MachineLearning-Lecture06

Instructor (Andrew Ng):Okay, good morning. Welcome back. Just one quick announcement for today, which is that this next discussion section as far as for the TA's will mostly be on, sort of, a tutorial on Matlab and Octaves. So I know many of you already have program Matlab or Octave before, but in case not, and you want to, sort of, see along the tutorial on how direct terms and Matlab, please come to this next discussion section.

What I want to do today is continue our discussion of Naïve Bayes, which is the learning algorithm that I started to discuss in the previous lecture and talk about a couple of different event models in Naïve Bayes, and then I'll take a brief digression to talk about neural networks, which is something that I actually won't spend a lot of time on, and then I want to start to talk about support vector machines, and support vector machines is the learning algorithms, the supervised learning algorithm that many people consider the most effective, off-the-shelf supervised learning algorithm. That point of view is debatable, but there are many people that hold that point of view, and we'll start discussing that today, and this will actually take us a few lectures to complete.

So let's talk about Naïve Bayes. To recap from the previous lecture, I started off describing spam classification as the most [inaudible] example for Naïve Bayes in which we would create feature vectors like these, right, that correspond to words in a dictionary. And so, you know, based on what words appear in a piece of email were represented as a feature vector with ones and zeros in the corresponding places, and Naïve Bayes was a generative learning algorithm, and by that I mean it's an algorithm in which we model PFX given Y, and for Naïve Bayes, specifically, we modeled it as product from I equals one to N, PFXI given Y, and also we model PFY, and then we use Bayes Rule, right, to combine these two together, and so our predictions, when you give it a new piece of email you want to tell if it's spam or not spam, you predict RFX over Y, PFY given X, which by Bayes Rule is RFX over Y, PFX given Y, times BY, okay?

So this is Naïve Bayes, and just to draw attention to two things, one is that in this model, each of our features were zero, one, so indicating whether different words appear, and the length or the feature vector was, sort of, the length N of the feature vector was the number of words in the dictionary. So it might be on this version on the order of 50,000 words, say.

What I want to do now is describe two variations on this algorithm. The first one is the simpler one, which it's just a generalization to if XI takes on more values. So, you know, one thing that's commonly done is to apply Naïve Bayes to problems where some of these features, XI, takes on K values rather than just two values, and in that case, you actually build, sort of, a very similar model where PFX given Y is really the same thing, right, where now these are going to be multinomial probabilities rather than Bernoulli's because the XI's can, maybe, take on up to K values.

It turns out, the situation where – one situation where this arises very commonly is if you have a feature that's actually continuous valued, and you choose to dispertise it, and you choose to take a continuous value feature and dispertise it into a finite set of K values, and so it's a perfect example if you remember our very first supervised learning problem of predicting the price of houses. If you have the classification problem on these houses, so based on features of a house, and you want to predict whether or not the house will be sold in the next six months, say.

That's a classification problem, and once you use Naïve Bayes, then given a continuous value feature like the living area, you know, one pretty common thing to do would be take the continuous value living area and just dispertise it into a few – discreet buckets, and so depending on whether the living area of the house is less than 500 square feet or between 1,000 and 1500 square feet, and so on, or whether it's greater than 2,000 square feet, you choose the value of the corresponding feature, XI, to be one, two, three, or four, okay? So that was the first variation or generalization of Naïve Bayes I wanted to talk about. I should just check; are there questions about this? Okay. Cool. And so it turns out that in practice, it's fairly common to use about ten buckets to dispertise a continuous value feature. I drew four here only to save on writing.

The second and, sort of, final variation that I want to talk about for Naïve Bayes is a variation that's specific to classifying text documents, or, more generally, for classifying sequences. So the text document, like a piece of email, you can think of as a sequence of words and you can apply this, sort of, model I'm about to describe to classifying other sequences as well, but let me just focus on text, and here's the idea.

So the Naïve Bayes algorithm as I've described it so far, right, given a piece of email, we were representing it using this binary vector value representation, and one of the things that this loses, for instance, is the number of times that different words appear, all right? So, for example, if some word appears a lot of times, and you see the word, you know, "buy" a lot of times. You see the word "Viagra"; it seems to be a common email example. You see the word Viagra a ton of times in the email, it is more likely to be spam than it appears, I guess, only once because even once, I guess, is enough.

So let me just try a different, what's called an event model for Naïve Bayes that will take into account the number of times a word appears in the email, and to give this previous model a name as well this particular model for text classification is called the Multivariate Bernoulli Event Model. It's not a great name. Don't worry about what the name means. It refers to the fact that there are multiple Bernoulli random variables, but it's really – don't worry about what the name means.

In contrast, what I want to do now is describe a different representation for email in terms of the feature vector, and this is called the Multinomial Event Model, and, again, there is a rationale behind the name, but it's slightly cryptic, so don't worry about why it's called the Multinomial Event Model; it's just called that. And here's what we're gonna do, given a piece of email, I'm going to represent my email as a feature vector, and so my IF

training example, XI will be a feature vector, XI sub group one, XI sub group two, XI subscript NI where NI is equal to the number of words in this email, right?

So if one of my training examples is an email with 300 words in it, then I represent this email via a feature vector with 300 elements, and each of these elements of the feature vector – lets see. Let me just write this as X subscript J. These will be an index into my dictionary, okay? And so if my dictionary has 50,000 words, then each position in my feature vector will be a variable that takes on one of 50,000 possible values corresponding to what word appeared in the J position of my email, okay?

So, in other words, I'm gonna take all the words in my email and you have a feature vector that just says which word in my dictionary was each word in the email, okay? So a different definition for NI now, NI now varies and is different for every training example, and this XJ is now indexed into the dictionary. You know, the components of the feature vector are no longer binary random variables; they're these indices in the dictionary that take on a much larger set of values.

And so our generative model for this will be that the joint distribution over X and Y will be that, where again N is now the length of the email, all right? So the way to think about this formula is you imagine that there was some probably distribution over emails. There's some random distribution that generates the emails, and that process proceeds as follows: First, Y is chosen, first the class label. Is someone gonna send you spam email or not spam emails is chosen for us.

So first Y, the random variable Y, the class label of spam or not spam is generated, and then having decided whether they sent you spam or not spam, someone iterates over all 300 positions of the email, or 300 words that are going to compose them as email, and would generate words from some distribution that depends on whether they chose to send you spam or not spam. So if they sent you spam, they'll send you words – they'll tend to generate words like, you know, buy, and Viagra, and whatever at discounts, sale, whatever. And if somebody chose to send you not spam, then they'll send you, sort of, the more normal words you get in an email, okay?

So, sort of, just careful, right? XI here has a very different definition from the previous event model, and N has a very different definition from the previous event model. And so the parameters of this model are – let's see. Phi subscript K given Y equals one, which is the probability that, you know, conditioned on someone deciding to spend you spam, what's the probability that the next word they choose to email you in the spam email is going to be word K, and similarly, you know, sort of, same thing – well, I'll just write it out, I guess – and Phi Y and just same as before, okay?

So these are the parameters of the model, and given a training set, you can work out the maximum likelihood estimates of the parameters. So the maximum likelihood estimate of the parameters will be equal to – and now I'm gonna write one of these, you know, big indicator function things again. It'll be a sum over your training sets indicator whether that was spam times the sum over all the words in that email where N subscript I is the

number of words in email I in your training set, times indicator XIJ, SK times that I, okay?

So the numerator says sum over all your emails and take into account all the emails that had class label one, take into account only of the emails that were spam because if Y equals zero, then this is zero, and this would go away, and then times sum over all the words in your spam email, and it counts up the number of times you observed the word K in your spam emails. So, in other words, the numerator is look at all the spam emails in your training set and count up the total number of times the word K appeared in this email. The denominator then is sum over I into our training set of whenever one of your examples is spam, you know, sum up the length of that spam email, and so the denominator is the total length of all of the spam emails in your training set.

And so the ratio is just out of all your spam emails, what is the fraction of words in all your spam emails that were word K, and that's your estimate for the probability of the next piece of spam mail generating the word K in any given position, okay? At the end of the previous lecture, I talked about LaPlace smoothing, and so when you do that as well, you add one to the numerator and K to the denominator, and this is the LaPlace smoothed estimate of this parameter, okay? And similarly, you do the same thing for – and you can work out the estimates for the other parameters yourself, okay? So it's very similar. Yeah?

Student:I'm sorry. On the right on the top, I was just wondering what the X of I is, and what the N of –

Instructor (Andrew Ng):Right. So in this second event model, the definition for XI and the definition for N are different, right? So here – well, this is for one example XY. So here, N is the number of words in a given email, right? And if it's the I email subscripting then this N subscript I, and so N will be different for different training examples, and here XI will be, you know, these values from 1 to 50,000, and XI is essentially the identity of the Ith word in a given piece of email, okay? So that's why this is grouping, or this is a product over all the different words of your email of their probability the Ith word in your email, conditioned on Y. Yeah?

Student:[Off mic].

Instructor (Andrew Ng):Oh, no, actually, you know what, I apologize. I just realized that overload the notation, right, and I shouldn't have used K here. Let me use a different alphabet and see if that makes sense; does that make sense? Oh, you know what, I'm sorry. You're absolutely right. Thank you. All right. So in LaPlace smoothing, that shouldn't be K. This should be, you know, 50,000, if you have 50,000 words in your dictionary. Yeah, thanks. Great. I stole notation from the previous lecture and didn't translate it properly. So LaPlace smoothing, right, this is the number of possible values that the random variable XI can take on. Cool. Raise your hand if this makes sense? Okay. Some of you, are there more questions about this? Yeah.

Student: On LaPlace smoothing, the denominator and the plus A is the number of values that Y could take?

Instructor (Andrew Ng):Yeah, let's see. So LaPlace smoothing is a method to give you, sort of, hopefully, better estimates of their probability distribution over a multinomial, and so was I using X to Y in the previous lecture? So in trying to estimate the probability over a multinomial – I think X and Y are different. I think – was it X or Y? I think it was X, actually. Well – oh, I see, right, right. I think I was using a different definition for the random variable Y because suppose you have a multinomial random variable, X which takes on – let's use a different alphabet. Suppose you have a multinomial random variable X which takes on L different values, then the maximum likelihood estimate for the probability of X, PFX equals K, will be equal to, right, the number of observations. The maximum likelihood estimate for the probability of X being equal to K will be the number of observations of X equals K divided by the total number of observations of X, okay? So that's the maximum likelihood estimate. And to add LaPlace smoothing to this, you, sort of, add one to the numerator, and you add L to the denominator where L was the number of possible values that X can take on. So, in this case, this is a probability that X equals K, and X can take on 50,000 values if 50,000 is the length of your dictionary; it may be something else, but that's why I add 50,000 to the denominator. Are there other questions? Yeah.

Student: Is there a specific definition for a maximum likelihood estimation of a parameter? We've talked about it a couple times, and all the examples make sense, but I don't know what the, like, general formula for it is.

Instructor (Andrew Ng):I see. Yeah, right. So the definition of maximum likelihood – so the question is what's the definition for maximum likelihood estimate? So actually in today's lecture and the previous lecture when I talk about Gaussian Discriminant Analysis I was, sort of, throwing out the maximum likelihood estimates on the board without proving them. The way to actually work this out is to actually write down the likelihood.

So the way to figure out all of these maximum likelihood estimates is to write down the likelihood of the parameters, phi K given Y being zero, phi Y, right? And so given a training set, the likelihood, I guess, I should be writing log likelihood will be the log of the product of I equals one to N, PFXI, YI, you know, parameterized by these things, okay? Where PFXI, YI, right, is given by NI, PFX, YJ given YI. They are parameterized by – well, I'll just drop the parameters to write this more simply – oh, I just put it in – times PFYI, okay?

So this is my log likelihood, and so the way you get the maximum likelihood estimate of the parameters is you – so if given a fixed training set, given a set of fixed IYI's, you maximize this in terms of these parameters, and then you get the maximum likelihood estimates that I've been writing out. So in a previous section of today's lecture I wrote out some maximum likelihood estimates for the Gaussian Discriminant Analysis model, and for Naïve Bayes, and then this – I didn't prove them, but you get to, sort of, play with

that yourself in the homework problem as well and for one of these models, and you'll be able to verify that when you maximize the likelihood and maximize the log likelihood that hopefully you do get the same formulas as what I was drawing up on the board, but a way is to find the way these are derived is by maximizing this, okay? Cool.

All right. So that wraps up what I wanted to say about – oh, so that, more or less, wraps up what I wanted to say about Naïve Bayes, and it turns out that for text classification, the Naïve Bayes algorithm with this second event model, the last Naïve Bayes model I presented with the multinomial event model, it turns out that almost always does better than the first Naïve Bayes model I talked about when you're applying it to the specific case – to the specific of text classification, and one of the reasons is hypothesized for this is that this second model, the multinomial event model, takes into account the number of times a word appears in a document, whereas the former model doesn't.

I should say that in truth that actually turns out not to be completely understood why the latter model does better than the former one for text classification, and, sort of, researchers are still debating about it a little bit, but if you ever have a text classification problem, you know, Naïve Bayes Classify is probably not, by far, the best learning algorithm out there, but it is relatively straightforward to implement, and it's a very good algorithm to try if you have a text classification problem, okay? Still a question? Yeah.

Student: So the second model is still positioning a variant, right? It doesn't actually care where the words are.

Instructor (Andrew Ng):Yes, all right.

Student: And, I mean, X variable, if my model like you had exclamation in, does that usually do better if you have enough data?

Instructor (Andrew Ng):Yeah, so the question is, sort of, the second model, right? The second model, the multinomial event model actually doesn't care about the ordering of the words. You can shuffle all the words in the email, and it does exactly the same thing. So in natural language processing, there's actually another name; it's called a Unique Grand Model in natural language processing, and there's some other models like, sort of, say, higher order markup models that take into account some of the ordering of the words. It turns out that for text classification, the models like the bigram models or trigram models, I believe they do only very slightly better, if at all, but that's when you're applying them to text classification, okay?

All right. So the next thing I want to talk about is to start again to discussion of non-linear classifiers. So it turns out – well, and so the very first classification algorithm we talked about was logistic regression, which had the forming form for hypothesis, and you can think of this as predicting one when this estimator probability is greater or equal to 0.5 and predicting zero, right, when this is less than 0.5, and given a training set, right? Logistic regression will maybe do grade and descends or something or use Newton's method to find a straight line that reasonably separates the positive and negative classes.

But sometimes a data set just can't be separated by a straight line, so is there an algorithm that will let you start to learn these sorts of non-linear division boundaries? And so how do you go about getting a non-linear classifier? And, by the way, one cool result is that remember when I said – when we talked about generative learning algorithms, I said that if you assume Y given X is exponential family, right, with parameter A, and if you build a generative learning algorithm using this, right, plus one, if this is A to one. This is exponential family with natural parameter A to zero, right.

I think when we talked about Gaussian Discriminant Analysis, I said that if this holds true, then you end up with a logistic posterior. It actually turns out that a Naïve Bayes model actually falls into this as well. So the Naïve Bayes model actually falls into this exponential family as well, and, therefore, under the Naïve Bayes model, you're actually using this other linear classifier as well, okay?

So the question is how can you start to get non-linear classifiers? And I'm going to talk about one method today which is – and we started to talk about it very briefly which is taking a simpler algorithm like logistic regression and using it to build up to more complex non-linear classifiers, okay? So to motivate this discussion, I'm going to use the little picture – let's see. So suppose you have features X1, X2, and X3, and so by convention, I'm gonna follow our earlier convention that X0 is set to one, and so I'm gonna use a little diagram like this to denote our logistic regression unit, okay?

So think of a little picture like that, you know, this little circle as denoting a computation note that takes this input, you know, several features and then it outputs another number that's X subscript theta of X, given by a sigmoid function, and so this little computational unit – well, will have parameters theta.

Now, in order to get non-linear division boundaries, all we need to do – well, at least one thing to do is just come up with a way to represent hypotheses that can output non-linear division boundaries, right, and so this is – when you put a bunch of those little pictures that I drew on the previous board, you can then get what's called a Neural Network in which you think of having my features here and then I would feed them to say a few of these little sigmoidal units, and these together will feed into yet another sigmoidal unit, say, which will output my final output H subscript theta of X, okay? And just to give these things names, let me call the values output by these three intermediate sigmoidal units; let me call them A1, A2, A3.

And let me just be completely concrete about what this formula represents, right? So each of these units in the middle will have their own associated set of parameters, and so the value A1 will be computed as G of X transpose, and then some set of parameters, which I'll write as theta one, and similarly, A2 will be computed as G of X transpose theta two, and A3 will be G of X transpose, theta three, where G is the sigmoid function, all right? So G of Z, and then, finally, our hypothesis will output G of A transpose theta four, right? Where, you know, this A vector is a vector of A1, A2, A3. We can append another one to it at first if you want, okay?

Let me just draw up here this – I'm sorry about the cluttered board. And so H subscript theta of X, this is a function of all the parameters theta one through theta four, and so one way to learn parameters for this model is to write down the cost function, say, J of theta equals one-half sum from Y equals one to M, YI minus H subscript theta of XI squared, say. Okay, so that's our familiar quadratic cost function, and so one way to learn the parameters of an algorithm like this is to just use gradient interscent to minimize J of theta as a function of theta, okay? See, in the phi gradient descent to minimize this square area, which stated differently means you use gradient descent to make the predictions of your neural network as close as possible to what you observed as the labels in your training set, okay?

So it turns out green descent on this neural network is a specific name, the algorithm that implements grand descent is called back propagation, and so if you ever hear that all that means is – it just means gradient interscent on a cost function like this or a variation of this on the neural network that looks like that, and – well, this algorithm actually has some advantages and disadvantages, but let me actually show you. So, let's see.

One of the interesting things about the neural network is that you can look at what these intermediate notes are computing, right? So this neural network has what's called a hidden layer before you then have the output layer, and, more generally, you can actually have inputs feed into these computation units, feed into more layers of computation units, to even more layers, to more layers, and then finally you have an output layer at the end

And one cool thing you can do is look at all of these intermediate units, look at these units and what's called a hidden layer of the neural network. Don't worry about why it's called that. Look at computations of the hidden unit and ask what is the hidden unit computing the neural network? So to, maybe, get a better sense of neural networks might be doing, let me show you a video – I'm gonna switch to the laptop – this is made by a friend, Yann LeCun who's currently a professor at New York University. Can I show a video on the laptop?

So let me show you a video from Yann LeCun on a neural network that he developed for Hammerton Digit Recognition. There was one other thing he did in this neural network that I'm not gonna talk about called a Convolutional Neural Network that – well, his system is called LeNet, and let's see. Would you put on the laptop display? Hum, actually maybe if – or you can just put on the screen on the side; that would work too if the big screen isn't working. Let's see. I'm just trying to think, okay, how do I keep you guys entertained while we're waiting for the video to come on?

Well, let me say a few more things about neural network. So it turns out that when you write a quadratic cost function like I wrote down on the chalkboard just now, it turns out that unlike logistic regression, that will almost always respond to non-convex optimization problem, and so whereas for logistic regression if you run gradient descent or Newton's method or whatever, you converse the global optimer. This is not true for neural networks. In general, there are lots of local optimer and, sort of, much harder optimization problem.

So neural networks, if you're, sort of, familiar with them, and you're good at making design choices like what learning rate to use, and how many hidden units to use, and so on, you can, sort of, get them to be fairly effective, and there's, sort of, often ongoing debates about, you know, is this learning algorithm better, or is that learning algorithm better? The vast majority of machine learning researchers today seem to perceive support vector machines, which is what I'll talk about later, to be a much more effective off-the-shelf learning algorithm than neural networks. This point of view is contested a bit, but so neural networks are not something that I personally use a lot right now because there's a hard optimization problem and you should do so often verge, and it actually, sort of works. It, sort of, works reasonably well. It's just because this is fairly complicated, there's not an algorithm that I use commonly or that my friends use all time. Oh, cool.

So but let me just go and show you an example of neural network, which was for many years, you know, the most effective learning algorithm before support vector machines were invented. So here's Yann LeCun's video, and – well, there's actually audio on this too, the soundboard. So I'll just tell you what's happening. What you're seeing is a trained neural network, and this display where my mouse pointer is pointing, right, this big three there is the input to the neural network.

So you're showing the neural network this image, and it's trying to recognize what is this. The final answer output by the neural network is this number up here, right below where it says LeNet-5, and the neural network correctly recognizes this image as a three, and if you look to the left of this image, what's interesting about this is the display on the left portion of this is actually showing the intermediate computations of the neural network. In other words, it's showing you what are the hidden layers of the neural network computing.

And so, for example, if you look at this one, the third image down from the top, this seems to be computing, you know, certain edges into digits, right? We're just computing digits on the right-hand side of the bottom or something of the input display of the input image, okay? So let me just play this video, and you can see some of the inputs and outputs of the neural network, and those are very different fonts. There's this robustness to noise. All right. Multiple digits, that's, kind of, cool. All right.

So, just for fun, let me show you one more video, which was – let's see. This is another video from the various CV's, the machine that changed the world, which was produced by WGBH Television in corporation with British Foreclass Incorporation, and it was aired on PBS a few years ago, I think. I want to show you a video describing the NETtalk Neural Network, which was developed by Terry Sejnowski; he's a researcher. And so NETtalk was actually one of the major milestones in the history of neural network, and this specific application is getting the neural network to read text.

So, in other words, can you show a piece of English to a computer and have the computer read, sort of, verbally produce sounds that could respond to the reading of the text. And it turns out that in the history of AI and the history of machine learning, this video created a lot of excitement about neural networks and about machine learning. Part of the reason

was that Terry Sejnowski had the foresight to choose to use, in his video, a child-like voice talking about visiting your grandmother's house and so on.

You'll see it in a second, and so this really created the perception of – created the impression of the neural network being like a young child learning how to speak, and talking about going to your grandmothers, and so on. So this actually helped generate a lot of excitement within academia and outside academia on neural networks, sort of, early in the history of neural networks. I'm just gonna show you the video.

[Begin Video] You're going to hear first what the network sounds like at the very beginning of the training, and it won't sound like words, but it'll sound like attempts that will get better and better with time. [Computer's voice] The network takes the letters, say the phrase, "grandmother's house," and makes a random attempt at pronouncing it. [Computer's voice] Grandmother's house. The phonetic difference between the guess and the right pronunciation is sent back through the network. [Computer's voice] Grandmother's house. By adjusting the connection strengths after each attempt, the net slowly improves.

And, finally, after letting it train overnight, the next morning it sounds like this: Grandmother's house, I'd like to go to my grandmother's house. Well, because she gives us candy. Well, and we – NETtalk understands nothing about the language. It is simply associating letters with sounds. [End Video]

All right. So at the time this was done, I mean, this is an amazing piece of work. I should say today there are other text to speech systems that work better than what you just saw, and you'll also appreciate getting candy from your grandmother's house is a little bit less impressive than talking about the Dow Jones falling 15 points, and profit taking, whatever. So but I wanted to show that just because that was another cool, major landmark in the history of neural networks. Okay. So let's switch back to the chalkboard, and what I want to do next is tell you about Support Vector Machines, okay?

That, sort of, wraps up our discussion on neural networks. So I started off talking about neural networks by motivating it as a way to get us to output non-linear classifiers, right? I don't really approve of it. It turns out that you'd be able to come up with non-linear division boundaries using a neural network like what I drew on the chalkboard earlier.

Support Vector Machines will be another learning algorithm that will give us a way to come up with non-linear classifiers. There's a very effective, off-the-shelf learning algorithm, but it turns out that in the discussion I'm gonna – in the progression and development I'm gonna pursue, I'm actually going to start off by describing yet another class of linear classifiers with linear division boundaries, and only be later, sort of, in probably the next lecture or the one after that, that we'll then take the support vector machine idea and, sort of, do some clever things to it to make it work very well to generate non-linear division boundaries as well, okay? But we'll actually start by talking about linear classifiers a little bit more.

And to do that, I want to convey two intuitions about classification. One is you think about logistic regression; we have this logistic function that was outputting the probability that Y equals one, and it crosses this line at zero. So when you run logistic regression, I want you to think of it as an algorithm that computes theta transpose X, and then it predicts one, right, if and only if, theta transpose X is greater than zero, right? IFF stands for if and only if. It means the same thing as a double implication, and it predicts zero, if and only if, theta transpose X is less than zero, okay?

So if it's the case that theta transpose X is much greater than zero, the double greater than sign means these are much greater than, all right. So if theta transpose X is much greater than zero, then, you know, think of that as a very confident prediction that Y is equal to one, right? If theta transpose X is much greater than zero, then we're gonna predict one then moreover we're very confident it's one, and the picture for that is if theta transpose X is way out here, then we're estimating that the probability of Y being equal to one on the sigmoid function, it will be very close to one. And, in the same way, if theta transpose X is much less than zero, then we're very confident that Y is equal to zero.

So wouldn't it be nice – so when we fit logistic regression of some of the classifiers is your training set, then so wouldn't it be nice if, right, for all I such that Y is equal to one. We have theta transpose XI is much greater than zero, and for all I such that Y is equal to zero, we have theta transpose XI is much less than zero, okay? So wouldn't it be nice if this is true? That, essentially, if our training set, we can find parameters theta so that our learning algorithm not only makes correct classifications on all the examples in a training set, but further it's, sort of, is very confident about all of those correct classifications. This is the first intuition that I want you to have, and we'll come back to this first intuition in a second when we talk about functional margins, okay? We'll define this later.

The second intuition that I want to convey, and it turns out for the rest of today's lecture I'm going to assume that a training set is linearly separable, okay? So by that I mean for the rest of today's lecture, I'm going to assume that there is indeed a straight line that can separate your training set, and we'll remove this assumption later, but just to develop the algorithm, let's take away the linearly separable training set. And so there's a sense that out of all the straight lines that separate the training set, you know, maybe that straight line isn't such a good one, and that one actually isn't such a great one either, but maybe that line in the middle is a much better linear separator than the others, right?

And one reason that when you and I look at it this one seems best is because this line is just further from the data, all right? That is separates the data with a greater distance between your positive and your negative examples and division boundary, okay? And this second intuition, we'll come back to this shortly, about this final line that I drew being, maybe, the best line this notion of distance from the training examples. This is the second intuition I want to convey, and we'll formalize it later when we talk about geometric margins of our classifiers, okay?

So in order to describe support vector machine, unfortunately, I'm gonna have to pull a notation change, and, sort of, unfortunately, it, sort of, was impossible to do logistic regression, and support vector machines, and all the other algorithms using one completely consistent notation, and so I'm actually gonna change notations slightly for linear classifiers, and that will actually make it much easier for us – that'll make it much easier later today and in next week's lectures to actually talk about support vector machine.

But the notation that I'm gonna use for the rest of today and for most of next week will be that my B equals Y, and instead of be zero, one, they'll be minus one and plus one, and a development of a support vector machine we will have H, have a hypothesis output values to the either plus one or minus one, and so we'll let G of Z be equal to one if Z is greater or equal to zero, and minus one otherwise, right? So just rather than zero and one, we change everything to plus one and minus one.

And, finally, whereas previously I wrote G subscript theta of X equals G of theta transpose X and we had the convention that X zero is equal to one, right? And so X is an RN plus one. I'm gonna drop this convention of letting X zero equals a one, and letting X be an RN plus one, and instead I'm going to parameterize my linear classifier as H subscript W, B of X equals G of W transpose X plus B, okay? And so B just now plays the role of theta zero, and W now plays the role of the rest of the parameters, theta one through theta N, okay? So just by separating out the interceptor B rather than lumping it together, it'll make it easier for us to develop support vector machines. So – yes.

Student:[Off mic].

Instructor (Andrew Ng):Oh, yes. Right, yes. So W is – right. So W is a vector in RN, and X is now a vector in RN rather than N plus one, and a lowercase b is a real number. Okay.

Now, let's formalize the notion of functional margin and germesh margin. Let me make a definition. I'm going to say that the functional margin of the hyper plane WB with respect to a specific training example, XIYI is – WRT stands for with respect to – the function margin of a hyper plane WB with respect to a certain training example, XIYI has been defined as Gamma Hat I equals YI times W transpose XI plus B, okay?

And so a set of parameters, W, B defines a classifier – it, sort of, defines a linear separating boundary, and so when I say hyper plane, I just mean the decision boundary that's defined by the parameters W, B. You know what, if you're confused by the hyper plane term, just ignore it. The hyper plane of a classifier with parameters W, B with respect to a training example is given by this formula, okay? And interpretation of this is that if YI is equal to one, then for each to have a large functional margin, you want W transpose XI plus B to be large, right? And if YI is equal minus one, then in order for the functional margin to be large – we, sort of, want the functional margins to large, but in order for the function margins to be large, if YI is equal to minus one, then the only way for this to be big is if W transpose XI plus B is much less than zero, okay?

So this captures the intuition that we had earlier about functional margins – the intuition we had earlier that if YI is equal to one, we want this to be big, and if YI is equal to minus one, we want this to be small, and this, sort of, practice of two cases into one statement that we'd like the functional margin to be large. And notice this is also that so long as YI times W transpose XY plus B, so long as this is greater than zero, that means we classified it correctly, okay?

And one more definition, I'm going to say that the functional margin of a hyper plane with respect to an entire training set is going to define gamma hat to be equal to min over all your training examples of gamma hat, I, right? So if you have a training set, if you have just more than one training example, I'm going to define the functional margin with respect to the entire training set as the worst case of all of your functional margins of the entire training set. And so for now we should think of the first function like an intuition of saying that we would like the function margin to be large, and for our purposes, for now, let's just say we would like the worst-case functional margin to be large, okay? And we'll change this a little bit later as well.

Now, it turns out that there's one little problem with this intuition that will, sort of, edge us later, which it actually turns out to be very easy to make the functional margin large, all right? So, for example, so as I have a classifiable parameters W and B. If I take W and multiply it by two and take B and multiply it by two, then if you refer to the definition of the functional margin, I guess that was what? Gamma I, gamma hat I equals YI times W times transpose B. If I double W and B, then I can easily double my functional margin.

So this goal of making the functional margin large, in and of itself, isn't so useful because it's easy to make the functional margin arbitrarily large just by scaling other parameters. And so maybe one thing we need to do later is add a normalization condition. For example, maybe we want to add a normalization condition that de-norm, the alter-norm of the parameter W is equal to one, and we'll come back to this in a second. All right. And then so –

Okay. Now, let's talk about – see how much time we have, 15 minutes. Well, see, I'm trying to decide how much to try to do in the last 15 minutes. Okay. So let's talk about the geometric margin, and so the geometric margin of a training example – [inaudible], right? So the division boundary of my classifier is going to be given by the plane W transpose X plus B is equal to zero, okay? Right, and these are the X1, X2 axis, say, and we're going to draw relatively few training examples here. Let's say I'm drawing deliberately few training examples so that I can add things to this, okay?

And so assuming we classified an example correctly, I'm going to define the geometric margin as just a geometric distance between a point between the training example – yeah, between the training example XI, YI and the distance given by this separating line, given by this separating hyper plane, okay? That's what I'm going to define the geometric margin to be.

And so I'm gonna do some algebra fairly quickly. In case it doesn't make sense, and read through the lecture notes more carefully for details. Sort of, by standard geometry, the normal, or in other words, the vector that's 90 degrees to the separating hyper plane is going to be given by W divided by the norm of W; that's just how planes and high dimensions work. If this stuff – some of this you have to use, take a look t the lecture notes on the website.

And so let's say this distance is gamma I, okay? And so I'm going to use the convention that I'll put a hat on top where I'm referring to functional margins, and no hat on top for geometric margins. So let's say geometric margin, as this example, is gamma I. That means that this point here, right, is going to be XI minus gamma I times W over normal W, okay? Because W over normal W is the unit vector, is the length one vector that is normal to the separating hyper plane, and so when we subtract gamma I times the unit vector from this point, XI, or at this point here is XI. So XI minus, you know, this little vector here is going to be this point that I've drawn as a heavy circle, okay? So this heavy point here is XI minus this vector, and this vector is gamma I time W over norm of W, okay?

And so because this heavy point is on the separating hyper plane, right, this point must satisfy W transpose times that point equals zero, right? Because all points X on the separating hyper plane satisfy the equation W transpose X plus B equals zero, and so this point is on the separating hyper plane, therefore, it must satisfy W transpose this point – oh, excuse me. Plus B is equal to zero, okay? Raise your hand if this makes sense so far? Oh, okay. Cool, most of you, but, again, I'm, sort of, being slightly fast in this geometry. So if you're not quite sure why this is a normal vector, or how I subtracted this, or whatever, take a look at the details in the lecture notes.

And so what I'm going to do is I'll just take this equation, and I'll solve for gamma, right? So this equation I just wrote down, solve this equation for gamma or gamma I, and you find that – you saw that previous equation from gamma I – well, why don't I just do it? You have W transpose XI plus B equals gamma I times W transpose W over norm of W; that's just equal to gamma times the norm of W because W transpose W is the norm of W squared, and, therefore, gamma is just – well, transpose X equals, okay? And, in other words, this little calculation just showed us that if you have a training example XI, then the distance between XI and the separating hyper plane defined by the parameters W and B can be computed by this formula, okay?

So the last thing I want to do is actually take into account the sign of the – the correct classification of the training example. So I've been assuming that we've been classifying an example correctly. So, more generally, to find the geometric margin of a training example to be gamma I equals YI times that thing on top, okay? And so this is very similar to the functional margin, except for the normalization by the norm of W, and so as before, you know, this says that so long as – we would like the geometric margin to be large, and all that means is that so long as we're classifying the example correctly, we would ideally hope of the example to be as far as possible from the separating hyper

plane, so long as it's on the right side of the separating hyper plane, and that's what YI multiplied into this does.

And so a couple of easy facts, one is if the norm of W is equal to one, then the functional margin is equal to the geometric margin, and you see that quite easily, and, more generally, the geometric margin is just equal to the functional margin divided by the norm of W, okay? Let's see, okay. And so one final definition is so far I've defined the geometric margin with respect to a single training example, and so as before, I'll define the geometric margin with respect to an entire training set as gamma equals min over I of gamma I, all right?

And so the maximum margin classifier, which is a precursor to the support vector machine, is the learning algorithm that chooses the parameters W and B so as to maximize the geometric margin, and so I just write that down. The maximum margin classified poses the following optimization problem. It says choose gamma, W, and B so as to maximize the geometric margin, subject to that YI times – well, this is just one way to write it, subject to – actually, do I write it like that? Yeah, fine. There are several ways to write this, and one of the things we'll do next time is actually – I'm trying to figure out if I can do this in five minutes. I'm guessing this could be difficult.

Well, so this maximizing your classifier is the maximization problem over parameter gamma W and B, and for now, it turns out that the geometric margin doesn't change depending on the norm of W, right? Because in the definition of the geometric margin, notice that we're dividing by the norm of W anyway. So you can actually set the norm of W to be anything you want, and you can multiply W and B by any constant; it doesn't change the geometric margin. This will actually be important, and we'll come back to this later. Notice that you can take the parameters WB, and you can impose any normalization constant to it, or you can change W and B by any scaling factor and replace them by ten W and ten B whatever, and it does not change the geometric margin, okay?

And so in this first formulation, I'm just gonna impose a constraint and say that the norm of W was one, and so the function of the geometric margins will be the same, and then we'll say maximize the geometric margins subject to – you maximize gamma subject to that every training example must have geometric margin at least gamma, and this is a geometric margin because when the norm of W is equal to one, then the functional of the geometric margin are identical, okay?

So this is the maximum margin classifier, and it turns out that if you do this, it'll run, you know, maybe about as well as a – maybe slight – maybe comparable to logistic regression, but it turns out that as we develop this algorithm further, there will be a clever way to allow us to change this algorithm to let it work in infinite dimensional feature spaces and come up with very efficient non-linear classifiers. So there's a ways to go before we turn this into a support vector machine, but this is the first step. So are there questions about this? Yeah.

Student:[Off mic].

Instructor (Andrew Ng):For now, let's just say you're given a fixed training set, and you can't – yeah, for now, let's just say you're given a fixed training set, and the scaling of the training set is not something you get to play with, right? So everything I've said is for a fixed training set, so that you can't change the X's, and you can't change the Y's. Are there other questions?

Okay. So all right. Next week we will take this, and we'll talk about authorization algorithms, and work our way towards turning this into one of the most effective off-the-shelf learning algorithms, and just a final reminder again, this next discussion session will be on Matlab and Octaves. So show up for that if you want to see a tutorial. Okay. See you guys in the next class.

[End of Audio]

Duration: 74 minutes