So these are two very different problems. We'll define them more formally me later and talk about how to address each of these problems, but for now I hope you appreciate that there is this issue of selecting features. So if you want to just teach us the learning problems there are a few ways to do so. We'll talk about feature selection algorithms later this quarter as well. So automatic algorithms for choosing what features you use in a

regression problem like this. What I want to do today is talk about a class of algorithms called non-parametric learning algorithms that will help to alleviate the need somewhat for you to choose features very carefully. Okay? And this leads us into our discussion of locally weighted regression. And just to define the term, linear regression, as we've defined it so far, is an example of a parametric learning algorithm. Parametric learning algorithm is one that's defined as an algorithm that has a fixed number of parameters that fit to the data. Okay? So in linear regression we have a fix set of parameters theta, right? That must fit to the data. In contrast, what I'm gonna talk about now is our first non-parametric learning algorithm. The formal definition, which is not very intuitive, so I've replaced it with a second, say, more intuitive. The, sort of, formal definition of the non-parametric learning algorithm is that it's an algorithm where the number of parameters goes with M, with the size of the training set. And usually it's defined as a number of parameters grows linearly with the size of the training set. This is the formal definition. A slightly less formal definition is that the amount of stuff that your learning algorithm needs to keep around will grow linearly with the training sets or, in another way of saying it, is that this is an algorithm that we'll need to keep around an entire training set, even after learning. Okay? So don't worry too much about this definition. But what I want to do now is describe a specific non-parametric learning algorithm called locally weighted regression. Which also goes by a couple of other names – which also goes by the name of Loess for self-hysterical reasons. Loess is usually spelled L-O-E-S-S, sometimes spelled like that, too. I just call it locally weighted regression. So here's the idea. This will be an algorithm that allows us to worry a little bit less about having to choose features very carefully. So for my motivating example, let's say that I have a training site that looks like this, okay? So this is X and that's Y. If you run linear regression on this and you fit maybe a linear function to this and you end up with a more or less flat, straight line, which is not a very good fit to this data. You can sit around and stare at this and try to decide whether the features are used right. So maybe you want to toss in a quadratic function, but this isn't really quadratic either. So maybe you want to model this as a X plus X squared plus maybe some function of sin of X or something. You actually sit around and fiddle with features. And after a while you can probably come up with a set of features that the model is okay, but let's talk about an algorithm that you can use without needing to do that.

So if – well, suppose you want to evaluate your hypothesis H at a certain point with a certain query point low K is X. Okay? And let's say you want to know what's the predicted value of Y at this position of X, right? So for linear regression, what we were doing was we would fit theta to minimize sum over I, YI minus theta, transpose XI squared, and return theta transpose X. Okay? So that was linear regression. In contrast, in locally weighted linear regression you're going to do things slightly different. You're going to look at this point X and then I'm going to look in my data set and take into account only the data points that are, sort of, in the little vicinity of X. Okay? So we'll look at where I want to value my hypothesis. I'm going to look only in the vicinity of this point where I want to value my hypothesis, and then I'm going to take, let's say, just these few points, and I will apply linear regression to fit a straight line just to this sub-set of the data. Okay? I'm using this sub-term sub-set – well let's come back to that later. So we take this data set and I fit a straight line to it and maybe I get a straight line like that.

And what I'll do is then evaluate this particular value of straight line and that will be the value I return for my algorithm. I think this would be the predicted value for – this would be the value of then my hypothesis outputs in locally weighted regression. Okay?

So we're gonna fall one up. Let me go ahead and formalize that. In locally weighted regression, we're going to fit theta to minimize sum over I to minimize that where these terms W superscript I are called weights. There are many possible choice for ways, I'm just gonna write one down. So this E's and minus, XI minus X squared over two. So let's look at what these weights really are, right? So notice that – suppose you have a training example XI. So that XI is very close to X. So this is small, right? Then if XI minus X is small, so if XI minus X is close to zero, then this is E's to the minus zero and E to the zero is one. So if XI is close to X, then WI will be close to one. In other words, the weight associated with the, I training example be close to one if XI and X are close to each other. Conversely if XI minus X is large then – I don't know, what would WI be?

Student: Zero.

Instructor (Andrew Ng):Zero, right. Close to zero. Right. So if XI is very far from X then this is E to the minus of some large number and E to the minus some large number will be close to zero. Okay? So the picture is, if I'm quarrying at a certain point X, shown on the X axis, and if my data set, say, look like that, then I'm going to give the points close to this a large weight and give the points far away a small weight. So for the points that are far away, WI will be close to zero. And so as if for the points that are far away, they will not contribute much at all to this summation, right? So I think this is sum over I of one times this quadratic term for points by points plus zero times this quadratic term for faraway points. And so the effect of using this weighting is that locally weighted linear regression fits a set of parameters theta, paying much more attention to fitting the points close by accurately. Whereas ignoring the contribution from faraway points. Okay? Yeah?

Student: Your Y is exponentially [inaudible]?

Instructor (Andrew Ng):Yeah. Let's see. So it turns out there are many other weighting functions you can use. It turns out that there are definitely different communities of researchers that tend to choose different choices by default. There is somewhat of a literature on debating what point – exactly what function to use. This, sort of, exponential decay function is – this happens to be a reasonably common one that seems to be a more reasonable choice on many problems, but you can actually plug in other functions as well. Did I mention what [inaudible] is it at? For those of you that are familiar with the normal distribution, or the Gaussian distribution, say this – what this formula I've written out here, it cosmetically looks a bit like a Gaussian distribution. Okay? But this actually has absolutely nothing to do with Gaussian distribution. So this is not that a problem with XI is Gaussian or whatever. This is no such interpretation. This is just a convenient function that happens to be a bell-shaped function, but don't endow this of any Gaussian semantics. Okay?

So, in fact – well, if you remember the familiar bell-shaped Gaussian, again, it's just the ways of associating with these points is that if you imagine putting this on a bell-shaped bump, centered around the position of where you want to value your hypothesis H, then there's a saying this point here I'll give a weight that's proportional to the height of the Gaussian – excuse me, to the height of the bell-shaped function evaluated at this point. And the way to get to this point will be, to this training example, will be proportionate to that height and so on. Okay? And so training examples that are really far away get a very small weight.

One last small generalization to this is that normally there's one other parameter to this algorithm, which I'll denote as tow. Again, this looks suspiciously like the variants of a Gaussian, but this is not a Gaussian. This is a convenient form or function. This parameter tow is called the bandwidth parameter and informally it controls how fast the weights fall of with distance. Okay? So just copy my diagram from the other side, I guess. So if tow is very small, if that's a query X, then you end up choosing a fairly narrow Gaussian – excuse me, a fairly narrow bell shape, so that the weights of the points are far away fall off rapidly. Whereas if tow is large then you'd end up choosing a weighting function that falls of relatively slowly with distance from your query. Okay?

So I hope you can, therefore, see that if you apply locally weighted linear regression to a data set that looks like this, then to ask what your hypothesis output is at a point like this you end up having a straight line making that prediction. To ask what kind of class this [inaudible] at that value you put a straight line there and you predict that value. It turns out that every time you try to vary your hypothesis, every time you ask your learning algorithm to make a prediction for how much a new house costs or whatever, you need to run a new fitting procedure and then evaluate this line that you fit just at the position of the value of X. So the position of the query where you're trying to make a prediction. Okay? But if you do this for every point along the X-axis then you find that locally weighted regression is able to trace on this, sort of, very non-linear curve for a data set like this. Okay?

So in the problem set we're actually gonna let you play around more with this algorithm. So I won't say too much more about it here. But to finally move on to the next topic let me check the questions you have. Yeah?

Student: It seems like you still have the same problem of overfitting and underfitting, like when you had a Q's tow. Like you make it too small in your –

Instructor (Andrew Ng):Yes, absolutely. Yes. So locally weighted regression can run into – locally weighted regression is not a penancier for the problem of overfitting or underfitting. You can still run into the same problems with locally weighted regression. What you just said about – and so some of these things I'll leave you to discover for yourself in the homework problem. You'll actually see what you just mentioned. Yeah?

Student: It almost seems like you're not even thoroughly [inaudible] with this locally weighted, you had all the data that you originally had anyway.

Instructor (Andrew Ng):Yeah.

Student:I'm just trying to think of [inaudible] the original data points.

Instructor (Andrew Ng):Right. So the question is, sort of, this – it's almost as if you're not building a model, because you need the entire data set. And the other way of saying that is that this is a non-parametric learning algorithm. So this –I don't know. I won't debate whether, you know, are we really building a model or not. But this is a perfectly fine – so if I think when you write a code implementing locally weighted linear regression on the data set I think of that code as a whole – as building your model. So it actually uses – we've actually used this quite successfully to model, sort of, the dynamics of this autonomous helicopter this is. Yeah?

Student:I ask if this algorithm that learn the weights based on the data?

Instructor (Andrew Ng):Learn what weights? Oh, the weights WI.

Student: Instead of using [inaudible].

Instructor (Andrew Ng):I see, yes. So it turns out there are a few things you can do. One thing that is quite common is how to choose this band with parameter tow, right? As using the data. We'll actually talk about that a bit later when we talk about model selection. Yes? One last question.

Student:I used [inaudible] Gaussian sometimes if you [inaudible] Gaussian and then –

Instructor (Andrew Ng):Oh, I guess. Lt's see. Boy. The weights are not random variables and it's not, for the purpose of this algorithm, it is not useful to endow it with probable semantics. So you could choose to define things as Gaussian, but it, sort of, doesn't lead anywhere. In fact, it turns out that I happened to choose this, sort of, bell-shaped function to define my weights. It's actually fine to choose a function that doesn't even integrate to one, that integrates to infinity, say, as you're weighting function. So in that sense, I mean, you could force in the definition of a Gaussian, but it's, sort of, not useful. Especially since you use other functions that integrate to infinity and don't integrate to one. Okay? It's the last question and let's move on

Student: Assume that we have a very huge [inaudible], for example. A very huge set of houses and want to predict the linear for each house and so should the end result for each input – I'm seeing this very constantly for –

Instructor (Andrew Ng):Yes, you're right. So because locally weighted regression is a non-parametric algorithm every time you make a prediction you need to fit theta to your entire training set again. So you're actually right. If you have a very large training set then this is a somewhat expensive algorithm to use. Because every time you want to make a prediction you need to fit a straight line to a huge data set again. Turns out there are algorithms that – turns out there are ways to make this much more efficient for large

data sets as well. So don't want to talk about that. If you're interested, look up the work of Andrew Moore on KD-trees. He, sort of, figured out ways to fit these models much more efficiently. That's not something I want to go into today. Okay? Let me move one. Let's take more questions later.

So, okay. So that's locally weighted regression. Remember the outline I had, I guess, at the beginning of this lecture. What I want to do now is talk about a probabilistic interpretation of linear regression, all right? And in particular of the – it'll be this probabilistic interpretation that let's us move on to talk about logistic regression, which will be our first classification algorithm. So let's put aside locally weighted regression for now. We'll just talk about ordinary unweighted linear regression. Let's ask the question of why least squares, right? Of all the things we could optimize how do we come up with this criteria for minimizing the square of the area between the predictions of the hypotheses and the values Y predicted. So why not minimize the absolute value of the areas or the areas to the power of four or something? What I'm going to do now is present one set of assumptions that will serve to "justify" why we're minimizing the sum of square zero. Okay?

It turns out that there are many assumptions that are sufficient to justify why we do least squares and this is just one of them. So just because I present one set of assumptions under which least squares regression make sense, but this is not the only set of assumptions. So even if the assumptions I describe don't hold, least squares actually still makes sense in many circumstances. But this sort of new help, you know, give one rationalization, like, one reason for doing least squares regression. And, in particular, what I'm going to do is endow the least squares model with probabilistic semantics. So let's assume in our example of predicting housing prices, that the price of the house it's sold four, and there's going to be some linear function of the features, plus some term epsilon I. Okay? And epsilon I will be our error term. You can think of the error term as capturing unmodeled effects, like, that maybe there's some other features of a house, like, maybe how many fireplaces it has or whether there's a garden or whatever, that there are additional features that we jut fail to capture or you can think of epsilon as random noise. Epsilon is our error term that captures both these unmodeled effects. Just things we forgot to model. Maybe the function isn't quite linear or something. As well as random noise, like maybe that day the seller was in a really bad mood and so he sold it, just refused to go for a reasonable price or something. And now I will assume that the errors have a probabilistic – have a probability distribution. I'll assume that the errors epsilon I are distributed just till they denote epsilon I is distributive according to a probability distribution. That's a Gaussian distribution with mean zero and variance sigma squared. Okay? So let me just scripts in here, n stands for normal, right? To denote a normal distribution, also known as the Gaussian distribution, with mean zero and covariance sigma squared.

Actually, just quickly raise your hand if you've seen a Gaussian distribution before. Okay, cool. Most of you. Great. Almost everyone. So, in other words, the density for Gaussian is what you've seen before. The density for epsilon I would be one over root 2 pi sigma, E to the negative, epsilon I squared over 2 sigma squared, right? And the

density of our epsilon I will be this bell-shaped curve with one standard deviation being a, sort of, sigma. Okay? This is form for that bell-shaped curve. So, let's see. I can erase that. Can I erase the board? So this implies that the probability distribution of a price of a house given in si and the parameters theta, that this is going to be Gaussian with that density. Okay? In other words, saying goes as that the price of a house given the features of the house and my parameters theta, this is going to be a random variable that's distributed Gaussian with mean theta transpose XI and variance sigma squared. Right? Because we imagine that the way the housing prices are generated is that the price of a house is equal to theta transpose XI and then plus some random Gaussian noise with variance sigma squared. So the price of a house is going to have mean theta transpose XI, again, and sigma squared, right? Does this make sense? Raise your hand if this makes sense. Yeah, okay. Lots of you.

In point of notation – oh, yes?

Student: Assuming we don't know anything about the error, why do you assume here the error is a Gaussian?

Instructor (Andrew Ng):Right. So, boy. Why do I see the error as Gaussian? Two reasons, right? One is that it turns out to be mathematically convenient to do so and the other is, I don't know, I can also mumble about justifications, such as things to the central limit theorem. It turns out that if you, for the vast majority of problems, if you apply a linear regression model like this and try to measure the distribution of the errors, not all the time, but very often you find that the errors really are Gaussian. That this Gaussian model is a good assumption for the error in regression problems like these. Some of you may have heard of the central limit theorem, which says that the sum of many independent random variables will tend towards a Gaussian. So if the error is caused by many effects, like the mood of the seller, the mood of the buyer, some other features that we miss, whether the place has a garden or not, and if all of these effects are independent, then by the central limit theorem you might be inclined to believe that the sum of all these effects will be approximately Gaussian. If in practice, I guess, the two real answers are that, 1.) In practice this is actually a reasonably accurate assumption, and 2.) Is it turns out to be mathematically convenient to do so. Okay? Yeah?

Student: It seems like we're saying if we assume that area around model has zero mean, then the area is centered around our model. Which it seems almost like we're trying to assume what we're trying to prove. Instructor?

That's the [inaudible] but, yes. You are assuming that the error has zero mean. Which is, yeah, right. I think later this quarter we get to some of the other things, but for now just think of this as a mathematically – it's actually not an unreasonable assumption. I guess, in machine learning all the assumptions we make are almost never true in the absence sense, right? Because, for instance, housing prices are priced to dollars and cents, so the error will be – errors in prices are not continued as value random variables, because houses can only be priced at a certain number of dollars and a certain number of cents and you never have fractions of cents in housing prices. Whereas a Gaussian random

variable would. So in that sense, assumptions we make are never “absolutely true,” but for practical purposes this is a accurate enough assumption that it’ll be useful to make. Okay? I think in a week or two, we’ll actually come back to selected more about the assumptions we make and when they help our learning algorithms and when they hurt our learning algorithms. We’ll say a bit more about it when we talk about generative and discriminative learning algorithms, like, in a week or two. Okay?

So let’s point out one bit of notation, which is that when I wrote this down I actually wrote P of YI given XI and then semicolon theta and I’m going to use this notation when we are not thinking of theta as a random variable. So in statistics, though, sometimes it’s called the frequentist’s point of view, where you think of there as being some, sort of, true value of theta that’s out there that’s generating the data say, but we don’t know what theta is, but theta is not a random vehicle, right? So it’s not like there’s some random value of theta out there. It’s that theta is – there’s some true value of theta out there. It’s just that we don’t know what the true value of theta is. So if theta is not a random variable, then I’m going to avoid writing P of YI given XI comma theta, because this would mean that probably of YI conditioned on X and theta and you can only condition on random variables. So at this part of the class where we’re taking sort of frequentist’s viewpoint rather than the Dasian viewpoint, in this part of class we’re thinking of theta not as a random variable, but just as something we’re trying to estimate and use the semicolon notation. So the way to read this is this is the probability of YI given XI and parameterized by theta. Okay? So you read the semicolon as parameterized by. And in the same way here, I’ll say YI given XI parameterized by theta is distributed Gaussian with that.

All right. So we’re gonna make one more assumption. Let’s assume that the error terms are IID, okay? Which stands for Independently and Identically Distributed. So it’s going to assume that the error terms are independent of each other, right? The identically distributed part just means that I’m assuming the outcome for the same Gaussian distribution or the same variance, but the more important part of is this is that I’m assuming that the epsilon I’s are independent of each other. Now, let’s talk about how to fit a model. The probability of Y given X parameterized by theta – I’m actually going to give this another name. I’m going to write this down and we’ll call this the likelihood of theta as the probability of Y given X parameterized by theta. And so this is going to be the product over my training set like that. Which is, in turn, going to be a product of those Gaussian densities that I wrote down just now, right? Okay?

Then in parts of notation, I guess, I define this term here to be the likelihood of theta. And the likely of theta is just the probability of the data Y, right? Given X and prioritized by theta. To test the likelihood and probability are often confused. So the likelihood of theta is the same thing as the probability of the data you saw. So likely and probably are, sort of, the same thing. Except that when I use the term likelihood I’m trying to emphasize that I’m taking this thing and viewing it as a function of theta. Okay? So likelihood and for probability, they’re really the same thing except that when I want to view this thing as a function of theta holding X and Y fix are then called likelihood. Okay? So hopefully you hear me say the likelihood of the parameters and the probability

of the data, right? Rather than the likelihood of the data or probability of parameters. So try to be consistent in that terminology.

So given that the probability of the data is this and this is also the likelihood of the parameters, how do you estimate the parameters theta? So given a training set, what parameters theta do you want to choose for your model? Well, the principle of maximum likelihood estimation says that, right? You can choose the value of theta that makes the data as probable as possible, right? So choose theta to maximize the likelihood. Or in other words choose the parameters that make the data as probable as possible, right? So this is massive likely your estimation from six to six. So it's choose the parameters that makes it as likely as probable as possible for me to have seen the data I just did.

So for mathematical convenience, let me define lower case l of theta. This is called the log likelihood function and it's just log of capital L of theta. So this is log over product over I to find sigma E to that. I won't bother to write out what's in the exponent for now. It's just saying this from the previous board. Log and a product is the same as the sum of over logs, right? So it's a sum of the logs of – which simplifies to m times one over root two pi sigma plus and then log of explanation cancel each other, right? So if log of E of something is just whatever's inside the exponent. So, you know what, let me write this on the next board.

Okay. So maximizing the likelihood or maximizing the log likelihood is the same as minimizing that term over there. Well, you get it, right? Because there's a minus sign. So maximizing this because of the minus sign is the same as minimizing this as a function of theta. And this is, of course, just the same quadratic cos function that we had last time, J of theta, right? So what we've just shown is that the ordinary least squares algorithm, that we worked on the previous lecture, is just maximum likelihood assuming this probabilistic model, assuming IID Gaussian errors on our data. Okay?

One thing that we'll actually leave is that, in the next lecture notice that the value of sigma squared doesn't matter, right? That somehow no matter what the value of sigma squared is, I mean, sigma squared has to be a positive number. It's a variance of a Gaussian. So that no matter what sigma squared is since it's a positive number the value of theta we end up with will be the same, right? So because minimizing this you get the same value of theta no matter what sigma squared is. So it's as if in this model the value of sigma squared doesn't really matter. Just remember that for the next lecture. We'll come back to this again. Any questions about this? Actually, let me clean up another couple of boards and then I'll see what questions you have.

Okay. Any questions? Yeah?

Student: You are, I think here you try to measure the likelihood of your nice of theta by a fraction of error, but I think it's that you measure because it depends on the family of theta too, for example. If you have a lot of parameters [inaudible] or fitting in?

Instructor (Andrew Ng):Yeah, yeah. I mean, you're asking about overfitting, whether this is a good model. I think let's – the thing's you're mentioning are maybe deeper questions about learning algorithms that we'll just come back to later, so don't really want to get into that right now. Any more questions? Okay.

So this endows linear regression with a probabilistic interpretation. I'm actually going to use this probabil – use this, sort of, probabilistic interpretation in order to derive our next learning algorithm, which will be our first classification algorithm. Okay? So you'll recall that I said that regression problems are where the variable Y that you're trying to predict is continuous values. Now I'm actually gonna talk about our first classification problem, where the value Y you're trying to predict will be discreet value. You can take on only a small number of discrete values and in this case I'll talk about binding classification where Y takes on only two values, right? So you come up with classification problems if you're trying to do, say, a medical diagnosis and try to decide based on some features that the patient has a disease or does not have a disease. Or if in the housing example, maybe you're trying to decide will this house sell in the next six months or not and the answer is either yes or no. It'll either be sold in the next six months or it won't be. Other standing examples, if you want to build a spam filter. Is this e-mail spam or not? It's yes or no. Or if you, you know, some of my colleagues sit in whether predicting whether a computer system will crash. So you have a learning algorithm to predict will this computing cluster crash over the next 24 hours? And, again, it's a yes or no answer.

So there's X, there's Y. And in a classification problem Y takes on two values, zero and one. That's it in binding the classification. So what can you do? Well, one thing you could do is take linear regression, as we've described it so far, and apply it to this problem, right? So you, you know, given this data set you can fit a straight line to it. Maybe you get that straight line, right? But this data set I've drawn, right? This is an amazingly easy classification problem. It's pretty obvious to all of us that, right? The relationship between X and Y is – well, you just look at a value around here and it's the right is one, it's the left and Y is zero. So you apply linear regressions to this data set and you get a reasonable fit and you can then maybe take your linear regression hypothesis to this straight line and threshold it at 0.5. If you do that you'll certainly get the right answer. You predict that if X is to the right of, sort of, the mid-point here then Y is one and then next to the left of that mid-point then Y is zero.

So some people actually do this. Apply linear regression to classification problems and sometimes it'll work okay, but in general it's actually a pretty bad idea to apply linear regression to classification problems like these and here's why. Let's say I change my training set by giving you just one more training example all the way up there, right? Imagine if given this training set is actually still entirely obvious what the relationship between X and Y is, right? It's just – take this value as greater than Y is one and it's less then Y is zero. By giving you this additional training example it really shouldn't change anything. I mean, I didn't really convey much new information. There's no surprise that this corresponds to Y equals one. But if you now fit linear regression to this data set you end up with a line that, I don't know, maybe looks like that, right? And now the

predictions of your hypothesis have changed completely if your threshold – your hypothesis at Y equal both 0.5. Okay? So –

Student: In between there might be an interval where it's zero, right? For that far off point?

Instructor (Andrew Ng):Oh, you mean, like that?

Student: Right.

Instructor (Andrew Ng):Yeah, yeah, fine. Yeah, sure. A theta set like that so. So, I guess, these just – yes, you're right, but this is an example and this example works. This –

Student:[Inaudible] that will change it even more if you gave it all –

Instructor (Andrew Ng):Yeah. Then I think this actually would make it even worse. You would actually get a line that pulls out even further, right? So this is my example. I get to make it whatever I want, right? But the point of this is that there's not a deep meaning to this. The point of this is just that it could be a really bad idea to apply linear regression to classification algorithm. Sometimes it work fine, but usually I wouldn't do it. So a couple of problems with this. One is that, well – so what do you want to do for classification? If you know the value of Y lies between zero and one then to kind of fix this problem let's just start by changing the form of our hypothesis so that my hypothesis always lies in the unit interval between zero and one. Okay? So if I know Y is either zero or one then let's at least not have my hypothesis predict values much larger than one and much smaller than zero. And so I'm going to – instead of choosing a linear function for my hypothesis I'm going to choose something slightly different. And, in particular, I'm going to choose this function, H subscript theta of X is going to equal to G of theta transpose X where G is going to be this function and so this becomes more than one plus theta X of theta transpose X. And G of Z is called the sigmoid function and it is often also called the logistic function. It goes by either of these names.

And what G of Z looks like is the following. So when you have your horizontal axis I'm going to plot Z and so G of Z will look like this. Okay? I didn't draw that very well. Okay. So G of Z tends towards zero as Z becomes very small and G of Z will ascend towards one as Z becomes large and it crosses the vertical axis at 0.5. So this is what sigmoid function, also called the logistic function of. Yeah? Question?

Student: What sort of sigmoid in other step five?

Instructor (Andrew Ng):Say that again.

Student: Why we cannot chose this at five for some reason, like, that's better binary.

Instructor (Andrew Ng):Yeah. Let me come back to that later. So it turns out that Y – where did I get this function from, right? I just wrote down this function. It actually turns out that there are two reasons for using this function that we'll come to. One is – we talked about generalized linear models. We'll see that this falls out naturally as part of the broader class of models. And another reason that we'll talk about next week, it turns out there are a couple of, I think, very beautiful reasons for why we choose logistic functions. We'll see that in a little bit. But for now let me just define it and just take my word for it for now that this is a reasonable choice. Okay? But notice now that my – the values output by my hypothesis will always be between zero and one. Furthermore, just like we did for linear regression, I'm going to endow the outputs and my hypothesis with a probabilistic interpretation, right? So I'm going to assume that the probability that Y is equal to one given X and parameterized by theta that's equal to H subscript theta of X, all right? So in other words I'm going to imagine that my hypothesis is outputting all these numbers that lie between zero and one. I'm going to think of my hypothesis as trying to estimate the probability that Y is equal to one. Okay? And because Y has to be either zero or one then the probability of Y equals zero is going to be that. All right? So more simply it turns out – actually, take these two equations and write them more compactly. Write P of Y given X parameterized by theta. This is going to be H subscript theta of X to the power of Y times one minus H of X to the power of one minus Y. Okay?

So I know this looks somewhat bizarre, but this actually makes the variation much nicer. So Y is equal to one then this equation is H of X to the power of one times something to the power of zero. So anything to the power of zero is just one, right? So Y equals one then this is something to the power of zero and so this is just one. So if Y equals one this is just saying P of Y equals one is equal to H subscript theta of X. Okay? And in the same way, if Y is equal to zero then this is P of Y equals zero equals this thing to the power of zero and so this disappears. This is just one times this thing power of one. Okay? So this is a compact way of writing both of these equations to gather them to one line. So let's hope our parameter fitting, right? And, again, you can ask – well, given this model by data, how do I fit the parameters theta of my model? So the likelihood of the parameters is, as before, it's just the probability of theta, right? Which is product over I, PFYI given XI parameterized by theta. Which is – just plugging those in. Okay? I dropped this theta subscript just so you can write a little bit less. Oh, excuse me. These should be XI's and YI's. Okay?

So, as before, let's say we want to find a maximum likelihood estimate of the parameters theta. So we want to find the – setting the parameters theta that maximizes the likelihood L of theta. It turns out that very often – just when you work with the derivations, it turns out that it is often much easier to maximize the log of the likelihood rather than maximize the likelihood. So the log likelihood L of theta is just log of capital L. This will, therefore, be sum of this. Okay? And so to fit the parameters theta of our model we'll find the value of theta that maximizes this log likelihood. Yeah?

Student:[Inaudible]

Instructor (Andrew Ng):Say that again.

Student:YI is [inaudible].

Instructor (Andrew Ng):Oh, yes. Thanks. So having maximized this function – well, it turns out we can actually apply the same gradient descent algorithm that we learned. That was the first algorithm we used to minimize the quadratic function. And you remember, when we talked about least squares, the first algorithm we used to minimize the quadratic error function was great in descent. So can actually use exactly the same algorithm to maximize the log likelihood. And you remember, that algorithm was just repeatedly take the value of theta and you replace it with the previous value of theta plus a learning rate alpha times the gradient of the cos function. The log likelihood will respect the theta. Okay? One small change is that because previously we were trying to minimize the quadratic error term. Today we're trying to maximize rather than minimize. So rather than having a minus sign we have a plus sign. So this is just great in ascents, but for the maximization rather than the minimization. So we actually call this gradient ascent and it's really the same algorithm.

So to figure out what this gradient – so in order to derive gradient descent, what you need to do is compute the partial derivatives of your objective function with respect to each of your parameters theta I, right? It turns out that if you actually compute this partial derivative – so you take this formula, this L of theta, which is – oh, got that wrong too. If you take this lower case l theta, if you take the log likelihood of theta, and if you take it's partial derivative with respect to theta I you find that this is equal to – let's see. Okay? And, I don't know, the derivation isn't terribly complicated, but in the interest of saving you watching me write down a couple of blackboards full of math I'll just write down the final answer. But the way you get this is you just take those, plug in the definition for F subscript theta as function of XI, and take derivatives, and work through the algebra it turns out it'll simplify down to this formula. Okay?

And so what that gives you is that gradient ascent is the following rule. Theta J gets updated as theta J plus alpha gives this. Okay? Does this look familiar to anyone? Did you remember seeing this formula at the last lecture? Right. So when I worked up Bastrian descent for least squares regression I, actually, wrote down exactly the same thing, or maybe there's a minus sign and this is also fit. But I, actually, had exactly the same learning rule last time for least squares regression, right? Is this the same learning algorithm then? So what's different? How come I was making all that noise earlier about least squares regression being a bad idea for classification problems and then I did a bunch of math and I skipped some steps, but I'm, sort of, claiming at the end they're really the same learning algorithm?

Student:[Inaudible] constants?

Instructor (Andrew Ng):Say that again.

Student:[Inaudible]

Instructor (Andrew Ng):Oh, right. Okay, cool.

Student: It's the lowest it –

Instructor (Andrew Ng):No, exactly. Right. So zero to the same, this is not the same, right? And the reason is, in logistic regression this is different from before, right? The definition of this H subscript theta of XI is not the same as the definition I was using in the previous lecture. And in particular this is no longer theta transpose XI. This is not a linear function anymore. This is a logistic function of theta transpose XI. Okay? So even though this looks cosmetically similar, even though this is similar on the surface, to the Bastrian descent rule I derived last time for least squares regression this is actually a totally different learning algorithm. Okay? And it turns out that there's actually no coincidence that you ended up with the same learning rule. We'll actually talk a bit more about this later when we talk about generalized linear models. But this is one of the most elegant generalized learning models that we'll see later. That even though we're using a different model, you actually ended up with what looks like the same learning algorithm and it's actually no coincidence. Cool.

One last comment as part of a sort of learning process, over here I said I take the derivatives and I ended up with this line. I didn't want to make you sit through a long algebraic derivation, but later today or later this week, please, do go home and look at our lecture notes, where I wrote out the entirety of this derivation in full, and make sure you can follow every single step of how we take partial derivatives of this log likelihood to get this formula over here. Okay? By the way, for those who are interested in seriously masking machine learning material, when you go home and look at the lecture notes it will actually be very easy for most of you to look through the lecture notes and read through every line and go yep, that makes sense, that makes sense, that makes sense, and, sort of, say cool. I see how you get this line. You want to make sure you really understand the material. My concrete suggestion to you would be to you to go home, read through the lecture notes, check every line, and then to cover up the derivation and see if you can derive this example, right? So in general, that's usually good advice for studying technical material like machine learning. Which is if you work through a proof and you think you understood every line, the way to make sure you really understood it is to cover it up and see if you can rederive the entire thing itself. This is actually a great way because I did this a lot when I was trying to study various pieces of machine learning theory and various proofs. And this is actually a great way to study because cover up the derivations and see if you can do it yourself without looking at the original derivation. All right.

I probably won't get to Newton's Method today. I just want to say – take one quick digression to talk about one more algorithm, which was the discussion sort of alluding to this earlier, which is the perceptron algorithm, right? So I'm not gonna say a whole lot about the perceptron algorithm, but this is something that we'll come back to later. Later this quarter we'll talk about learning theory. So in logistic regression we said that G of Z are, sort of, my hypothesis output values that were low numbers between zero and one. The question is what if you want to force G of Z to up the value to either zero one? So the perceptron algorithm defines G of Z to be this. So the picture is – or the cartoon is, rather than this sigmoid function. E of Z now looks like this step function that you were asking

about earlier. In saying this before, we can use H subscript theta of X equals G of theta transpose X. Okay? So this is actually – everything is exactly the same as before, except that G of Z is now the step function. It turns out there's this learning called the perceptron learning rule that's actually even the same as the classic gradient ascent for logistic regression. And the learning rule is given by this. Okay? So it looks just like the classic gradient ascent rule for logistic regression. So this is very different flavor of algorithm than least squares regression and logistic regression, and, in particular, because it outputs only values are either zero or one it turns out it's very difficult to endow this algorithm with probabilistic semantics. And this is, again, even though – oh, excuse me. Right there. Okay. And even though this learning rule looks, again, looks cosmetically very similar to what we have in logistics regression this is actually a very different type of learning rule than the others that were seen in this class. So because this is such a simple learning algorithm, right? It just computes theta transpose X and then you threshold and then your output is zero or one. This is – right. So these are a simpler algorithm than logistic regression, I think. When we talk about learning theory later in this class, the simplicity of this algorithm will let us come back and use it as a building block. Okay? But that's all I want to say about this algorithm for now.

Just for fun, the last thing I'll do today is show you a historical video with – that talks about the perceptron algorithm. This particular video comes from a video series titled The Machine that Changed The World and was produced WGBH Television in cooperation with the BBC, British Broadcasting Corporation, and it aired on PBS a few years ago. This shows you what machine learning used to be like. It's a fun clip on perceptron algorithm.

In the 1950's and 60's scientists built a few working perceptrons, as these artificial brains were called. He's using it to explore the mysterious problem of how the brain learns. This perceptron is being trained to recognize the difference between males and females. It is something that all of us can do easily, but few of us can explain how. To get a computer to do this it would involve working out many complex rules about faces and writing a computer program, but this perceptron was simply given lots and lots of examples, including some with unusual hairstyles. But when it comes to a beetle the computer looks at facial features and hair outline and takes longer to learn what it's told by Dr. Taylor. Andrew puts on his wig also causes a little part searching. After training on lots of examples, it's given new faces it has never seen and is able to successfully distinguish male from female. It has learned.

All right. Isn't that great? Okay. That's it for today. I'll see you guys at the next lecture.

[End of Audio]

Duration: 75 minutes