Week 1
Video Transcripts

Video 1 (03:35): Introduction

Welcome to the first lecture of machine learning. I'm your professor, John Paisley. I'm a member of the Department of Electrical Engineering as well as the Data Science Institute at Columbia and my specialty is in machine learning and this is this class that we'll be working through in the following lectures is directly based on the machine learning course that I teach here at Columbia as part of the Data Science Institute.

So in this first lecture, I want to primarily focus on a general overview of the course and of what machine learning is and then go into a little bit of detail on a very specific problem working with Multivariate Gaussian in order to kind of highlight the different aspects and the different components of what we will be discussing in this course.

So, this course will cover model-based techniques for extracting information from data with some sort of an end task in mind. So these tasks could include, for example, predicting an unknown output given some corresponding input. We could simply want to uncover the information underlying our data set with the goal of better understanding what's contained in our data that we have. Or we could do things like data driven recommendation grouping, classification, ranking etcetera. So, using the data to help us learn how to perform these sorts of end tasks.

So in a course like this, there's a few ways the information can be presented, different orderings of the information. One example would be to partition it as in one half supervised
learning, the other half unsupervised learning, and I'll discuss that in a bit more detail in this lecture because that's the perspective that we're going to take but we could also think in terms of probabilistic models where we are working with probability distributions versus non- probabilistic models where we're learning from the data without any sort of probability probabilistic motivation.

There's also a dichotomy between modeling approaches. So, what is the model that we want to define for our data, versus optimization approaches which is very tightly linked with modeling approaches but with optimization now that we've defined a model, how do we learn the model? So, these are two separate problems with various techniques in these two sub-problems. So again, we're going to partition this course into roughly two halves.

The first half will focus on supervised learning and the second half will focus on unsupervised learning. And these additional ideas such as probabilistic versus non-probabilistic or modeling approaches versus optimization techniques will be sort of discussed as we go along, as needed. So those will be interwoven

throughout the course but the first part of this course will be strictly supervised learning and the second part will be unsupervised learning.

Video 2 (05:30): Supervised Machine Learning

What do we mean when we say we want to perform supervised learning?

In a nutshell, what we're saying is that we want to take inputs and predict corresponding outputs. So for example, if we do—want to do regression, we might have pairs of data. In this case, a one-dimensional value for x and a corresponding one-dimensional value for t. And then we would want to learn some sort of a function so that we input x and we make a prediction for the output t. So for example, here we have several data points as indicated by circles where the x axis is the input for that particular point and the t axis is the corresponding output. And now that we have this data set of these several points, we want to define some sort of a regression function. For example, this blue line that in some way interpolates the outputs as a function of the inputs. And then the goal is given this smoothing function that we've learned. For some new input x, we want to predict the corresponding output, t. So, we for future time points, we obtained x. We obtained an input and we want to predict the corresponding output.

So that's regression. We say its regression because the outputs are assumed to be real valued. Classification is another supervised learning problem that is slightly different. The form or the structure is very similar. We have pairs of inputs and outputs. And we get this data set which has many of these pairs of inputs and outputs and we want to learn some function so that in the future, when we get a new input for which we don't have the output, we can make a prediction of the output that's going to be accurate.

However, the key difference here is that, where with regression, the output is a real-valued output, with classification it's a discrete value thing. So, it's a category or a class. So, in this right plot, what I'm showing are input-output pairs, except now the input is a two-dimensional vector. So here, the input would be this two-dimensional point and the output for this plot is being encoded by a color. So, the output could be one of two values, either a blue value or an orange value.

So in this case, we want to take our data inputs and classify them into one of two outputs. So, we get a data set like this with all of these input locations and the corresponding color coded output. And now our goal is to learn some sort of a function, a classifier, so that we can partition the space such as is shown here, where for a brand new point, any of these points that we don't have the output, we can evaluate the function at that point and make a prediction of the output.

So we might say for this data set, we would partition this entire region here. These two regions into the orange class and this region here into the blue class. So any new points falling in this region will be

assigned to the blue class. So, the key here with supervised learning is that we're learning an algorithm based on a function mapping from input to output. We—the outputs are basically going to be telling us how to map the inputs so that we have an accurate function.

So we have input-output pairs. So, to look at a classic example we could think of spam detection. Given some set of inputs like these two chunks of text, we would want to assign it a label, plus one or minus one, sometimes we would say plus one or zero but we would want to sign it one of two possible labels. One label would correspond to an email that is spam. And we would want to then, you know, automatically delete that email.

And the other class would be non-spam emails. Emails that we want to put into our inbox and actually read. So it's essentially a filtering problem. So for example, we might have a data set, a data point like this containing this text and we would want to now input this into some sort of a function and say is it spam or not? In this case, most likely it's not. Or a data point like this. This piece of text where we would input it into the same exact function with the same classifier and in this case, that same classifier would say this email is a spam.

So, we classify this email to spam and this email to non-spam using the same classifier learned from examples of labeled spam and labeled non-spam emails.

Video 3 (05:21): Unsupervised Machine Learning

So essentially, the first half of this course is going to be all about learning different ways that we can define these functions to map inputs to outputs, either regression models or classification models, depending on the problem, as well as algorithms or techniques, for then learning the parameters of these models based on data.

So, that will take up the first half of this course. There are many very useful techniques, very different techniques for performing these two tasks. They'll entail different ways of thinking about the problems, probabilistic versus non-probabilistic. The models that we define will require different ways, or different techniques for learning them, so we'll need different optimization techniques.

So, the first half of this course will be all about supervised learning. Then in the second half of the course, we'll transition to unsupervised learning. And with unsupervised learning, the goal is a bit more vague. Supervised learning is very nice because we know that we want to map an input to an output. And honestly, we don't necessarily even care how it's mapped.

We don't care, whether we learn anything by mapping it, or in many cases we don't, perhaps, in some cases we do. We simply want to say, here is my input, what should I map it to as an output? And we measure the performance based purely on how well it does that task.

With unsupervised learning, we don't have, in most cases, this sort of an input-output mapping. We want to perform more abstract or vague tasks, such as understanding what is the information in our dataset. For example, we don't have an infinite amount of time to read, you know, so many thousands or millions of documents. So, we want a fast algorithmic way for taking in information, taking in data and extracting the information for us. So, for example, with unsupervised learning, we might want to do something like topic modeling, where we have many texts, data, many documents provided to us, we don't have any labels for these documents.

All we have is the text for each document and then we want to extract what is the underlying — what are the underlying themes within these documents, so that's the idea of topic modeling.

We also might want to do recommendations. This would be where we have many users and many objects and users will give feedback or input on a subset of these objects, either through a review or through some sort of a rating, like a star rating.

For example, with Netflix, a user could rate a movie one to five stars, and we want to take all of this information and learn some sort of a latent space where we can embed users and movies such that users who are close to each other share similar tastes, movies that are close to users are somehow appropriate recommendations to be made to those users, movies that are close to each other are similar and their content, and things that are very far apart or very unlike each other.

So, we want to learn this information simply from the data, from the raw data, and some model assumption that we have to make. So, for example, one of the most well-known unsupervised learning tasks is the topic modeling problem. And so, what I'm showing here is an example of what a topic model will learn if you provide it with over a million documents from the New York Times, over roughly 20 year period.

So, what we have in these documents is simply a tag that says that this is a document, and these are the words in the document. And we have that repeated for all of the documents in our dataset, again, it can be over a million of these documents. We want to make some sort of a modeling assumptions such that we find the words that should somehow cluster together.

These words would then define topics underlying our dataset. So, for example, by simply inputting the raw data from the New York Times, making a modeling assumption that doesn't, in advance, tell us which words should go with which other words, we can then run an algorithm to extract information like this that says that this set of words belongs together, this set of words belongs together, and so on.

So, we can learn, for example, 10 or more of these what are called topics that tell us which words belong together. And then not shown here is also how each document uses that topic. So, for example, for a particular document, we might say it's composed of these two topics and no other topics. And so, this

is information that's extracted from the raw data. We don't a priori tell the algorithm what it should learn. We simply say there is this structure that we want to learn, here's the data, tell me the structure.

Video 4 (4:58): Data Modeling

Everything that we are going to discuss in this course can, in some way, boil down to what's the flowchart that's contained on this slide. So, we have four components here, we have the data, which I'm calling block one. We have a model for the data, which I'm calling block two. We then have some sort of an inference algorithm, block three. And we have a task or some goal that we want to do, like predicting or exploring the data, which I'll call block four. And they all fall into all the algorithms or techniques, the motivations, have this same sort of flow.

So, we have a dataset, we start with that. We have a goal, what we want to do with this dataset, we have this here. We then have to define some sort of a model that's going to get us to our goal, this part here. We have to define a model for the data that's going to do for us what we want it to do, like make predictions or find the structure in the data. But this model is going to have unknown parameters to it and we need to learn these parameters, and so that's where block three, the inference portion comes into play. So, we basically say here's a model with some unknown parameters, here's a dataset that we're going to say that we want to model with this–with this model, now tell me what are the parameters I should set the model to in order to model it well, that's block three. And when we learn those parameters, we can then take new data and make predictions or explore the dataset.

So, if you remember from an earlier slide, we discussed the two different perspectives–or different pairs of perspectives that we could take. And they all kind of fall into these different ways that we can view these blocks. So, for example, the difference between supervised versus unsupervised learning can be thought of as comparing as block one and block four, thinking of block one and block four. So, what is the data and what do we want to do? If we have data where we want to predict an input–predict the output for an input by learning some function that map's inputs to outputs, then we're doing supervised learning, so that's what block one and block four will tell us to do. We want to simply learn the structure underlying our dataset with some sort of a model.

For example, we have documents and we want to learn the topics of those documents to explore the dataset, we might want to do an unsupervised learning model. And so, unsupervised versus supervised can be thought of as what block one and block four is telling us. And then block two and block three are the set of techniques that we perform toward that end.

If we think probabilistic versus non-probabilistic models, we're now not really thinking about the data or what we want to do with the data, we're thinking about the model, itself, primarily. So, we're defining a model that uses probability distributions in the probabilistic case, or we're defining a model where

probability distributions really don't come into play at all in the non-probabilistic case. And also, to a certain extent, the probabilistic versus non-probabilistic dichotomy appears in block three as well, in that some algorithmic techniques are purely motivated by the fact that we're doing a probabilistic model. However, a lot of techniques for non-probabilistic models apply equally as well to probabilistic models with no modifications at all. So, to some extent, block three is shared across the two and the difference between probabilistic versus non-probabilistic modeling is primarily in the types of models that we're going to define. Also, there was the difference between focusing on defining models versus focusing on defining the optimization techniques for learning models or for maximizing or minimizing objective functions. And so, this one is very clear-cut. If we wanted to focus on model development, we would be focusing on block two.

If we wanted to focus on the different optimization techniques, regardless of what model that we're thinking of, we would focus on block three. So, these are just ways of thinking about the problems. Again, every algorithm, every model, every lecture that we work through in this course is going to, in some way, be able to be boiled down to the information on this figure.

Video 5 (5:54): Gaussian Distribution (Multivariate)

In the remainder of this lecture, I want to work through this flowchart that we've just been discussing for one specific simple problem. So, here, we're going to be thinking of a Multivariate Gaussian, a Gaussian in d dimensions. And so, with–speaking in terms of the four blocks from the previous slide, I won't–this language won't be used in the remainder of the course but for this specific lecture, I'll focus on what are the different blocks in this flowchart.

With block one being the data, we assume that we have data points 'x_1' through 'x_n'. We have 'n' observations and each observation is a d-dimensional vector that can take any value. So, this is the data that we're dealing with, there's no other data that we have that we would want to model. We have a data set of 'n' d-dimensional vectors and we want to model those vectors.

We believe those vectors can take any value. Block two then, is the model that we want to define. So, we have this data, now what do we–what do we want to do with it? Well, how do we want to model it, I should say. So, for this problem, what we're going to think of– and I'll return to these in more detail in the rest of the lecture, we're going to say that we're going to model this data with a Gaussian distribution, a single–a Multivariate Gaussian probability distribution.

And we're going to say that the end datapoints are *iid*, independent and identically distributed, according to this distribution. So, we'll return to that in a few minutes– moments, but that's block two. We've now defined a model for the data. We've said this is the structure that we want to learn from the data because this is a probability distribution, so we're looking at this problem probabilistically.

We now get to add the intuition that we're saying, this is how the data was generated. We have a probability model that's saying the data was generated in this way. Okay, so we have the data, we've defined the model for the data, now we have to transition to block three and learn the parameters of the model. So, a Gaussian has parameters, we'll discuss it, it's on the slide.

We have the data that we say came from this Gaussian, how do we now fix or tune the parameters to this Gaussian in a way to explain the data? And so, this is going to lead to a problem of having to define some sort of an objective function and then defining a way for optimizing that objective function.

And so, what we'll discuss in the rest of this lecture is something called maximum likelihood.

In block four, we can leave undefined for now. We don't need to say what we want to do with this data. We don't need to say why it is that we want to learn a model for the data. It could simply be that, we just want to reduce the data so that we can have a compact representation of it, and then throw the data away, or for many other reasons, several of which we'll discuss in future lectures.

Okay, so looking at block two, we're going to define a Multivariate Gaussian distribution. So, to refresh our memories, when we define a probability distribution, we're saying that the probability of 'x', or the density of 'x', given parameters mu and Sigma, is equal to some function. That function has to be nonnegative everywhere, so it's a function of 'x', as well as mu and Sigma. It has to be nonnegative everywhere and it has to integrate to one. And the Multivariate Gaussian has this specific form, where we take the input vector, so this is a d-dimensional vector. We subtract off its mean. We perform this quadratic function using a mean vector mu that's d-dimensional and a d by d matrix Sigma, that's positive definite.

We evaluate this function and that gives us a density for a particular point 'x' using a Gaussian with mean mu and covariance Sigma. So, we get something that looks like this. If d is two, so we have a two-dimensional vector 'x', we are evaluating this function 'p' in a two-dimensional space. This third dimension is then the value of the function at a particular location.

So, we would assume, looking at this function, that the mean is somewhere say around zero, zero. And the covariance defines this sort of a shape, so a point in this region will have very tiny density, very small density. So, we won't expect to see much data in this region, whereas in this hump region, we have large density, meaning most of the data is going to come from this region.

And we remember, with a Multivariate Gaussian that the mean parameter defines the expectation. So, the expectation of 'x' is equal to mu. And the covariance is equal to the spread of the distribution in the various dimensions, as well as the correlation between those dimensions. And so, that's equal to this function where we take the mean of 'x', subtract it off a point, and then evaluate the expectation of this outer product that's equal to Sigma.

Video 6 (4:01): A Probabilistic Model

Okay, so we use this distribution as part of block two, defining the model. Again, in this case, we're defining a probabilistic model, so where we're thinking probabilistically here. So, what a probabilistic model is simply a set of probability distributions on our data. So, we have data x, we have to define some probability distribution on 'x'. That distribution is going to take parameters theta, which we don't know. So, the probabilistic model is simply this model, this probability definition. We then want to pick the actual parameter theta. So, by this definition, we haven't actually defined the final distribution.

We've defined a distribution family meaning this probability distribution can take many different values for theta and no matter what value we set theta to, we're always working within the same distribution family. So, in the case of the Gaussian distribution that we've been thinking of, this density, 'p' of 'x' given theta, is a Multivariate Gaussian. That's the distribution family, a Multivariate Gaussian, and it takes parameters mu and Sigma, mean, and covariance. And for all values of mu and Sigma, we're always working with the same density, the same Multivariate Gaussian, the same distribution family. So, we've defined this model with—we've defined the distribution family. We've also made an assumption, an *iid* assumption. What *iid* stands for is independent and identically distributed, and so this is a very common assumption that makes working with data much easier. It's very often made, unless we know that it shouldn't be made, we usually make it otherwise. What this means is that intuitively, it means that every single datapoint is simply independent from every other, and it has the same distribution family. So, if we have 'n' Multivariate Gaussians, we assume that every single one of those 'n' vectors in our dataset was generated from a Multivariate Gaussian using exactly the same mean and covariance parameter.

And they were generated without reference to what any of the other values were. It was essentially a brand new experiment for each of the 'n' vectors that we obtain. You can also think of this in coin-flipping terms, where if we flip a coin where the coin has a 0.5 probability of heads, 0.5 of tails, the density or the distribution, in this case, not density that we're working with, is a binomial distribution with parameter 0.5, so that's the distribution. And the independent and identically distributed comes into play by saying that every coin that we flip has the same distribution of heads versus tails, and they're all flipped completely separate of each other. They don't influence each other in any way. So, we make this assumption and what that allows us to do is write our joint density this way. So, we have a probability of 'n' datapoints 'n' d-dimensional vectors coming from a distribution with parameter theta. And we can say that that joint density, the joint probability of all 'n' of these observations is simply the product of the probability of each of them individually. So, again, if these weren't independent, if these 'x' values were not independent of each other, we could not write it as this product form.

Video 7 (14:45): Maximum Likelihood Estimation

Okay, so we've discussed the data. We have 'n' d-dimensional vectors, that's the data. We've discussed block two, which is the model that we're going to define, a Multivariate Gaussian with an *iid* assumption. We now move to block three, which says, how do we learn the parameters of this model? Again, we've said that the data is *iid* Gaussian with a mean—and that Gaussian has a mean in a covariance, but we don't know what the mean and the covariance are yet. And, we also don't know how do we pick a good value of the mean and covariance. What is a good value? What's a bad value? We haven't even defined what's called an objective function to optimize. So, this all pertains to block three, the inference or the learning portion of the flowchart.
So, with this block, we have to define an objective function. We have to say, here is a mathematical function and it always boils down to defining some mathematical function, which is a function of our data and a function of unknown model parameters that are defined by block two for us. And this math function tells us which values of those parameters are good, which are bad. If we have two values of the parameters and we evaluate that function at those two values, then the value that is larger or smaller, depending on how we define our objective function, but that evaluation will tell us which of the two values is better than the other value. So, we need to define an objective function. Then once we've defined that objective function, we need to come up with some sort of an algorithm for then optimizing it, finding the parameters that maximize or minimize the function, depending on which of— which of those two we want to do. So, a common approach with probabilistic models is something called maximum likelihood estimation and so we'll look at that for the remainder of this lecture. So, with the maximum likelihood approach, we simply want to find the value of theta that maximizes the joint likelihood function. So, we've defined the model, this is a probability distribution on our data. For a particular value of theta, it tells us what is the likelihood of the data coming from that model. What is the probability of the observations that we have, given that particular value of theta. And maximum likelihood simply says, well, if we've defined a model and each evaluation of this function for a particular theta says how likely the data is given that value, we simply want to find the value of theta that maximizes the likelihood. We want to find a value for theta that makes it the most— that says that this dataset is the most probable from a Multivariate Gaussian having this parameterization.

So, in a sense, we're fitting the data that we have the best. And so, that defines the maximum likelihood problem. We want to find the arg max over theta, so we don't necessarily care what the value is. So, we aren't doing max theta, we're doing the arg max, meaning we care what the value of theta is more than what this function evaluates to at that theta. So, we take the arg max of theta of the joint likelihood and we call that the maximum likelihood estimate. Now for this maximum likelihood estimate, we need a way to find the maximum value, so we can—we can simply take arbitrary values of theta and plug them in and continue to do that, until we're satisfied that we've picked the best one out of the options we've given ourselves. Or we can do a more principled approach, where we actually define some sort of an algorithm that's going to find for us, very efficiently, the optimal solution. And so, that leads to the maximum likelihood equation. So, we have this joint likelihood function. We've made an *iid* assumption, so we can write this thing as a product of the likelihoods of each individual observation. This now gives us some function that we want to maximize and so to do that, we follow the maximum likelihood criterion, the optimality criterion, that says that the maximum of a function is the point at which the

gradient is equal to zero. So, if we want to think about this in terms of an image, we have something that looks like this. This is a very rough thing. Now we evaluate this function at all values for theta. We want to find the maximum that's going to be the point here at which the gradient is equal to zero.

So, we can't move in any direction and improve our function. We can't modify theta in any way to improve the—the value of the function, so we find a peak. So, this is what we want to optimize. We want to find the point of theta such that the gradient of our objective function, in this case, the joint likelihood is equal to zero. So, we have more that we can do with block three here. We've defined the objective function. This is a quote unquote block three problem. This is defining the thing that we want to optimize, the joint likelihood. We've said how we can do that. We can find the point for theta, such that the gradient of the objective function is equal to zero. But we're not yet done. We have to give some sort of an actual—an algorithm, something we can run on a computer that will do this for us. So, this will require a technique that is problem specific. For this particular problem, we'll discuss one algorithm. For other lectures, we'll define models for which this particular algorithm has no use whatsoever, in which case we'll define another algorithm for that particular problem. So, here, we're working with probability distributions in an *iid* setting. And more often than not, this leads to something called the logarithm trick that makes optimization very, very easy. So, in this case if, we want to take the gradient of the product of likelihoods, this product is—again, because of *iid* assumption, if we want to take the gradient of this, actually calculate derivatives with respect to theta and set to zero, we very often will find that this is a very complicated thing to do. It's not a nice thing to work with and that's because we're taking a derivative of a product of many things. So, here, we're going to use the fact that the logarithm is a monotonically increasing function in R plus. What this means, is that, if we evaluate the logarithm of a positive number, then if the—if that positive number is increasing, then the logarithm of those numbers are also increasing.

So, simply put, the logarithm doesn't change locations of maximum and minimum. In this case, we can take a joint likelihood that can be written in this way, as a function— as a product of terms. And when we take the logarithm of that product, we have the fact that, that turns into the sum of the logarithm of each individual term. So, the logarithm of a product of functions, 'fi', is equal to the sum of the logarithm of those functions. And so, this turns our problem into something very simple. And again, we care about the location, we care about the value for theta that maximizes this function. What this means is that even though the maximum of the log of a function 'g' over 'y' is not equal to the maximum over 'y' of the function 'g'. If we actually plug in the value 'y', that maximizes this left hand side, and then plug in the value 'y' that maximizes this right hand side, those values are not going to be equal—the maximum are not going to be equal to the same thing because the left side is the logarithm of the right side. The arg max is the same, so the location doesn't change. So, the value of 'y' that maximizes the log of a function 'g' is equal to the value of 'y' that maximizes the function 'g', if 'g' is a positive function. So, the location of the maximum doesn't change, even though the value of the maximum clearly does change because g is not equal to log 'g'. And since we only care about the location of the maximum, because that's going to be the parameter, we can use this fact that the log of our joint likelihood is equal—the maximum of the log of the joint likelihood, that location is equal to the location that maximizes the joint likelihood.

Okay, so we still have to follow through and calculate all these things. So, again, tying it all together, the—using maximum likelihood with the logarithm trick, we have that the value of theta that maximizes the likelihood, this is the arg max over theta of the product of these individual likelihoods over each datapoint, this is the *iid* assumption here, is equal to the value of theta that maximizes the log of that product of likelihoods. And the log of a product is equal to the sum of the logs. And so, the value of theta that maximizes the sum of the log of each individual likelihood of each point, given a parameter theta, is equal to the value of theta that maximizes the original joint likelihood. So, instead of working with this function, we work with this function. We solve for the maximum likelihood solution, meaning we now have to find the value of theta such that the gradient of this function at that value is equal to zero. And so, notice what we've done here. Instead of having to take the gradient of this product, which can be a nasty mathematical thing to do, we now have to take the gradient of the log of each individual likelihood over each datapoint and sum those all together, and that's going to be something that's very easy to do, usually. And now find that the point of theta at which this sum is equal to zero. Okay, so now, depending on the choice of the model, there are going to be different ways in which we can actually solve this. So, the nicest case is when we can do this analytically. What this means is that we take this—we take the log of the likelihoods, we take the gradient. We actually calculate this thing, try to set it to zero, and realize that we have a very simple mathematical equation that we can plug things into to solve for theta.

So, that's how it's going to work out in this particular case that we're talking about because it's a fairly simple model. However, for more complicated models, we can't do this. We can't take the derivative and set it equal to zero and then solve for the value of theta at which that equality holds. So, we have to do something with a numerical algorithm. We have to use an iterative algorithm such that, even though we can't find the value of theta that maximizes or, in some cases, minimizes our objective function, we can have iterations where we're continually modifying the value of theta. And each modification is getting us closer and closer to the solution and it'll finally converge to the solution. So, these are involved numerical algorithms. We'll discuss several of these in this class. Also, we might find ourselves in the situation where we can't optimize this function exactly, we can only approximately optimize it. And so, intuitively, we can think of it in this way; really, these approximate algorithms are all based on numerical algorithms from part two, so really, three is not introducing any new modeling techniques, per se. But it's understanding what exactly two, these numerical algorithms defined by two are going to converge to. So, if we have a simple objective like this with one maximum—one local maximum, the global maximum is the local maximum, they're all the same. We can know—we can say that we're converging to the optimal point. But, if we have an objective function that looks something like this, then depending on where we start, we might converge to a different local optimal solution. So, for example, if we have these two local optimal solutions, think of them as peaks of a hill. Then, if we start in one location, for example, in this location, our algorithm might eventually converge to this point here, which corresponds to the global optimal solution. However, if we start from a different point, like this point here, we can only converge to this local optimal solution, which is not the global optimal solution. So, this is going to be something that happens a lot, where our algorithms are only going to converge to a locally optimal solution. And the model space is going to be much more complicated than one-

dimensional or one-dimensional space, so we can't inspect it with our eyes to see whether it's locally or how good of a local optimal solution it is, or if in fact, it's the global optimal solution. All we can say is that we have approximately optimized our objective function.

Video 8 (12:36): Examples

So now, returning to our Multivariate Gaussian example by again using the flowchart and the block language that's going to be isolated to this particular lecture. We have in block two that we've defined a Multivariate Gaussian model for our data. We've said that we have data points 'x_1' through 'x_n'. Each of these data points is in 'R^d'. We are going to assume that they're i.i.d from a Multivariate Gaussian, meaning that each of these data points is an independent experiment, an independent trial where we get a random vector from a Multivariate Gaussian with a mean mu, which is a vector in 'R^d', and covariance Sigma, which is a D by d matrix that's a positive definite. We've defined this model, but we don't know the meaning of the covariance. So really, we've defined a family of distributions. We've defined the family of Multivariate Gaussian distributions for our data with an *iid* assumption. So, that's our block two problem. With block three now, we've defined that we want to find a value for mu and Sigma that maximizes the joint likelihood of our data. So, we've defined the joint likelihood of our data as the function that we want to maximize. In this case, it's maximizing the function because we want it to be more probable, our dataset to be more probable. In other cases, we'll want to minimize the function when usually when we have no probabilistic interpretation. So, we've said that now we want to find a value for mu and Sigma, and the value that we want to find is the one that maximizes the log of the joint likelihood, which has this form, the sum over the log of the individual likelihoods. And, the value that maximizes this function is the value of mu and Sigma such that the gradient at that point is equal to zero.

So to finally solve block three, to find the value of these parameters for our model, we have to now take derivatives. So, here's how we're going to do that. So again, we want the final gradient to be equal to zero. So, we'll put a zero here. First, let's take a derivative with respect to mu, and so the way that we're going to now take these derivatives is simply unique to this problem. Other problems we might do the derivatives in a different way or in a different ordering. Right now, I'm simply focusing on the derivative with respect to mu and ignoring the derivative with respect to Sigma, and it's going to work out in that case. So, we have the log of the joint likelihood, that is, the sum over each of the 'n' observations of the log of the Multivariate Gaussian, evaluated at each observation. So, here we're starting to get into the actual math where we're plugging things in and solving math equations. So, we write out the joint–the multivariate Gaussian density function. We evaluated at each of the 'n' data points for particular mu and Sigma, and we get this. We can now take this log and simplify it, in this way. So, we've simply written the log of this product of two terms, as the sum of the log of each of those two terms, and we get this function. And now, we take the derivative of this function with respect to mu, and we find that we get this term here. So, this term here is the gradient of the log of the joint likelihood with respect to the

mean vector mu, and we want to set this to zero. So, notice that we don't actually set it to the scale of zero, because this is a d-dimensional vector. We're actually going to set it to a vector of d zeros, in this case. So, that's one thing that we need to note.

Okay! So now, let's see if we can solve this and so we simply do it by inspection, and we notice that fortunately Sigma doesn't matter. For any value of Sigma, we can find the optimal value of mu by setting it equal to the empirical average of all of our data points. So, if we set mu to be equal to the sum over all of the data points divided by 'n', the empirical average, call that the maximum likelihood value for mu, then that is the value at which this function here is equal to zero any value of Sigma. So fortunately, it turns out in this case that Sigma does not change our maximum likelihood solution for mu, which is why we could simply solve mu first. We now need to find the maximum likelihood solution for the parameter Sigma of our Multivariate Gaussian. So, we perform the same exact steps as on the previous slide with the exception that we're now taking the gradient of the log of the joint likelihood with respect to the matrix Sigma, and setting that gradient equal to a matrix of zeros. So again, because Sigma is a d by d matrix, when we take the gradient of this function with respect to that matrix, what we get is not Scaler's zero. Often we just write zero, even if we're not talking about a Scalar value. We get a matrix of zeros, that's a d by d matrix of zeros. So, we have our log of our joint likelihood. We take the gradient of that with respect to Sigma and we get this function here. This is a d by d matrix. Notice that it's a function of mu and it's a function of Sigma, and we want to find the value of Sigma and mu such that this is equal to a matrix of zeros. So, we already know mu. We have, we know that mu is optimized at the point mu ML here, from the previous slide. So, we know that we're going to evaluate this thing at the point mu ML from the previous slide. And so, solving then for Sigma by setting this term equal to the matrix of zeros we find that Sigma, the maximum likelihood value for the matrix Sigma is simply equal to the empirical average of the outer products of our data where we've subtracted off the maximum likelihood mean. So, that's the solution for performing a maximum likelihood of the Multivariate Gaussian distribution using an *iid* assumption.

So to again summarize, if we have 'n' observations, so this is the data block one, each of those is an 'R^d'. And, we then want to transition to block two. We want to define a model for it. So, we hypothesize that the data was generated *iid* from a Multivariate Gaussian. So, we've defined the model. We now need to figure out how to set the parameters of that Multivariate Gaussian so that then motivates block three, which is the inference portion where we define the objective that we want to optimize to be the likelihood function. So, we want to bind the value of the parameters of a Multivariate Gaussian that maximizes the likelihood, and then derive an algorithm for doing that. And, when we followed through on that process we found that the maximum likelihood solution for Gaussian was to set the mean equal to the empirical average of the data, and to set the covariance equal to the empirical covariance of the data where we subtract off the maximum likelihood mean.

So, we've solved the problem and now the question is, are we done. So, obviously we're not done. There are many assumptions and issues with this approach that makes finding the quote, unquote best parameters not a complete victory. So again, we made a model assumption. We assumed that the data

was *iid*, independent, and identically distributed as a Multivariate Gaussian. This was an assumption that we made that might not be true. For example, we might think that there's sequential information in these vectors that the vector, the i^{th} vector in this sequence depends somehow on the previous vector in that sequence. This model does not make that assumption because that would violate the *iid* assumption. So, we've made a modeling assumption, block two. We made an *iid* assumption. Then, in block three we defined an objective that said what the best value of the parameters are. This also was an assumption, what it means to say that one parameter setting is better than another depends on what function that we define that we want to optimize, which is a choice that we also make. So, the choice of objective function was also arbitrary on our part.

Once we made that choice, then the actual algorithm that we followed by taking derivatives was optimal. We solved the problem. But that problem was solved for a function that we chose, that we made an assumption was a good function to model the data. So, in many cases perhaps we don't want to maximize the likelihood. So for example, we haven't been talking about block four. Here's where block four would come into play. We often want to use the maximum likelihood solution. So for example, we would want to use the mean and covariance of our Multivariate Gaussian to make predictions about a new value that we haven't observed yet, to say here's where I think, here's the region of the space that I think the next value is going to come in. And also, here's the region that I don't think the next value will fall in.

So, how does the maximum likelihood generalize to this new data point. And so, we'll discuss a little bit in future lectures about this. However, at a high level if the data that we're modeling 'x_1' through 'x_n', if that data doesn't somehow represent our space or capture our space well, if it doesn't cover everything that we can expect to see, if there are huge gaps that are missing in our data, then the maximum likelihood solution can overfit. It can—it'll only learn based on what you give it and it's not going to have any sort of way for accounting for the fact that there might be something new or unseen in the future. It simply hones in on predicting as well as possible the data that you've observed. And, if you don't have a lot of data or if your data doesn't represent what you plan to—what you think is out there very well, for example, if you only sample from one region within the United States but you don't sample anything from another region, then you're going to model the region that you sample from very well, but it might not generalize to the new region well. So, maximum likelihood in this case can fail. And so, there are many decisions that we have to make. We'll discuss several of these throughout this course that makes machine learning in a sense very much an art, as well as a science. And, we've gone through one specific example, a simple example here. But, in future lectures we'll discuss many more.

Video 9 (9:93): Linear Regression

In this lecture, we will be discussing something called, linear regression. So, this is the most basic form of regression, and to introduce it all we use a very simple problem, the problem of Old Faithful. So, with

this problem what we have is a geyser that's erupting for a certain amount of time and then not erupting for a certain amount of time.

So say, it erupts for a few minutes, and then it's silent for a few minutes. And, what we want to do is we want to come up with a way to predict when is the next eruption going to be. And so, one way that we might do this is to collect some data. So, collect pairs of inputs and outputs, where the input might be how long the geyser erupted for, in minutes, and then the output would be after that eruption how long we had to wait for the next eruption, in minutes. And so, what linear regression is going to do is, it's going to take the input, in this case, how long the geyser erupted for, and try to predict the output, how long it's going to now be silent until the next eruption.

So let's, look at the data, at what we have and how we might want to model this. So, here we clearly see a trend. In the x axis, we have how long the geyser erupted for, and then in the y axis, we have how long we have to wait for the next eruption. So for example, this point here, tells us that the geyser erupted for roughly a little under four minutes. And then, after that we had to wait for a little over an hour for the next eruption. So now, the question is, can we meaningfully predict the time between eruptions using only the duration of the last eruption.

So by looking at this data, we might want to come up with some sort of a function, a simplifying function, where you simply have to take the input, which is how long the geyser erupted for, perform a function on that and then the output will be a prediction of how long we think we're going to have to wait for the next eruption. And by looking at this, the most natural choice is to model this as a linear function. So, this is the most simple regression setting, where we want to do linear regression, where we have an input that we believe is related according to some linear function to the output. So, let's look at one model for this. The model that we're going to consider is called linear regression, where we have the output, which is the waiting time, is a function, where we have some constant, w0 times the last duration of the eruption, how long it erupted for, times a weight, w_1. And so, w_0 and w_1 are two values that we're going to now learn from the data. So, this is an example of linear regression, in this case, because we are simply learning a line to fit the data.

So, how would this translate to higher dimensions just to get some intuitions? For example, imagine that we have two inputs and we want to predict an output based on those two inputs. In this case, we would use the function, also a linear function, where we say that the output is approximately equal to some offset or bias plus, the first input times a weight on the first input, plus the second input times a weight on the second input. And so, here in this plot you can see this, where we have data that we have the inputs that are two-dimensional, and the output again is one-dimensional. And, instead of a line, in this case we're learning a plane through the data. But again, it's a very simple linear function of the weights. We want to use data. We want to use pairs of inputs and outputs to learn the function, meaning to learn the values w_0, w_1, and in this case, also w_2. So let's, look at a more basic definition of the regression problem. So, what we have as data are inputs. We'll call those 'x' that are d-dimensional and 'R^d'. These 'x' —these inputs have many names. We can call them measurements, covariates, features,

independent variables, and I'll probably switch back and forth between these different names throughout the lectures. And, the outputs are the corresponding response or dependent variable, 'y', which is a real value. So, this is the data that we're dealing with. We have inputs and those can be in 'R^d', and outputs that are real-valued numbers that we want to predict. And, the goal of the regression problem is to define a function, 'f' that maps an input to an output so the function, 'f' takes 'x' as an input, that's an 'R^d', and maps that to a value in 'R', which is in the output space, such that the output can be reasonably assumed to be approximately equal to the mapping of the input through the function, along with some parameters or free variables of the model, 'w'. So, the function, 'f' is called a regression function, and we're going to call its free parameters, 'w'. So, those are the — the functional form is 'f', whereas 'w' are the parameter settings to that function that change how the input, 'x' maps to the output, 'y', and the goal is now to learn those parameters, given data.

So what makes this a linear regression model, is that the function, 'f' is a linear function of the unknown parameters, 'w'. So, we'll see later that this does not necessarily mean that it's a linear function of the covariates, 'x'. We can have nonlinear functions of 'x', but the weights, the values of the weights that we want to learn interact linearly with the inputs, 'x' to map to the outputs, 'y'. Okay! So, let's look at the simplest linear regression model, and a way that we can learn it through these squares. So, the model again takes an input, 'x_i', passes it to a function with parameters, 'w', and predicts an output, 'y_i' associated with 'x_i'. According to this function, where we have the weight, 'w_0', is just a bias that shifts the plane or shifts the line through the intercept. And then, we have a dot product or a element-wise product of the weight, 'w_j' with the j^{th} dimension of input, 'x_i'. So, 'x_i' is an 'R^d'. We take the j^{th} dimension of X_i, multiply it, by a weight 'w_j', do this for each dimension and sum those values up, and add the bias to it, and predict the output to be approximately equal to this linear function. So, we've defined the model, and we now collect some training data, meaning we collect pairs of instances of inputs and outputs that we know through measuring them or obtaining them in some way. So, we have 'n' pairs, 'x_1', 'y_1', where, 'y_1' is the corresponding response for input, 'x_1', through 'x' and 'y_n'. And now, the goal, and this is pervasive throughout machine learning, is to use this data to learn the vector, 'w', such that we can make predictions or we can approximate outputs according to that function. So, what does it mean to find the vector, 'w'—How can we find these values for 'w' that give us this prediction, according to this linear function—So, in order to know what a good value for the vector 'w' is, we need to define an objective function.

And, what this does is, it basically tells us what are good values for the vector, 'w', and what are bad values. So, for least squares the objective function is the most straightforward one you could think of. It's the sum of the squared errors. So, here we have the output, 'y_i' from our training data set, and we subtract our prediction of what 'y_i' is according to our linear regression model. That's the error of our prediction. We then square that value, so it's always a positive number, and then sum up those values to get the total error, the total sum of squared errors, of our model.

Video 10 (2:34): Linear Regression Example

So, we saw the example of Old Faithful for a one-dimensional problem, where we have an input that's a one-dimensional value, and we map that to the output. Let's look at an example of a two-dimensional problem, where we have for each pair, 'x', 'y', we have a two-dimensional vector, 'x'. So, that means that we have, for each object, two numbers that characterize that object with which we want to then predict the output. So, here's a very simple example of that where we have as an input, a two-dimensional vector that has some measure of education and seniority.

So for example, how many years of education do you have and how many years seniority do you have, that would be the input for an individual. And then, the output is that person's income and so that's a– we'll assume it's a real valued number. And so, the model now is to predict the income of a person, as the education of that person times some weight, plus the seniority times some other weight, plus some offset, 'w_0'. And so, that's our two-dimensional linear regression model. And so, let's ask a question. First, what do these weights tell us? What's–how can we interpret these weights? So first, both weights, we imagine that we have some data.

We learn this model and both weights are positive. What does this tell us? How can we interpret what these weights are telling us?

So, the answer is that as the education or seniority go up, income tends to go up, because for every increase–unit increase in education or in seniority, we're multiplying that by a positive number, which means that we have an increase in the predicted income. And so, this is just a way that we can then use our weights, not only to predict an output, but also to tell us how the inputs relate to that output. And so, as one caveat, we should say that this is not a statement about causation, so we're not saying that education or seniority caused the income to increase. We're saying–what we are saying is that these variables are correlated. So, these inputs are correlated with the output in this way.

Video 11 (17:00): Least Squares

So, let's look at what this least squares objective function is doing in pictures. So again, we have a two-dimensional input, and we're predicting something in R, which is this Y dimension here. And so, these red dots each are data point that we have.

So for example, this dot here gives us a two-dimensional location in the 'X_1', 'X_2' plane, and the Y value is the corresponding output. So, we can represent this in three dimensions. Then, we learn a hyper plane or we learn a linear function of this where we have, according to the previous slide, we have this sort of a function where d is now equal to two, and that looks something like this.

So, we take 'X_1', multiply it by a weight. We take 'X_2', multiply it by a weight. So, pretend that we have this point here. We take the 'X_1' coordinate, multiply it by its weight, the 'X_2' coordinate, multiply it by its

weight, and then add the offset, 'w_0' to that and we get our prediction, which is this point on the plane here.

And the error of that prediction, if we're looking at this point, is equal to the length of this point to- it's equal to the length of this point to the plane, squared. So, that's the squared error. So, we basically take each point. We drop a line parallel to the Y axis to the plane. That length is our error of our prediction. We then square that value, sum up those values, and that's our total error. And so, what least squares is trying to do is it's trying to find an orientation for this plane, so that it cuts through the data in a way where the sum of the squares of these lengths are minimized.

Okay. So, thus far we have our data pairs, 'x_i', 'y_i', of the measurements in 'R^d' and their corresponding responses that we're hoping to predict in 'R'. We've hypothesized that, there's a linear relationship between 'x_i' and 'y_i', meaning that we can multiply 'x_i', the j^{th} dimension of 'x_i', by some weight, 'w_j', add up those multiplications, include our offset, 'w_0', and then we have some error, and we have an approximation of 'y_i'. And, our goal now is to minimize the sum of the squared errors, which means that we're trying to minimize the sum of the squares of these epsilon, i's, what the number that we have to add in order to get this to become an equality. And of course, that's simply just taking the output and subtracting our regression function, which is, in this case, is equal to this.

So, that's our error. We square that value and then sum up those errors, and that's our objective function. So, this is what we're trying to minimize. This sum is what we're trying to minimize in 'w_0', 'w_1' through 'w_d'. So before we do this, let's ask whether we can use some nicer math notation in order to represent this in a way that's easier to work with. When we look at the notation, what we're going to do is just something very simple, and often this is done either in your code or if you're reading papers where these things are explained- what we're going to do is we're going to take the intercept and add that to the dimension of 'X'. So, we're going to extend 'X' from being an 'R^d' to being an 'R^d' plus one, where we take each point 'x_i' and we add a one to the first dimension of it. And now, we can interpret that first dimension as being multiplied against 'w_0', because in that case every single function evaluation for each of these 'x_i's' will have 'w_0' times one, which is the offset. Then, we're going to take each of these vectors, 'x_i' and put them along a row of a matrix, call that capital 'X'. So, that's what we have here. We have the first column-is all equal to one to account for the offset. And then, each feature vector is put along a row of the matrix from the second column through the 'd' plus first column. And now, we're going to put the weights in a vector that's also in 'd' plus one. So, we take each of these weights and we put them into a vector, and that's what we're going to call 'w'. And so notationally, we're always going to think of these columns-these vectors as column vectors, meaning if I write 'x_i', and this is a vector, then I'm assuming that for notation purposes-I'm assuming that this is a column vector. So for example, we could-for the i^{th} person we could fill in those values with their age, their heights, weights, their income, and those could be the inputs that we're using. And, the set of 'n' vectors are going to be stacked row-wise into a matrix. So, when I write capital 'X', I'm assuming that each row corresponds to one observation. So, the vectors, 'x_i' is transposed to become a row vector, and put along the i^{th} row of this matrix, 'X'. And also, an assumption that we're going to make is that features are

treated as continuous valued.

That's not necessarily going to be the case, but it's a simplifying assumption we're going to make. And, we're going to assume that we have more observations than we have dimensions, and we'll see why that is, in a minute. Okay so, let's look at the least squares objective function in vector form using our vector notation. So, this is the original objective function that we have. We have the output minus our approximation using linear regression. That's the error. We square that, sum up those squared errors, and that's the objective that we're trying to minimize in 'w_0' through 'w_j'. And so, using vector notation we've simply taken this term here and written it as a dot product between the input, 'x_i' and the coefficient vector, 'w'. This is simply equal to this, according to our definitions previously. And so now, we have our output-is approximated by the dot product between the inputs and the coefficient vector, 'w'. We square the error, sum it up, and we have that this is equal to this. So now, our goal is to minimize this objective function in the vector, w. And it's-We see immediately why it's nice to have this vector notation. What that means is we simply want to take the gradient of the objective function with respect to the vector, 'w' and find the point at which it's minimized, meaning the point at which the gradient is equal to zero. And if you think intuitively, what that means is if this is the function, 'L' and this is 'w', the gradient - zero is the minimum of that function, which is this point here. So, we're looking for that minimum- we're hoping that we can analytically solve this. We're hoping that we can find the global minimum. In this simple case, all of those things are possible. So, we take the gradient of the objective function. We find that the gradient of 'L' is just the sum of the gradients of each observation, and set that equal to zero. When we take the gradient, we remember that the gradient of this function here is equal to this. We solve for 'w' and call that least square solution, indicated by subscript, 'LS'. So, we take the gradient of the objective. We get this term here. We want to find the value of the vector, 'w' at which that is equal to zero. And, we can solve that to obtain this solution here. So, let's just quickly look at how we could write this, in matrix form. So, this is introducing some notations that will be used throughout the course, vector forms, matrix forms, you know, it's very convenient to use these shorthands. So, taking these vector representation and putting it into a matrix representation using the previous definitions, what we have is that we can stack the values of 'y_i' into a 'n'-dimensional vector, because we have 'n' observations.

So, 'y' now is an 'n'-dimensional column vector of all of the outputs. And, we're approximating that vector with the matrix, 'X' times the vector, 'w'. So, 'X', remember, is an n by d plus one matrix, and so, the i^{th} row of the vector, y-the ith dimension of 'y', is approximated by the i^{th} row of 'x' times the vector, 'w', which is simply what we have written here. Except now, we can write that as the error vector. So, we take the vector, 'y' and subtract off the vector, 'xw', which is our approximation of the vector, 'y', and then we take the dot product of that-those error vectors in order to represent the sum of squared errors. So, this vector here is the vector of errors of our approximation, and this dot product then gives us the sum of the square of those errors. So now, if we work with this function, and we take the gradient with respect to 'w' again, we find that we have this as being the gradient of the objective. We again set that equal to zero, and we find the least square solution, which now has this nice matrix vector form. So, just as a refresher to quickly give a review of what matrix, you know, matrix vector product versus a sum

of vectors times a Scalar, how do these things relate. So, remember that if we have a matrix, 'X' transpose here, so in this case, because each 'x_i' is along a row of the matrix, 'X', when we transpose that each 'xi' now as a column of 'x' transposed times the vector, 'y', which is the stacked outputs of each observation. We simply take the first dimension times the first column, and add to it the second dimension times the second column, all the way to the nth dimension times the nth column in this way. So, this is how we think of a matrix vector product as taking each column of the matrix, 'X', multiplying it or scaling it by the corresponding value in the vector, 'y', and then adding those up. And so, that's simply what these two notations are doing. And similarly, when we have matrices multiplied by each other, in this case 'x' transpose 'x', what we can think of that as is taking the first row or the first column of the left matrix times the first row of the right matrix which is this outer product, 'x_i', 'x_i' transpose, and then adding those all up. So we get 'x_1', 'x_1' transpose, all the way to 'x_n', 'x_n' transpose. We add those outer products up, and that's what this equality is simply saying. So basically, we have two notations for the key equation for least squares linear regression. We have that the vector, 'w' that minimizes the sum of squared errors, the least squared solution, can be written as either the inverse of the matrix form by taking all of the inputs and putting them along the rows as well as adding a column of ones to the first column, taking that product, inverting it, and then multiplying it by this matrix vector product. And so, the result is a vector of dimension d plus one. Or similarly, we could write it this way. It's saying exactly the same thing in different notation.

So now, that we have the least squares vector, this is the solution to our problem, this is the value of the vector, 'w' that minimizes the sum of squared errors. What is the first thing that we immediately think that we would want to do with this? Well, immediately we think we want to make predictions. So in real life, we have the data, the pairs of inputs and outputs, we learn our model, and now, we want to use that model to predict the output that we don't know for an input that is coming to us. So for example - in the Old Faithful example, we learn our model from the data that we've collected, we, then, measure the amount of time the geyser is erupting. When it finishes erupting, we have that amount of time. We put that into our linear regression model to get a prediction for how long we're going to have to wait for the next eruption, and then, we show that on a sign so that people know when they should come back. So basically, what this is saying is that given a new input, 'x_{new}', we take our least square solution and we predict the output that we don't get to observe as the dot product of the input with our least squares vector. And so, this is our approximation according to our linear regression model of what we think the output is going to be for the new input. I've worked out the math and everything worked out fine, but we've kind of skirted some potential issues, and one of those is that we actually can evaluate this equation in order to find the least square solution. And what that really, ultimately means is we're assuming that we can invert this matrix, 'X' transpose 'X', and that's not always going to be possible.

So, when do we know that it's not possible? So, we know that this matrix, 'X' transpose 'X' is not going to be invertible when it's not full rank. So, we need the matrix, 'X' transpose 'X' to be a full rank matrix. And so, we immediately ask, "Well, when is that matrix full rank?" And so, the answer is that when we look at the 'n' by d plus one matrix, 'x', it has at least d plus one linearly independent rows. And so, what this means is that any point in our d plus one can be reached by some weighted combination of d plus one

rows of 'x'. So this is one condition for that matrix, X transpose X being full rank. So, we'll return to this later, but obviously we can immediately say that if the number of data points is smaller than the-dimensionality of the problem plus one, then we can't do least squares, because it's not possible to find d plus one linearly independent rows of 'X', because there aren't that many rows of 'X' in this case. So, what we need–the kind of take away is that what we want for least squares to work is the number of data points to be much greater than the-dimensionality of the problem. So, we want the matrix, X to be tall and skinny. That's not a guarantee that it's going to work. We'll discuss more later about when it still might not work, but we're well on our way, if we have many more data points than dimensions of our problem.

Video 12(7:48): Broadening Linear Regression I

So, the nice thing about least squares linear regression is that it's actually very versatile. So, you might think that this is a bit of a simplistic model, if we have to assume that we're drawing a line through our data or hyperplane through our data, and that's all that we're allowed to model our data with, using linear regression.

But, recall that linear regression is–it's called linear regression because the output is linear in the weights. So, we have a linear function of the weights, not necessarily of the inputs 'x'. So, here's for example, a problem where linear regression is not going to be something that we want to do, as we've been discussing it so far. But, we'll see how we can modify it very, very easily in order to handle this problem, as easily as modeling it with a line. So, if we look at the original linear regression model that we've been formulating thus far, we, again, have the output 'y'—is approximately equal to the input 'x' times some weight 'w_1', plus an offset 'w_0', which is simply the intercept term. So, we're basically saying that the data has to be modelled by a line, in this case.

And of course, for this particular data set that we're looking at here, that's not an ideal function. We can see clearly that the function–the data does not follow a line, so we would want some sort of more complex model and more complex representation of how the inputs relate to the outputs.

So for example, and this is how the data was generated, we could say that the output is now equal to some offset, plus a weight times the input 'x', so again the input is just one-dimensional, plus, another weight times the input 'x' squared, plus a third weight times the input 'x' cubed—so, a third-order polynomial. So, notice that in this case, we're taking our data, our input 'x', and performing a function of it to take it from one dimension to now three dimensions. So, we have three kind of views of our input 'x', 'x'...'x' squared, and 'x' cubed, and we're multiplying each of those by their own separate independent weight, 'w_1', 'w_2', and 'w_3'. And according to this model, we can take our input, for example, the input 2, perform this function, so we have 2 here, 4 here, and 8 here. And then, using those four weights, we would get this output. And, if we sweep all of these values, we get this sort of a third-order function.

So, can we use linear regression, least squares linear regression, everything that we've been talking about thus far, to learn this problem?

And, so the answer is yes, it's very simple, and we end up with something called polynomial regression, which is simply taking your inputs 'x', performing some polynomial function, a particular function on that, and then doing least squares linear regression on the polynomial that you construct from the inputs. So, in this case, let's go back and let's say— let's ask why can we use everything that we've been doing thus far with minimal modifications. So, remember that the definition of linear regression is that the function 'f' is a linear function of the unknown parameters 'w'. It doesn't have to be a linear function of the inputs 'x', only of the parameters vector 'w'. So therefore, a function such as this second-order polynomial, is still a linear function in 'w', even though it's not a linear function in 'x'. And so, what we'll find is that we can actually do these squares and learn exactly the same solution, the only difference being that the matrix 'X' has a little bit more pre-processing to be — in order to construct it. So, let's look at a simple case of data 'x' coming in 'R'. So, we have one-dimensional inputs, and we're using that to predict a one-dimensional output. This is exactly like the previous slides we've been looking at. And, we want to learn a pth-order polynomial approximation of the relationship between the input and the output.

So very quickly, let's look at how can we do this. So let's construct the matrix 'X', where again we put in the first column—a column of 1s for the offset. And, the second column we then put the original data, 'x_1' through 'x_n'. In the third column, we then take those inputs and square them.

So, we have 'x_1' squared through 'x_n' squared. And, if we want to learn a p^{th}-order polynomial, we continue that up until we get 'x_1' to the 'p' through 'x_n' raised to the 'p'. And then, we solve this exactly as before, and we can approximate the output 'y'. For example, if we look at the output 'y' as being this vector through 'y_n', we can then say that 'y' is approximately equal to 'x' times a weight vector, where, for example, if we look at 'y_1', we're saying that's approximately equal to 1 times 'w_0', plus 'x_1' times 'w_1', plus 'x_1' squared times 'w_2', and so on. And so, we're basically approximating the output of 'y_1', as a p^{th}-order polynomial of the inputs 'x_1'. So then, if we set this approximation up, and we want to now learn 'w' using least squares, we follow exactly the same procedure as before. And, we learn that the least squares solution is 'x' transpose 'x' inverse times 'x' transpose times 'y'. Except now, even though the inputs are one-dimensional in this case, the matrix that we've constructed is 'p' plus one-dimensional.

And so, this vector, 'w_{LS}', is a 'p' plus one-dimensional vector, where the first dimension is the offset, the second dimension of 'w' corresponds to the weight on the input 'x', and the third dimension of 'w' corresponds to the weight on the input 'x' squared, and so on and so on. And so, that's a simple way that we can model a polynomial function of the inputs using least squares linear regression, where all we have to do is perform this pre-processing to construct our matrix 'X'.

Video 13(11:25): Broadening Linear Regression II

Here we have a one-dimensional input 'x' and a one-dimensional output, call it 't'. We have 10 data points, so 10 pairs of inputs and outputs indicated by these blue circles. And then, we want to learn a zeroth-order polynomial. What does this correspond to? This corresponds to the 'DC' offset 'w_0'. So, we don't even use the data. We simply construct 'X' as a column of 1s.

So in this case, we're looking at this matrix where a zeroth-order polynomial is simply this first column, and we have nothing else here. And, this is what we learn. We simply - this is the least squares solution, if all we're allowing ourself is a shift of the axis, a shift of the offset. Now, let's go back to what we've been discussing, which is what we had earlier discussed, which is a least squares where we have a linear relationship of the inputs to the outputs. In this case, we get this line. And our matrix, again, 'X', is simply a matrix of all 1s in the first column, and then the data in the second column, and then we perform least squares to learn 'w_0' and 'w_1', and we approximate the data with this function here. If we then want to expand this to a third-order polynomial, so in this case, our data looks look this where you have 1s in the first column, the original data in the second column, and then up to the original data cubed in the fourth column. And, we perform least squares linear regression with this matrix 'X'. We learn four weights in this case. And now, we map–we learn–we approximate the output as a third-order polynomial on the inputs. We get this sort of a function here, by the red line.

So, what we would do just to remind you is we would take our input, we would then predict this unmeasured value by taking the input, squaring it, cubing it, having these three numbers, and then saying that this value is equal to 'w_0', which is according–corresponds to the first column, plus 'w_1' times this value 'x' here, we'll call this 'x_{11}', plus 'w_2' times 'x_{11}' squared, plus 'w_3' times 'x_{11}' cubed, sum those three values up, and we would get this value along the red line. Okay! So, we can try to be clever and push this.

Instead of learning just a third-order polynomial, how about we try a ninth-order polynomial? So in this case, clearly things start to break down. And, why would this be? Because in this case, the matrix 'X' is 10 by 10 because the first column of 'X' is, again, all 1s. But, we have now nine more columns of the data up to a ninth-order polynomial. So, our matrix is now 10 by 10, meaning that we can easily learn a 10-dimensional vector that passes through all 10 of these points, because we are trying to predict a 10-dimensional output 'y' as approximately equal to 'x' times a weight vector 'w'. And, if we have 10 outputs and a 10 by 10 matrix, assuming that this is full rank, we can easily learn a vector that perfectly reaches or touches all of these points–that passes through all of these points. But, we see that in the regions in-between the points, for example here, we're going to be way off. And, this is an example of overfitting.

So, what we've done is we've overfit our data. So, we've basically, perfectly predicted our training set of 10 points. And as a result, because of this overfitting, we are way off on new points. And so, this is a

caveat about modeling and using more and more variables or more and more parameters to better model your training data. The better you model it, at some point, you'll start to overfit, and we'll discuss this a little more later, where you are now doing very well on your training set, but you are doing very, very poorly on your testing set, which would be a new point, for example, this one here. Okay! So, we've been discussing polynomial regression in one dimension. Let's just, extend this to two dimensions to see, you know, the logic of going to higher dimensions. And, hopefully after two dimensions, it's straightforward how you would do this in 'n' dimensions or 'd' dimensions.

So for example, if we wanted to do a second-order polynomial regression model in 'R^2', what we would have is for the 'y^{th}' observation, we would model that as approximately equal to an offset 'w_0' plus the original-plus the linear regression problem 'w_1' times the first dimension of 'x_i', plus 'w_2' times the second dimension of 'x_i'. So, this is the original linear regression problem we introduced at the beginning.

And then, we take those two dimensions and square them, and then, put two more weights on those squared values. With third-order polynomials, what we would have is the same thing, and then, we would take each of those values and cube them and then, put two more additional weights because we have two-dimensional inputs. And so in general, the width of this matrix 'X' that we're going to construct grows as the order of the polynomial times the-dimensionality of the input vector 'x' plus 1 for the offset. So, if we look back at the original problem of how we can map the years of–the amount of education and seniority versus a person's income using a second–a first-order polynomial, in this case, linear model in the inputs, we get a plane. We approximate the output as being a hyperplane, a noisy hyperplane, which is evaluated on the two initial inputs. If we now want to do a third-order polynomial instead, we get a function that can look like this, where we've simply taken the two-dimensional inputs, performed squarings and cubings of their values, and turned it into a six-dimensional problem instead of a three-dimensional problem, including the offset. So, what are some further extensions that we can do of this model? So, really as long as the function is linear in the weights, we can do any function of the inputs 'x' that we can dream up and construct the matrix 'X', where on each column, we perform some function of the original input vector 'x', little 'x'. So more generally, if we imagine that we have inputs 'x' that are in 'd' plus 1, meaning the first dimension is a 1, by definition, we can define a least squares linear regression function to be of this form, where we simply say there are 's' weights, where 's' is variable. We then take for each dimension function that we're going to perform, we take our input and pass it through a particular function, call it 'g_s'. So this function 'g_s', takes our input 'x_i' and outputs a one-dimensional output, which is some function, and multiply that by a weight. So for example, in the original problem that we looked at, we could have had 'g_s' simply takes the input 'x_i' and returns the 'S^{th}' dimension of it, meaning that this is simply taking a vector and picking out one of its dimensions, and that would give us the original linear regression problem that we looked at the beginning of the lecture. But, there are more examples.

So for example, we can take, we can define the 'S^{th}' function to be the square of the 'j^{th}' dimension, which is related to the polynomial regression that we've been discussing. We can define a function to be

a log of the input of a particular dimension. We can do something that's not a continuous function. We can have an indicator that is equal to one, if the 'j^{th}' dimension is less than some number 'a' or equal to zero, if it's greater than or equal to 'a'. Or, we could even compare dimensions of our inputs and define the 'S^{th}' function to be an indicator of the J^{th} dimension being less than the 'j' prime-th dimension. So, there's a wide variety of functions that we can perform. This is where domain knowledge, expertise would come into play, where you believe that there are certain relations between the dimensions of your data and the outputs. For example, you don't think that it should be linear in the input, you think it should be a logarithmic in the input, so, you can perform any function that you can define. And, as long as it's linear in the weights, 'w_1' through 'w_s', we can construct the matrix 'X', capital 'X', by putting the transformed vectors of each 'x_i' along the 'i^{th}' row, adding a 1 to the first column, and then solving the least squares solution, which always has this form. Of course, the caveat that we saw from previously with the ninth-order polynomial regression function was that the more functions that we use, the more data that we're going to need to avoid overfitting.