COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Week 8
Video Transcripts

https://anandology.com/python-practice-book/iterators.html

Video 1 (15:45): Maximum Likelihood

So now, let's look at the Gaussian mixture model in a little bit more detail. For the Gaussian mixture model, which is one specific case of the mixture modeling clustering framework that we're discussing in this lecture, we have this setup. So again, we let Pi be a 'K' dimensional probability distribution and now we have, for each cluster, for example the k^{th} cluster, we have a mean 'Mu$_k$' and a covariance 'Sigma$_k$' which defines the Gaussian– the Multivariate Gaussian for that specific cluster. When we generate data from a Gaussian mixture model, for the i^{th} observation we assign it to a cluster by generating the cluster assignment *iid*. from a discrete Pi distribution. So this has not changed at all. But in the specific case of the Gaussian mixture model, when we now want to generate the observation 'xi', we generate it from a Multivariate Gaussian using the mean Picked out by 'ci' and using the covariance Picked out by 'ci'.
So if we define Mu, and Sigma in boldface to be the set of means and covariances, our goal now is that given some data, we want to learn Pi, Mu, and Sigma. So that's the inference goal. We've defined the hypothesis of how the data got to us by defining the generative model. Now that defines a joint likelihood on the data given some parameters. And so once we have the data, our goal is now to go back and infer, what are the parameters that could have generated that data? So now we have the inference problem. There are several different ways that we could do inference for the Gaussian mixture model. In this lecture, I'm just going to focus on the maximum likelihood approach. So the objective in this case is to maximize the likelihood over the model parameters Pi, Mu, and Sigma. And we're going to treat 'ci' as an auxiliary data in the EM framework. So 'ci', the cluster assignment, now is going to be the additional hidden data that we introduce and then do EM. So we want to integrate out 'ci'.
Remember from the EM algorithm that we discussed last time, what that means is that we want to maximize the likelihood of all of the data 'x_1' through 'xn' given Pi, Mu, and Sigma. And because we hypothesize that it's iid, that simplifies into a product over the individual likelihoods of each observation given the parameters of the Gaussian mixture model. But notice that there's no cluster assignment here– 'ci' is missing. Where did it go? We integrated it out, so 'ci' can be written this way– or can be brought back into the model this way where we have the joint likelihood of the i^{th} observation and the cluster assignment, which cluster the i^{th} observation came from. But now we integrate or, in this case, because 'ci' can only take a discrete and finite number of values, we sum over all of those possible values to integrate them out. So this term here–right here–is equal to this term. So if we want to know what this is, we simply write out the joint likelihood according to our model and then sum over the possible values of 'ci' to find what this is.
So our goal now is to maximize this joint likelihood of the data given only Pi, Mu, and Sigma where we have summed the parameter 'ci' out of the model. However, it's difficult to do that. If you write out what this actually is and you take its log and you try to take the derivative with respect to Pi, Mu, and

Sigma, you'll quickly find that we're not left with a closed form solution. We can't solve it exactly. And so what we could do is we could derive a gradient method where we take the derivative and then step in that direction a little bit. But given our discussion of the EM algorithm last time, we're instead going to try to see if we can't use the cluster assignment 'ci' in an EM sort of framework for doing this maxiMum likelihood. So first, before we do that, the immediate question is why don't we just do maxiMum likelihood over the model that includes 'ci'? So why don't we keep 'ci' as well in the model as something that we want to infer? We want to do a point estimate of 'ci' in addition to a point estimate of Pi, Mu, and Sigma– because when we do that, when we write out the augmented joint likelihood where we include 'ci' as a parameter we want to do a point estimate over, we take the log of this and we take derivatives and solve.
What we would find is that we can optimize that very easily. We can get closed form updates for every parameter in the model, including 'ci'. So that is something that can be done. But what we would find by doing that is we would end up with a hard clustering model, meaning that when we wanted to update a particular cluster assignment, 'ci', maximizing that 'ci' would lead to it assigning the data point to the most probable cluster. So the maxiMum likelihood solution or the MAP solution, actually, for 'ci' is to assign the ith data point to the most probable cluster. And so we would have a hard-clustering algorithm. Our goal here is to integrate out the effect of the cluster assignment, which we're going to see when we do that in the EM framework is going to lead to a soft clustering algorithm that we anticipate should be more accurate and should be better than the hard clustering algorithm because we've integrated out one of the unknowns in our model so we have fewer potential local optimal solutions that we could converge to.
So we're goin...not going to derive the EM algorithm from scratch. I'll just give the outline in this lecture. However, we do–to repeat– we do treat these cluster assignment indicators as the latent data–the missing data in our model that we want to integrate over, so we include this thing. Therefore, we're going to use EM to maximize the log of this marginal likelihood where 'c' is gone by using the log of this augmented or extended joint likelihood to help us do that. So let's look at the equation we saw last time and how it is modified for this particular model structure. Last lecture, we saw that the generic EM objective looks like this. We want to do maxiMum likelihood over a model with parameters 'Theta$_1$'. So we want to maximize the log of the probability of data 'x' given parameter 'Theta$_1$', but that's difficult. And so we wrote this equality where we had this term here, which we called 'L'.
So this term we called 'L'. And this term here was the Kullback-Liebler divergence between 'q', and the conditional posterior distribution. And we introduce the additional variable to our model, 'Theta$_2$', and then integrate it out and we do a two-step procedure. So remember the procedures were to set 'q' equal to 'p'. Given that updated 'q' distribution, calculate this expectation. And then maximize this expectation over Theta1. And 'Theta$_2$' is gone because it was integrated out. Now, if we take this equality and we just write it using the Gaussian mixture model parameters and setup, we get something like this. So the log of the likelihood of all of the data given Pi, Mu, and Sigma -- because the data are assumed to be *iid*, that turns into the log of the product, which turns into the sum of the log of each individual likelihood. So we have a sum over all of the data of the log of the probability of each point given our Gaussian mixture model parameters. And now we introduce the hidden variable.

So what we're doing here is you can ignore this first sum the first time you look at it— just ignore this first sum. And now we're going to treat this term here as being like that term. And we're going to introduce c as this 'Theta$_2$'. So 'Theta$_1$' corresponds to these parameters. 'Theta$_2$' corresponds to 'ci'. And because 'ci' is a discrete variable, the integral is going to turn into a sum. So when you have continuous variables, you integrate. When you have discrete ones, you sum. That's...That's the difference. And so this integral over all of the values that 'Theta$_2$' can take turns into a sum over all of the values that 'ci' can take. But notice that, again, we have the data 'x'– we have 'xi' as mapping to 'x' here. For 'Theta$_1$', we're mapping our mixture model parameters to 'Theta$_1$'. For 'Theta$_2$', we're mapping 'ci' to 'Theta$_2$' here. And we have 'ci'. And then, for the integral over all of the values 'ci'...'Theta$_2$' can take, we're mapping that to the sum over all of the values that 'ci' can take– and similarly, with the KL divergence. This term maps to this term. And now because each of these individual terms are equal to each other, when we sum over all of the data on the left-hand side and sum over all of the data on the right-hand side, again, nothing changes. We have the left-hand side is equal to the right-hand side.
Let's now look at the EM algorithm for this model. Remember, with the EM algorithm our goal is to first set the KL divergence equal to zero. So we have a sum over 'n' different KL divergences here. Each KL divergence is calculated for– we have one KL divergence for one data point and the ith KL divergence is equal to this term right here. And so to set the entire KL divergence to zero, we can set each individual KL divergence to zero. So we want to set 'q' of 'ci' equals 'k', which is our probability of 'ci' equaling 'k' to be equal to the conditional posterior probability. So this should be also evaluating 'ci' is equal to 'k'. So that's the first of part of the E-step. We set this distribution 'q' 'ci' equal 'k' to be equal to the conditional posterior distribution of the ith data point being assigned to the kth cluster given the value of the ith data point and given the parameters for the Gaussian mixture model at the current iteration. To do this, we use Bayes' rule. So Bayes' rule says that the conditional posterior is proportional to the prior of... assigning the ith data point to cluster 'k' times the likelihood of the observation– of the ith observation given that it was assigned to cluster 'k'. So we see that this term, the prior probability of assigning a data point to cluster 'k', is just equal to 'Pi''k'. And the likelihood of the ith observation coming from cluster 'k' is equal to a Gaussian where we evaluate the likelihood of 'xi' given the mean and covariance of the kth cluster. And then, to turn that into a probability distribution, we simply normalize the right-hand side. So this is a proportionality.
To turn this into an equality, we normalize this thing by dividing over the sum of all possible values that 'ci' can take, meaning we sum this numerator over all values 'k'. So we take this, we divide it by the sum over all possible values for the cluster assignment. And we get–we define this to be 'Phii' of 'k'. So this is serving identically the same purpose as with the weighted K-means algorithm. The only difference is that now we have a firm probabilistic interpretation of what we're doing. So that's just the first step of the EM algorithm. Remember that the second step–I'm sorry– the first step of the E-step of the EM algorithm. The second part of the E-step, after we update each of these 'q' distributions, is to calculate this expectation. So, in that case, we want to calculate this thing where we are now summing over the possible values for each 'ci'. And so that turns into this. In the previous slide, we had a sum where we had this term here. Now, we take this term and we replace it with what we defined it. So this is just our shorthand. But now, we have the sum over each observation where we evaluate the log of the likelihood of the data coming from each cluster– so that's this term, the log of the likelihood of the ith

data point coming from the 'k^{th}' cluster–plus the log of the prior probability of it coming from the k^{th} cluster. So that's equivalent to the log of the joint likelihood of the i^{th} observation coming from the k^{th} cluster and it coming from any point coming from the k^{th} cluster.
We take that log and we Multiply it by the weight that we assign to the k^{th} cluster and sum it up. So this term is simply equal to this term where, again, we are going ignore this part because it's simply a constant. It's not going to change anything. So we can write this as the log of 'p' minus the log of 'q'. We can separate these two terms and then this term is not going to involve the parameters that we're interested in updating, so we're going to ignore it and simply look at the log of this term here. And again, we replace 'q' with what we've defined 'q' to be. Okay, so we can actually write this out. We can expand what this is. We can write out the Gaussian form for it and then get this function that we want to maximize– remember, we called this 'L' last time, not 'q'. And so then we maximize 'L' with respect to Pi, 'Mu_k', and '$Sigma_k$' for each 'k'.

Video 2 (18:57): Expectation-Maximization Algorithm

So EM is short for expectation-maximization, those are the two steps within each iteration. We're going to have an expectation step, that's like one part of the coordinate of ascent algorithm and then a maximization step, which is like the second part of a coordinate of ascent algorithm within one iteration. Okay, so I want to discuss this in the context of a motivating example...simple example, where EM can be used. And then we're going to see how the EM algorithm can be used for this problem. So this is the problem where we have missing data, and in particular, a certain type of missing data with a certain model assumption. So, let's imagine that we have data 'x_1' through 'x_n', and 'x_i'– each 'x' is in 'R^d'. Except that now a vector that has missing values. We have a vector, each vector is in 'R^d'.
But for each vector, there might be some values that are missing. And these values don't form any pattern. Some data can have many values missing. Some data can have no values missing. And the dimensions along which they're missing is a random...is totally random. So there's no pattern in the way that the data is missing. So let's split this vector 'x_i' into two parts. We'll call ''x_i'^{o}', the observed portion of the vector. So that's the subvector that we get by taking the observed values of 'x_i' and forming a smaller vector corresponding to the observed values. So ''x_i'^{o}' only as measured values in it. And then, ''x_i'^{m}' is a...the mission...the missing portion of the of the vector 'x_i'. So this is a subvector where we don't know any of the values in it. And the dimensionality of each of these can change for each 'i'. So we have to make a model assumption in order to use the EM algorithm, otherwise it's not going to apply. And our assumption is that each of these vectors 'x_i' are generated *iid*. from a Multivariate Gaussian with mean Mu and covariant Sigma. So we're going to make this modeling assumption.
Each 'x_i' is generated *iid*. And then after it's generated, some of the values, for whatever reason, go missing. Now, our goal is going to be to solve this maximum likelihood problem, where we want to find the maximum likelihood values for mu and Sigma, where we only can take the likelihood over the observed portion of the vector. Because if we don't have any of the data in a missing portion, then we can't factor it into the likelihood. So we want to maximize the log of the joint likelihood, only over what we observe. So we'll quickly see that this is tricky. It's not impossible actually, but it's tricky. Because

each of these ''x_i'^{o}'s can have different dimensionality and those dimensions can correspond to different subsets of the dimensions in Mu and Sigma. So it's not something that we can simply–directly take a derivative of and set to zero. However, we think that if we knew the missing portion, so if we knew ''x_i'^{m}', if we had values for those missing dimensions, and therefore we knew all 'x_i', then we could easily maximize this thing. So we can take the arg max of the log of the joint likelihood of the observed and missing portion over Mu and Sigma. That's actually what we've already seen on a previous slide. But that assumes that we know the values in this missing portion of the vector.
So I want to connect this now to a more general set up. What we're going to return to in a later slide is a...is an algorithm, an EM algorithm, for optimizing this thing over Mu and Sigma, and also simultaneously imputing the values of all of the missing portions of each vector. So the EM algorithm is going to give us a technique for actually optimizing this thing and also filling in all of these missing values. However, I want to discuss it first in the more general set up. So what we are now going to imagine is we have two parameter sets, 'Theta$_1$' and 'Theta$_2$', where 'Theta$_2$' in this model relates to 'Theta$_1$' in the following way, that the marginal distribution of 'x' given 'Theta$_1$' is equal to an integral of a joint distribution of 'x' and 'Theta$_2$' given 'Theta$_1$' over 'Theta$_2$' So the distribution on 'x' given 'Theta$_1$' is equal to the integral of this joint distribution. So this 'Theta$_2$' is like a hidden variable or a missing latent auxiliary variable that we don't get to observe in the data. So for example, if we want to return to the missing data problem, we can see that this likelihood what we want to maximize over the observed portion of the vector and where 'Theta$_1$' could be the set Mu and Sigma is equal to the integral of the joint distribution of the missing and observed portion. So this is a Multivariate Gaussian, with mean Mu and covariant Sigma.
But now we integrate out a missing portion of the vector. So we can write this distribution as an integral over the missing portion of the vector, and that's equal to– we can show, I'm not going to derive it, but we can show that that is equal to a Multivariate Gaussian where the mean is equal to the portion of the mean vector restricted to the observed dimensions and the covariance is equal to the submatrix formed by only considering the observed dimensions. So this is a common derivation and very useful derivation of working with Multivariate Gaussians, where if we want to integrate out a subset of the dimensions of the Multivariate Gaussian, we simply get a Gaussian back with mean and covariance equal to their appropriate subsets.
So now we need to define an objective function, as always, that's going to give us what we want. So, what we want is a way to optimize the marginal distribution returning to the more abstract parameters, and notation that allows us to maximize this distribution of 'x' given 'Theta$_1$' over 'Theta$_1$', where 'Theta$_2$'is nowhere to be seen. But that it uses this joint distribution; it uses this additional variable 'Theta$_2$' to help us maximize this thing for computational reasons. So again, we're going to stay abstract, and make it more concrete in the context of missing data later. To achieve these two objectives using the EM algorithm, we form this equality. And so I'm going to pick this apart in the following slide. But for right now I just claim that the left-hand side, which you notice is what we want to maximize over Theta$_1$ is equal to this right-hand side. And notice that the right-hand side involves this joint likelihood over an additional variable 'Theta$_2$'.
It also involves the conditional posterior distribution of 'Theta$_2$' given 'x' and given 'Theta$_1$'. And it also includes this additional probability distribution 'q' on 'Theta$_2$'. Okay, before I pick this apart, I want to

make a few immediate comments. First what we're going to do, what the EM algorithm does is instead of working with this left-hand side, it's going to work with this right-hand side. And it's going to specifically define an algorithm for working with this right-hand side, such that we end up with a value for 'Theta$_1$' that gives us a local maximum of this left-hand side. So we're going to actually optimize this left-hand side without ever using it, without ever working with it. We're going to optimize this left-hand side by working with this right-hand side. A second thing to notice is that this 'q', this mysterious 'q distribution' on 'Theta$_2$', is a probability distribution. And in principle it can be any probability distribution that we want on 'Theta$_2$', as long as it's defined on the values that 'Theta$_2$' can take. However, the EM algorithm is going to tell us how to set this 'q' distribution. Also, if we're looking at this right-hand side and saying that we want to somehow work with this thing because it's easier than working with the left-hand side, that's going to imply that this missing variable, 'Theta$_2$', that the conditional posterior of it can be calculated given 'x' in 'Theta$_1$'.
So this conditional posterior we're going to assume is going to be something that we can calculate in close form very easily. Okay, so first let's show that the quality is actually an equality. This is just a copy of what was on the previous slide. We want to basically prove that this side is equal to that side. So first to do this what we're going to do is combine these two things. Recall that the log of a product of two terms, so we have this term and this term. The log of the product of two terms is equal to the sum of the log of each term separately. And so we've taken this integral and this 'q', and combined this term here, and this term here. And then written the sum of these two logs as a log of a product of what's inside the logs. So these two terms are equal to each other. We immediately get to cancel these two things out. Next, let's remember some rules of probability. We saw in a previous lecture that if we wanted to find the probability of 'a' and 'b', given 'c', we can equivalently write that as being equal to the probability of 'a' given 'b' and 'c' times the probability of 'b' given 'c'. so we can write this joint probability into a conditional probability times another conditional probability. And so since this is true, we can now solve for 'b' given 'c' by dividing both sides by this term to find that the probability of 'b' given 'c' can be written in this way. So notice that there's an 'a' on this side, but there's no 'a' on this side. Okay, so now I notice that I had a typo here. Let 'a' equal 'Theta$_2$', 'b' equal 'x' and 'c' equal 'Theta$_1$'. Plug those into the right-hand side here. And you'll notice that the right-hand side is equal to this term. And then using the rules of probability, we can say that the right-hand side is equal to the left-hand side.
So, we can take this term and replace it with this term, where b is equal to 'x' and 'c' is equal to 'Theta$_1$'. And we get this here. So this...this complicated term simplifies into this term here. And then finally, there's no 'Theta$_2$' here, so we can bring this term out front... we can bring this term out front. And then whatever 'q' is equal to. And so this is why it doesn't matter what we define 'q' to be equal to, as long as it's a probability distribution, the integral of a probability distribution equals one. And so, we're left with this, the left-hand side is equal to the right-hand side trivially. So we've shown that this term is equal to this term. So now the question is, why in the world would it be easier to work with this right-hand side, this much more complicated right-hand side, than this much simpler looking left-hand side. And so that's the genius of the EM algorithm. Now, let's focus in on these two terms a bit more. So we've taken our log of our likelihood of 'x' given 'Theta$_1$', and we've written it into the sum of two different terms.
First let's look at this term here, the first term. Notice that once we define 'q' of 'Theta$_2$', this term is actually only a function of 'Theta$_1$', because we've integrated out 'Theta$_1$'. It's also a function of the data,

but we plug that and we don't change the data, we change 'Theta_1'. And so this thing actually even though it looks like it has 'Theta_2' in it, once we solve it, 'Theta_2' will be integrated out and we'll only have a function here of 'Theta_1'. So that's the crucial observation number one. The second crucial observation is that this second term is a function that has a...that's very famous, widely used and has a name, it's called the Kullback-Liebler divergence. So this can be thought of as like a distance measured between two probability distributions. It's not a proper distance because it doesn't satisfy the triangle inequality. However, it can be thought of as a distance for two key reasons. First, we can show that this term is greater than or equal to zero. No matter what the probability is here, and no matter what the probability is here, this integral will be greater than or equal to zero. So that requires a proof. It's not at all obvious. You can't look at it and see that that's true. We need to have a proof to show that this is the case. But the proof does show that this term is always greater than or equal to zero for every possible probability distributions that you plug in here.
Similarly, it's only equal to zero when these two distributions are equal. So, when 'q' is equal to 'p', when this distribution is equal to this distribution, then the KL divergence equals zero. And that's the only case where it equals zero. And then more intuitively speaking, we can think of when the distribution 'q' overlaps more with the distribution 'p', the KL divergence gets smaller. And then as the two distributions get farther and farther apart, the KL divergence gets bigger. But for our purposes, the only two important properties are that this is always positive or equal to zero and it only equal zero when this distribution and this distribution are equal to each other. Okay, so we're going to see how we can use those two terms to optimize the log likelihood of 'x' given 'Theta_1'. But before we do that, I first want to ask, what does it mean to iteratively optimize this function with respect to 'Theta_1'.
\So what is it that we're looking for when we say that we want to maximize this thing. When we say we want to come up with an algorithm that's going to maximize this thing. So one way of thinking about it is to say that we're looking for an algorithm, we're looking for a sequence of rules to follow, such that we get a sequence of values, the algorithm will output a sequence of values for 'Theta_1', where the log of the probability of 'x' given 'Theta_1' at iteration 't', for any 't', is greater than the value of the log of the probability of 'x' given the value of 'Theta_1' at the previous iteration. So these values...these values for 'Theta_1' are being output to us as a function of iteration, the first, second, third, and fourth, etc., iteration. And if we plug in the sequence of values, we're monotonically increasing this log likelihood. So that's intuitively what it means to optimize this thing. We also have to then prove that that sequence is converging to a local optimal solution. In this lecture what we're going to do is show how the EM algorithm gives us this. It's going to give us a sequence of steps to follow so that, we are continually finding new values for 'Theta_1' that are increasing this log likelihood. But we're not going to discuss 'Theta_2' that shows that the sequence also converges to a local optimal. And so intuitively if you want to know what that means, you can think of it this way. If this is what we're trying to maximize, so imagine that this is the maximum point right here. We want a sequence of values, 'Theta_1'^{1}, and then a second value 'Theta_1'^{2}. We want a sequence of values such that we're always increasing– we're always finding new values that are better than the old values.
However, you can imagine that there's a point here where we're continually having our distance to this point. In that case we're always finding points that are monotonically increasing the objective function, however, it's actually converging to this point here. So, that's like a thought experiment that you can do

to show that it's not enough to say that you're always finding better values. You also have to show that those values are converging to a local or a global optimal value.

Video 3 (17:15): The EM Algorithm

I want to first define the algorithm, the EM algorithm, I want to define the sequence of steps that you follow, it's basically two steps, an E-step and an M-step. Then I'll prove in the next slide that by following that rule we're monotonically increasing the objective function. All right, so I've written the EM equality one more time. Remember what we want to do is, maximize this left-hand side over 'Theta$_1$', we want to find the value of 'Theta$_1$' that maximizes it. And we're going to work with the right-hand side in doing this. So we're going to work with this term, call this 'L' and it's going to be a function of 'x' which is our data and a function of 'Theta$_1$'' because 'Theta$_2$' again is integrated out. So this term 'L' is a function of the data and 'Theta$_1$' only. And then here we have the 'Kullback-Leibler Divergence'. And even though these two values can change for different values of 'q', for different distributions 'q', the sum of the two values is always equal to the same thing. If we fix 'Theta$_1$' then the sum of these two values equals the same thing for all...all distributions 'q', so we're going to use that fact.
Now the EM algorithm follows these two steps. Let's focus on one iteration, iteration 't' and say that if we're in the tth iteration, what do we do to update 'Theta$_1$' for the next iteration, 't' plus one. So, in that case we're going to assume that we're given a value for 'Theta$_1$' at iteration 't', at the first iteration–this could perhaps be randomly initialized, but at the tth iteration we have a value for 'Theta$_1$'. And our goal now is to find the value for 'Theta$_1$' at iteration 't' plus one. So this is our input, this is going to be our output of the tth iteration. So we do two steps, the first step is the E-step and this actually has two substeps. The first part of the E-step is to update the 'q' distribution. And so I put a subscript on 'q' to indicate what iteration it corresponds to because this distribution is changing for each iteration. So the 'q' distribution that we define on 'Theta$_2$' at iteration 't' is going to be equal to the conditional posterior of 'Theta$_2$' given 'x' and given the value of 'Theta$_1$' at iteration 't'. Again we're working only in an abstract setting here so I can't tell you at this level that we're speaking what exactly these distributions are.
But we assume that for the model we're working with we can calculate this thing. So we set this 'q' distribution to be equal to the conditional posterior of 'Theta$_2$' given the value of 'Theta$_1$' at iteration 't'. Then the second part is to calculate this term using that exact 'q' distribution that we updated for the iteration. So we calculate 'L' and I'll put a subscript 't' here in order to indicate what the 'q' distribution is that we use, which is equal to this integral. So we take the expected log jointly likelihood of 'x', and 'Theta$_2$' given 'Theta$_1$' where we integrate out 'Theta$_2$' using this 'q' distribution. And then we subtract this term here, which corresponds to splitting these two terms apart. Notice that this term is a constant as far as 'Theta$_1$' is concerned, so we're just going to ignore this part. This is the only part that is a function of 'Theta$_2$'. So now the E-step is completed. We have this term 'L's' of 't', it's a function of the data 'x', and Theta one and it was calculated using the 'q' distribution updated in iteration 't'. And then the second part is the M-step, the maximization step. And in this step, we now treat this function as a function where we're free to change 'Theta$_1$' to anything we want and so we maximize it over 'Theta$_1$', so that's the M-step. And the output of that is the value of 'Theta$_1$' for iteration 't' plus one.

So those are the two steps that we follow. It seems like a specific sequence of steps to follow, so how do we know that it's doing something good. In this slide, I have the proof of monotonic improvement. So I have the proof that by following those two steps in the previous slide that the value of this objective function evaluated at 'Theta$_1$' for the tth iteration is less than the value of the objective function using the value of 'Theta$_1$' at the 't' plus first iteration. So what we're going to do is show that this term is less than or, equal to this term. Remember, this is what we want to maximize over 'Theta$_1$'. And we have this sequence in between that's linking them. So this is another example where we're using analysis just like in the boosting lecture to prove that our algorithm is doing something good. The way that we do this is we use the EM equality, so we know that the log of the probability of 'x' given 'Theta$_1$' at iteration 't' is equal to the first term, the 'L' function. So you can go back to the previous slide or perhaps write it down to plug in the values here, but I'm going to use this shorthand just to make it look more cleaner and more compact. So this log likelihood is equal to the 'L' function evaluated at the same 'Theta$_1$' at iteration 't', plus the KL divergence between 'q' and the conditional posterior of 'Theta$_2$' given 'x'...given 'x' and given 'Theta$_1$' at iteration 't'. And so notice I don't have a subscript 't' here and I don't have a subscript 't' here because this equality is true for all probability distributions that we can place on 'Theta$_2$', any probability distribution this equality the sum of these two terms is equal to this term.
But the E-step of the EM algorithm specifically sets 'q' to be equal to this conditional posterior. But the first step of the EM algorithm is to actually pick a value for 'q'. And so we let 'qt' of 'Theta$_2$' be equal to 'p' of 'Theta$_2$' given 'x', and given 'Theta$_1$' at iteration 't'. So we actually set 'q' to be equal to this particular distribution. And so what does that...what does that do to this right-hand side? Well it doesn't change the left-hand side, so it doesn't change the sum of the right-hand side because we haven't changed 'Theta$_1$'. And the sum of the right-hand side is the same for all values of 'q'. However, by setting 'q' to be equal to this distribution at iteration 't' we make this term equal to zero because we've set the KL divergence between these two distributions to be equal to zero, which we know occurs when we set this distribution equal to that distribution. Therefore, this term is equal to this term. So notice that I've now put a subscript 't' down here to indicate that I have calculated this thing using the distribution 'qt', so the 'q' distribution at iteration 't'. And notice that I don't need to write this term anymore because I know that it equals zero because these two distributions are the same. So this is the E-step, these two lines constitute the E-step of the EM algorithm. Now let's look at the M-step, what do we do with the M-step.
We simply take this algorithm, we don't change the 'q' distribution on 't' on 'Theta$_2$' and so the subscript 't' here is not going to change because we use the same 'q' distribution. However, we let this distribution now vary in 'Theta$_1$', we don't force it to be evaluated at ''Theta$_1$' for iteration 't' anymore we let it free a function of 'Theta$_1$'. And now we maximize it over 'Theta$_1$'. So we know for a fact even though we don't know what the specific value is that we get for 'Theta$_1^{t+1}$'. We know that whatever value it is, it has to be greater than the value at iteration 't' here, right. So we have not proved that this is greater than this, but we have shown already that this has got to be greater than this, because we literally maximize this signal over 'Theta$_1$' and set that to be equal to 'Theta$_1^{t+1}$. So if the value of Theta$_1^t$ doesn't maximize this thing then this value is going to find the setting for 'Theta$_1$' that does maximize it and so it'll be greater then. If 'Theta^{t+1}' is equal to 'Theta^t', then it's equal and so that's where the equality could come in. Okay so this is the M-step. So the E-step took this right-hand side, put a

particular 'qt' in there to get rid of the KL divergence, but we still have inequality. The M-step then let this value...this variable 'Theta$_1$' be free to change and we changed it to the value that maximized this term here.
So we now know that the update of this first term of the right-hand side is greater than this term. However, we want to compare this term with this term, we don't want to compare this term with this term. So how can we now write this, how can we now achieve that goal? So notice that we can add a positive number to this term and we aren't going to change the equality. So we can now add something positive to this thing to again have an inequality where we have now this sum of two things is greater than or equal to this value here. So the term that we add to this is going to be very cleverly chosen, we choose the KL divergence between 'qt'. So remember, this is the 'q' distribution that we updated using the value of 'Theta$_1$' at iteration 't'. But now it's the KL divergence between that 'q' distribution and a conditional posterior distribution of 'Theta$_2$' given 'x' and given 'Theta$_1$' where we evaluate it at Theta, at the value we get at 't' plus one. So we plug in 'Theta$_1$' at iteration 't' plus one here. So, this term that we've now added is literally something that we arbitrarily chose to add. There’s no reason why we have to add this term to the right-hand side we're only doing it to help us prove what we want to prove. So we add this term, we know for a fact that this term has to be greater than zero because we know that this distribution is not equal to this distribution, assuming that 'Theta$_1$' changed between iteration 't' and ''t' plus one'. We know that this thing is equal to this term here and so if Theta one is changed, then this distribution is not equal to this distribution and so the KL divergence now is positive.
So we've added a positive number to the right-hand side and therefore we don't have to worry about the inequality switching directions because we've simply taken whatever number this is and added something positive to it. And so the sum of those two things has to be also greater than or equal to this term up here. Okay finally, we want to simplify this thing. So first let's now let 'q' suddenly be free to change and now we get, we simply remove the subscripts and we let 'q' be anything that we want it to be. However, we note that these two things add up to the same thing for all values of 'q'. And then finally we can recognize that these...that the sum of these two terms is equal to the log of the likelihood of 'x' given the value of 'Theta$_1$' at iteration ''t' plus one'. So there's this sequence if we follow it along, we find that this term is less than or equal to this term. And so the 'EM algorithm' has defined a method for giving us a sequence of values for 'Theta$_1$' that monotonically increasing the log of the likelihood. It might help to look at a visual representation of this to develop some more intuition about what's going on here.
Here on the right-hand side, just for reference I've shown the log of the likelihood of 'x' given 'Theta$_1$' that we want to maximize is equal to the 'L' plus 'KL' where 'L' is this term, 'KL' is this term. So this is what we've already been working with. Now this value, this term has some particular value for setting 'Theta$_1$'. So imagine that we plug in a value for 'Theta$_1$' here. It has some value which we're going to put here at this line. So imagine here that we've got the real line, we have zero here, these are negative numbers so these are less than zero. Down here we're going to put an arbitrary floor purely for visualization purposes, for no other reason we'll put an arbitrary floor down here much lower than all the other points, much more negative. And so the log of 'p' of 'x' given 'Theta$_1$' for a particular 'Theta$_1$' has a particular value. We don't know what it is but it has a particular value. We also know that that's

equal to 'L' plus 'KL'. And so 'L' because 'KL' is greater than or equal to zero this term 'L' has to be less than this term. So we have this is less than 'L' or equal to 'L' simply because this term has to be positive. So if we wanted to use the same visualization we could put a line here for what 'L' is equal to and if it helps to view it from going up from some floor you could also view it going up that way. So, we have the log of 'p' of 'x' given 'Theta$_1$' is here, that's equal to this term 'L' plus 'KL', which is a positive term, so this is the 'KL' term. I sum this value plus the 'KL' value and I get the log of 'p' of 'x' given 'Theta$_1$'. Okay so I now do EM, what does the E-step of EM do? Well be E-step is going to take the 'q' distribution and set it to the conditional posterior, I'm going to make 'KL' equal to zero. So the E-step takes 'KL', sets it to zero, meaning that it finds a value for 'q' such that this term 'L' is equal to this term log of 'px' 'Theta$_1$'. So I've taken 'KL', by setting it to zero, I've increased this term such that this is equal to that. So that's what E has done, the E-step has done. Now what has the M-step done? It's taken this value, the old value of 'L' and found a new value for 'Theta$_1$' such that this term increases. So by increasing that term it's pushing the ceiling up, but at the same time, the KL divergence has also become non-negative. So we have even more slack here. And so we've taken our objective function, and found a new value that's pushed it upwards like this.

Video 4 (14:42): EM for Missing Data

We're going to return to the problem that we discussed previously, where we have vectors generated from a Multivariate Gaussian with a mean and a covariance, but some of those vectors have missing values. So to give you a visualization, you might imagine something that looks like this where we have a data matrix, we imagine each of these columns is a Multivariate Gaussian with the same mean and the same covariance except some of these vectors now have missing values. So, we don't have a value for these white boxes, so it's somehow corrupted. And now our goal with this is to do a few things. First, we would–we might want to find the maximum likelihood values for Mu and Sigma given this data where we take the missing points into consideration. Another goal could be to fill in the missing values or to somehow learn what these missing values given the observed values that we have and to do it in some sort of an intelligent way. And so next, we're going to discuss an EM algorithm that's going to do both of these things simultaneously.
It's going to allow us to maximize the log of the likelihood of the data only taking the measured points into consideration over these two unknown parameters. And then the 'q' distributions that we're going to learn which we discussed previously are going to be conditional posterior distributions on these missing data points, which is going to tell us or give us a probabilistic statement about what we believe these values to be. Up until now we used this generic notation where we assume we had some parameters 'Theta$_1$', and some additional parameters 'Theta$_2$' that we added. So if we want to see how this thing translates to our missing data problem with the Multivariate Gaussian likelihood, we end up with something that looks like this. So, we can basically follow exactly the same procedure as before to show that the left-hand side here is equal to the right-hand side. If you look at the left-hand side, what it's equal to is the log likelihood of all of the observed portions of the vectors given the mean and covariance. So remember that 'x_i'^{o} is the subsector formed of 'x_i' by just considering the observed

portions of it. And then, because we made an *iid* assumption, the log of the likelihood of all of the data is equal to the sum of the logs of the individual likelihoods. So Mu and Sigma are going to be our 'Theta$_1$'. 'Theta$_2$' now is going to be the missing portion of the vectors. And also notice that because we have 'n' observations, we're taking these two sides and actually doing the breakdown over each individual observation. So this term corresponds to that, this term corresponds to that, and this term corresponds to that. But now because we have 'n' different observations and they're *iid*, we sum over the left-hand side and that's going to be equal to the sum over the right-hand side over our individual observations. So here now, our joint likelihood is over the observed portion of the ith data vector and the missing portion of the ith data vector. So this is the unknown missing variable in our model that corresponds to 'Theta$_2$' in the previous discussion. To derive an EM algorithm for maximizing this thing, we first do the E-step and remember that that's two substeps within the E-step. So, the E-step first requires us to take our 'q' distribution. In this case, we have a 'q' distribution on 'x_i' over the missing portion of the vector for each 'i', and set it equal to the conditional posterior of the missing portion of the vector of the ith vector given the observed portion of the ith vector, and given the mean, and the covariance using the current value for Mu and Sigma. This is another standard calculation with Multivariate Gaussians that's useful in many different contexts.

We actually used it for the Gaussian process previously, and now we're going to use the same calculation for a different purpose here to see how we can calculate this conditional posterior for the ith vector. First, let's get some new notation. So the vector 'x_i', the ith vector written in its complete form–we're going to say–can be broken down into the observed portion and the missing portion. And for simplicity, we're going to assume that the observed portion is on the top and the missing portion is on the bottom part. Now for different values of 'i', the missing [inaudible] is going to be different, so we can't make this kind of assumption for every vector 'i'. If we make it for one value, for one vector 'x_i', then for all the other vectors it might not hold that we have the observed portion in the top half and the missing portion in the bottom half. However, for notation purposes, if we derive this for one value of 'i', we can see how we do it for all values of 'i'. So let's focus on one value of 'i', and make this simplifying notational change where we put the observed portion on the top, and the missing portion on the bottom. Okay, and then this vector, now we could write as being generated from a Gaussian where the mean...mean corresponding to the observed dimensions and the mean corresponding to the missing dimensions also broken up like this. And the covariance is broken up into these four blocks where the first block is the covariance restricted to the observed dimensions.

This block diagonal is the covariance restricted to the missing dimensions. And then these two terms are the covariances of the observed dimensions with the missing dimensions. So we take our covariance, and we simply chop it into blocks according to how we've chopped this part up. Then a standard calculation with Gaussians that's worth deriving but we're not going to derive it in this course is that the conditional posterior of the missing portion of the vector given the observed portion, and given the mean Mu, and covariance Sigma is still a Multivariate Gaussian with mean 'Mu$_i$ hat' and the covariance 'Sigma$_i$ hat'. So the hat is now introduced. Where we define 'Mu$_i$ hat' to be this function of what we know. So we notice that, this function has the missing portion of the mean vector, it has the covariance sub matrices of the covariance. It involves the observed portion of the data vector and also the mean again of the observed portion and similarly with the covariance. So, these–it might look complex but the

thing to realize is that we can calculate this. We'll have a current value for the vector Mu, and Sigma in our algorithm. At the current iteration, we'll have a value for these things, and so we can actually calculate these functions to get the conditional posterior distribution of the mean, and the covariance back on the missing portion of the vector.
So we still have the second part of the E-step, in the first part we calculated this conditional posterior distribution for each value of 'i', so for each data point we have a separate conditional posterior distribution on the missing portion of the vector. If there is no missing portion of the vector, then we simply don't–that 'q' distribution is removed from the model, it doesn't appear. Now we need to take the E-step. So, the E-step is the expectation of the log of the joint likelihood of the observed and missing portions of the vector given the mean and covariance using the 'q' distribution of the missing portion. So it looks a little bit complicated to write out so I'll just focus in on the key term. We have to calculate this expectation where we have this complete data vector, the mean and the covariance. We can also write this as the trace of the covariance times the outer product of those two vectors, so these two terms are equivalent. And then bring the expectation inside, so we need to calculate this expectation.
Okay, so we calculate this integral over the missing portion of the vector 'x_i', which is something that we can do in closed form. To simplify it, I'm simply going to say what we're going to need to output from this term. So for the i^{th} observation let's let 'x_i' hat be equal to a vector where we take the missing values of 'x_i' and replace them with 'Mu$_i$ hat'. Remember that 'Mu$_i$ hat' is the 'q' distribution, the conditional posterior distribution on the missing portion of the vector of 'x_i'. So we fill in the missing terms in 'x_i' with the mean of the conditional posterior distribution on 'n'. And let's let this matrix 'V_i' hat be a matrix of zeros, that's the same dimensionality as 'x_i'. So if our original data is in R^d then 'V_i' hat is going to be a 'd' by 'd' matrix. We first fill it in with all zeros and then for the submatrix that corresponds to the missing portion of the vector 'x_i' we fill those values in with the covariance matrix that we got from the conditional posterior distribution of the missing values. So, I'm simply eliminating all the derivations from my slides, and jumping to the solution. So our solution is going to involve this vector and it's going to involve this matrix for each observation, so for each value of 'i'. So we've calculated the conditional posterior distribution on all of the missing portions of the vectors that we have. We've taken the E-step to calculate this function. We can now take the derivative of it with respect to Mu, and Sigma and solve. And what we'll find when we do this is that the update for the mean vector is the average of these 'x hats'.
So we sum up all the 'x hats', divide by the number of data, that's the update for mean. And if you remember what 'x_i' hat is. 'x_i' hat is equal to 'x_i' on the measured dimensions and on the missing dimensions from 'x_i', we fill those dimensions in with the mean of the conditional posterior on those dimensions. So the conditional posterior on the missing portion was a Multivariate Gaussian. We take the mean and we fill in the missing data with the mean. And so this term looks identical to maximum likelihood when we have all the data where the only difference is that the missing data we've now filled in. And then again the covariance looks very Much like the maximum likelihood solution when we have all the data. The only difference is that we've taken, we first filled in all of the missing dimensions using the mean of the posterior distribution on 'x_i', the missing portion of 'x_i', and we've added this additional vector matrix 'V' for each observation 'i', where the submatrix of 'V_i' is going to be non-zero on the dimensions that correspond to the missing data, and it will equal zero on the dimensions that

corresponds to the measured data. That simply reflects the fact that we are certain about the measured dimensions so there's no variance, but we're uncertain about the missing dimensions and so there's some covariance for those dimensions.
So we have our update for Mu, and Sigma and then we can return to the E-step where we calculate the conditional posteriors on all of the missing portions of each vector. And when we do that we iterate that process of going from 'E' to 'M', and when we go back and forth like that we get a sequence of means and covariances that look something like this when we evaluate the objectives. So when we evaluate the log of the marginal likelihood, we get a monotonically increasing function that eventually will converge and we monitor this thing, and when the marginal improvement is very small we can terminate the algorithm because it's converged. And out of this what we get is a vector Mu, and a covariance Sigma that is the maximum likelihood solution. So we have found a maximum likelihood solution for the original problem that we cared about where we've integrated out all the missing data.
And we also get a conditional posterior distribution on all the missing dimensions. And so this is going to allow us then to say something about the missing parts of the data that we have. It says out what the mean is of the missing portion, so we can just fill in the data with the mean if we want to. But we also get uncertainty, a measure of uncertainty of the missing data in the form of the covariance of the Gaussian that corresponds to this 'q' distribution.

Video 5 (18:01): Soft Clustering vs Hard Clustering

So, the hard clustering model that we discussed, K-means, goes like this. We're given data: 'x_1' through 'x_n', where each point is in 'R^d'. And we want to minimize an objective that looks like this, where we sum over each data point the square distance to the cluster centroid that it was assigned to as encoded by the parameter 'c_i'. So 'x_i' is assigned to one of 'K' clusters. And 'c_i' indexes what cluster that is. So we sum up the total number of square distances, and that's our objective that we're trying to minimize. When we derived our algorithm for minimizing this, we saw how we could do a coordinate descent algorithm, where we first update all the cluster assignments 'c', followed by an update of the cluster centroids, Mu. So when we updated 'c', what we did was for each data point– for example, the ith data point– we assigned that point to the cluster that it was closest to in the Euclidean sense. So that's what this is doing. Then when we allocate all of these data points to their nearest clusters, we update the cluster centroids by simply averaging–for the kth centroid, we average all of the data that was assigned to the kth cluster. This is an example of a hard clustering algorithm. And what we mean by that is when we're performing clustering and when the algorithm converges, and outputs the final clustering, every single data point is assigned to only one cluster. And we don't have any sort of uncertainty.
However, you can imagine that on the boundaries of these clusters– for example, a data point that is— could go either way– we might want to encode our uncertainty about the clustering of that data point with some sort of a weight. Whereas the points that are right in the middle of the cluster, we would want to be more confident about. So this leads to the concept of a soft clustering algorithm. Basically a soft clustering algorithm takes the data and it breaks the data into clusters in some sort of an intelligent way. So for example, let's look at the these plots, where we have the raw data according to their

clusters– so for example, all of the red points were generated from the same Gaussian, all of the green points were generated from a second Gaussian, and all of the blue points were generated from a third Gaussian. So the color is encoding which Gaussian that point came from. Now, when we want to cluster these data points, what we see is this data set right here. So this is all we get to see. We don't get to see any sort of cluster assignments or what the cluster centroids are, etc. If we wanted to do a soft clustering of this data set, we might want to cluster it into three groups, where we would get something like this. So the red points–red, green, and blue, again, indicates which of the three clusters the data is assigned to. But these boundary cases are in the context of K-means, this is something that's been considered and it leads to an algorithm called weighted K-means.

So we can take the K-means algorithm and make a simple modification to get a weighted K-means algorithm, which turns K-means from a hard clustering to a soft clustering algorithm. So let's quickly review this algorithm before we move on to the Gaussian mixture model later. So with weighted K-means, what we have is this. We have again data 'x_1' through 'x_n'. And we'll assume that the data is in 'R^d'. Now our goal is to minimize an objective that looks like this. Where we have still, for each data point 'i', a sum over each of the clusters. And we calculate the squared Euclidean distance of a data point to a cluster centroid. But now we divide that by a value beta that's positive. And we multiply it– instead of Multiplying it by an indicator that says that 'x_i' was assigned to cluster 'k', we multiply it by a weight 'Phi_i'. So, the subscript i is indicating the data point we're looking at. Evaluated at 'k', which indicates the cluster we're considering. And these 'Phi's' now can be viewed as probability weights.

So each of these–each value of '$\text{Phi}_i(k)$' is greater than zero and the sum of them is equal to one. And so in this case, we're breaking each data point up into different clusters according to how probable we think that cluster is in generating the data point we're looking at. The coordinate descent algorithm then gets modified in this way. So for a particular iteration– again, we look through each of our 'N' data points, and instead of assigning them to each cluster, we assign responsibilities across the clusters according to this weight phi. So for the i^{th} data point, if we want to know what is the responsibility of the k^{th} cluster for generating the i^{th} data point or the probability if you want to think that way, it's equal to the exponential of minus one over beta times the squared Euclidean distance of 'x_i' to that centroid. We do that for each value of 'k', and then we normalize to get a probability weight. So clearly we've seen this type of a form before with a radial basis function for kernel models.

If 'x_i' is very far away from a 'Mu''k', this value will be very small. And therefore the weight we assigned to that cluster will be very tiny. And on the other end, if 'x_i' is close relatively speaking to the k^{th} centroid–so this will be much smaller compared to the values for all the other clusters. And so this value will be much bigger relative to the other values evaluated at the other clusters. And so when we normalize, we'll get a high probability. So this is how we take proximity to the cluster centroids into account, to break the responsibility for a given data point across the clusters according to how close that data point is to each cluster. Then when we update the cluster centroids, we have a weighted average now, where we take each data point– if we want to look at the k^{th} centroid and update that vector, we look at each data point, we weight it by the probability of that point coming from the k^{th} cluster.

We sum that up, and then we divide by the sum of the probabilities. So you can imagine that if these phi's are our pure indicators– meaning they're vectors of all zeros except for a one in the dimension indicating what cluster 'x_i' came from– that would correspond to a hard clustering algorithm. And we would simply be summing up all the points in the k^{th} cluster and dividing by the total number of points. Now we are softening that by taking fractions of points and splitting them across clusters. So if we look at a plot like this, what soft K-means would do, for example, for a point like this, is say that–we're going to say that the '0.75' fraction of this point is assigned to the green cluster and a fraction '0.25' is assigned to the blue cluster. Whereas the point here would be entirely assigned to the blue cluster, and a point here might be entirely assigned to the red cluster. These border points now are split across clusters to account for the fact that we're less certain about which cluster they come from. And then beta, as with the Gaussian kernel, defines what proximity means. What is a region–what does it mean to be close, and what does it mean to be far away.

So this is now a parameter you would have to set using this algorithm. Let's now turn to a modeling framework called mixture models. With soft K-means, we took each data point and we– instead of assigning it to one and only one cluster, we broke that data point across clusters using a vector of weights. And we loosely use the term probability vector, even though there were no probabilistic assumptions. So now we're going to look at a modeling framework where we do make these sorts of probabilistic assumptions by defining probability distributions. So specifically, if we want to distinguish between probabilistic versus non-probabilistic soft clustering models, we have this weight vector, 'Phi_i'. We interpret it as being like a probability of 'x_i' being assigned to a particular cluster. Now we're going to do a mixture model, where we explicitly define a prior distribution on the cluster assignment 'c_i'. So 'c_i' see is now going to come back into our model and we're going to define a probability distribution on it. And we also get a likelihood function of seeing the observation 'x_i' given the assignment of that observation to a particular cluster encoded in 'c_i'. So we're now using a probabilistic definition to put these constraints on our parameters. Intuitively, something to keep in the back of your mind that we'll return to is the connection between the ideas of mixture modeling and the Bayes classifier that we discussed previously. And this will become more clear as we go on, but we can think of the class prior in the Bayes classifier model as now becoming a cluster prior. So it's the prior on a particular data point coming from a particular cluster except this time now we don't know the label. Previously, the cluster—each cluster was like a class, and we knew the label for each observation and therefore we knew which cluster it came from.

Now we don't know that information. And similarly, with the Bayes classifier we had the class conditional distributions, so given what class it comes from we have a specific class distribution on the data. Now we have cluster conditional distributions– so condition on a data point coming from a particular cluster, we have a specific distribution on that data point for that– coming from that cluster. Before we define the math of the mixture model, let's look at some key features in a little bit more detail. So again, mixture models are a generative model, meaning that we're going to define a probability distribution on the data and then try to learn the parameters of that distribution. A mixture model is a weighted combination of simpler distributions. So, for example, you can look up here where we have data in 'R_2', and so we want to learn a distribution on 'R_2'. And the third dimension corresponds to the density at a given point in 'R_2'. And we construct this complex density of these three humps here by taking a weighted combination of three simpler distributions. So these three distributions are actually three different Gaussian distributions with different means and different covariances.

So we have a -- we have three simple distributions, three simple Multivariate Gaussian distributions. And then we weight each of those and put them together according to a probability of a particular Gaussian to get our final density. So, from a distribution like this, we could get a data set that looks like this where this view is simply looking straight down on this density and we get a cluster here corresponding to these points, a cluster here corresponding to these points, and another cluster here corresponding to these points. So this is the data that we're actually going to be given. And now we want to propose a Gaussian mixture model and then learn the resulting means and covariances and mixing weights on those three different Gaussian. To define a mixture model, we have to define a generative process. So this is the same story with all Bayesian models. If we want to define what it's doing, we have to hypothesize how the data is generated from that model. So here's the generative model for a mixture model. We, again, have data 'x_1' through 'x_n' where each x is in some space. It could actually be a complicated space so it will be abstract at the beginning, but you can also think of the space as just being R^d. We have model parameters for the mixture model which include–for all mixture models, we'll include a 'K' dimensional probability distribution on 'K' different clusters. So we if we assume there are 'K' clusters, we have a 'K' dimensional probability distribution on each of those clusters and also parameters for each cluster.
So 'Theta$_1$' would be the parameters for the first cluster up through 'Theta$_K$', which are the parameters for the Kth cluster. The generative process–so the process for generating data from this model looks like this where, for each observation, we first generate a cluster assignment using a discrete distribution with the parameter Pi, and that is *iid*. So every single data point we're going to assume is generated independently and with the same discrete Pi distribution, meaning that the probability that we assign any particular data point to the kth cluster given pi is equal to 'Pi$_k$'. So this is the prior. Then, given the cluster assignment, we generate the actual observation, 'x_i', from some probability distribution on 'x' where the parameter that we use for that probability distribution is picked out by the indicator, 'c_i'.
So let's make some observations right away about this generative process. First, we see that each data point in apriori, we're assuming, is generated— is first assigned to a cluster. Before we see the data point, we have to first assign the data point to a cluster according to Pi. Then, when we assign it to the cluster, we pick the parameter to generate the data point according to the indexing of the cluster. And so we use the same exact probability distribution. So we're going to assume that 'p' is the same for all clusters so it's, for example, a Multivariate Gaussian or something more complicated than that. The only thing that's differing across clusters is what parameter we use for that cluster. So if 'p' is a Multivariate Gaussian, then Theta would be the mean and the covariance for each of those 'k' Gaussians. Also, if -- noticed that if two points, 'x_i' and 'x_j' are assigned to the same cluster, then their values, 'c_i' and 'c_j', are equal to each other. And in that sense, we're assuming that 'x_i' and 'x_j' both have the same– are generated from the same probability distribution. So they're clustered together by the fact that they share the same parameters to their data generating distribution.
So we can come back now to this picture for the Gaussian setting to see the impact of the weights only. So in these two Gaussians, we have the same Gaussians but we changed the weights. And so what we have is, for example, something that looks like this if we have uniform mixing weights and we get a data set like this. And if we change the weights so, for example, this cluster gets more probability, then we can get something like this, so where we have more data from one cluster than from another as

opposed to even weights, in which case they're equally probable. So the Gaussians define the region and the weights define how dense the data is in each region.

Video 6 (15:45): Gaussian Mixture Models

So now, let's look at the Gaussian mixture model in a little bit more detail. For the Gaussian mixture model, which is one specific case of the mixture modeling clustering framework that we're discussing in this lecture, we have this setup. So again, we let Pi be a 'K' dimensional probability distribution and now we have, for each cluster, for example the kth cluster, we have a mean 'Mu$_k$' and a covariance 'Sigma$_k$' which defines the Gaussian– the Multivariate Gaussian for that specific cluster. When we generate data from a Gaussian mixture model, for the ith observation we assign it to a cluster by generating the cluster assignment *iid*. from a discrete Pi distribution. So this has not changed at all. But in the specific case of the Gaussian mixture model, when we now want to generate the observation 'x_i', we generate it from a Multivariate Gaussian using the mean Picked out by 'c_i' and using the covariance Picked out by 'c_i'. So if we define Mu, and Sigma in boldface to be the set of means and covariances, our goal now is that given some data, we want to learn Pi, Mu, and Sigma. So that's the inference goal. We've defined the hypothesis of how the data got to us by defining the generative model.
Now that defines a joint likelihood on the data given some parameters. And so once we have the data, our goal is now to go back and infer, what are the parameters that could have generated that data? So now we have the inference problem. There are several different ways that we could do inference for the Gaussian mixture model. In this lecture, I'm just going to focus on the maximum likelihood approach. So the objective in this case is to maximize the likelihood over the model parameters Pi, Mu, and Sigma. And we're going to treat 'c_i' as an auxiliary data in the EM framework. So 'c_i', the cluster assignment, now is going to be the additional hidden data that we introduce and then do EM. So we want to integrate out 'c_i'. Remember from the EM algorithm that we discussed last time, what that means is that we want to maximize the likelihood of all of the data 'x_1' through 'x_n' given Pi, Mu, and Sigma. And because we hypothesize that it's *iid*, that simplifies into a product over the individual likelihoods of each observation given the parameters of the Gaussian mixture model. But notice that there's no cluster assignment here– 'c_i' is missing. Where did it go? We integrated it out, so 'c_i' can be written this way– or can be brought back into the model this way where we have the joint likelihood of the ith observation and the cluster assignment, which cluster the ith observation came from. But now we integrate or, in this case, because 'c_i' can only take a discrete and finite number of values, we sum over all of those possible values to integrate them out.
So this term here–right here–is equal to this term. So if we want to know what this is, we simply write out the joint likelihood according to our model and then sum over the possible values of 'c_i' to find what this is. So our goal now is to maximize this joint likelihood of the data given only Pi, Mu, and Sigma where we have summed the parameter 'c_i' out of the model. However, it's difficult to do that. If you write out what this actually is and you take its log and you try to take the derivative with respect to Pi, Mu, and Sigma, you'll quickly find that we're not left with a closed form solution. We can't solve it exactly. And so what we could do is we could derive a gradient method where we take the derivative

and then step in that direction a little bit. But given our discussion of the EM algorithm last time, we're instead going to try to see if we can't use the cluster assignment 'c_i' in an EM sort of framework for doing this maxiMum likelihood. So first, before we do that, the immediate question is why don't we just do maxiMum likelihood over the model that includes 'c_i'?
So why don't we keep 'c_i' as well in the model as something that we want to infer? We want to do a point estimate of 'c_i' in addition to a point estimate of Pi, Mu, and Sigma– because when we do that, when we write out the augmented joint likelihood where we include 'c_i' as a parameter we want to do a point estimate over, we take the log of this and we take derivatives and solve. What we would find is that we can optimize that very easily. We can get closed form updates for every parameter in the model, including 'c_i'. So that is something that can be done. But what we would find by doing that is we would end up with a hard clustering model, meaning that when we wanted to update a particular cluster assignment, 'c_i', maximizing that 'c_i' would lead to it assigning the data point to the most probable cluster. So the maxiMum likelihood solution or the MAP solution, actually, for 'c_i' is to assign the i^{th} data point to the most probable cluster. And so we would have a hard-clustering algorithm. Our goal here is to integrate out the effect of the cluster assignment, which we're going to see when we do that in the EM framework is going to lead to a soft clustering algorithm that we anticipate should be more accurate and should be better than the hard clustering algorithm because we've integrated out one of the unknowns in our model so we have fewer potential local optimal solutions that we could converge to.
So we're not going to derive the EM algorithm from scratch. I'll just give the outline in this lecture. However, we do–to repeat– we do treat these cluster assignment indicators as the latent data–the missing data in our model that we want to integrate over, so we include this thing. Therefore, we're going to use EM to maximize the log of this marginal likelihood where 'c' is gone by using the log of this augmented or extended joint likelihood to help us do that. So let's look at the equation we saw last time and how it is modified for this particular model structure. Last lecture, we saw that the generic EM objective looks like this. We want to do maxiMum likelihood over a model with parameters 'Theta_1'. So we want to maximize the log of the probability of data 'x' given parameter 'Theta_1', but that's difficult. And so we wrote this equality where we had this term here, which we called 'L'.
So this term we called 'L'. And this term here was the Kullback-Liebler divergence between 'q', and the conditional posterior distribution. And we introduce the additional variable to our model, 'Theta_2', and then integrate it out and we do a two-step procedure. So remember the procedures were to set 'q' equal to 'p'. Given that updated 'q' distribution, calculate this expectation. And then maximize this expectation over Theta1. And 'Theta_2' is gone because it was integrated out. Now, if we take this equality and we just write it using the Gaussian mixture model parameters and setup, we get something like this. So the log of the likelihood of all of the data given Pi, Mu, and Sigma -- because the data are assumed to be *iid*, that turns into the log of the product, which turns into the sum of the log of each individual likelihood. So we have a sum over all of the data of the log of the probability of each point given our Gaussian mixture model parameters. And now we introduce the hidden variable. So what we're doing here is you can ignore this first sum the first time you look at it— just ignore this first sum. And now we're going to treat this term here as being like that term. And we're going to introduce c as this 'Theta_2'. So 'Theta_1' corresponds to these parameters. 'Theta_2' corresponds to 'c_i'. And because 'c_i' is a discrete variable, the integral is going to turn into a sum.

So when you have continuous variables, you integrate. When you have discrete ones, you sum. That's the difference. And so this integral over all of the values that 'Theta$_2$' can take turns into a sum over all of the values that 'c_i' can take. But notice that, again, we have the data 'x'– we have 'x_i' as mapping to 'x' here. For 'Theta$_1$', we're mapping our mixture model parameters to 'Theta$_1$'. For 'Theta$_2$', we're mapping 'c_i' to 'Theta$_2$' here. And we have 'c_i'. And then, for the integral over all of the values 'c_i'...'Theta$_2$' can take, we're mapping that to the sum over all of the values that 'c_i' can take– and similarly, with the KL divergence. This term maps to this term. And now because each of these individual terms are equal to each other, when we sum over all of the data on the left-hand side and sum over all of the data on the right-hand side, again, nothing changes.
We have the left-hand side is equal to the right-hand side. Let's now look at the EM algorithm for this model. Remember, with the EM algorithm our goal is to first set the KL divergence equal to zero. So we have a sum over 'n' different KL divergences here. Each KL divergence is calculated for– we have one KL divergence for one data point and the ith KL divergence is equal to this term right here. And so to set the entire KL divergence to zero, we can set each individual KL divergence to zero. So we want to set 'q' of 'c_i' equals 'k', which is our probability of 'c_i' equaling 'k' to be equal to the conditional posterior probability. So this should be also evaluating 'c_i' is equal to 'k'. So that's the first of part of the E-step. We set this distribution 'q' 'c_i' equal 'k' to be equal to the conditional posterior distribution of the ith data point being assigned to the kth cluster given the value of the ith data point and given the parameters for the Gaussian mixture model at the current iteration. To do this, we use Bayes' rule. So Bayes' rule says that the conditional posterior is proportional to the prior of... assigning the ith data point to cluster 'k' times the likelihood of the observation– of the ith observation given that it was assigned to cluster 'k'. So we see that this term, the prior probability of assigning a data point to cluster 'k', is just equal to 'Pi''k'. And the likelihood of the ith observation coming from cluster 'k' is equal to a Gaussian where we evaluate the likelihood of 'x_i' given the mean and covariance of the kth cluster. And then, to turn that into a probability distribution, we simply normalize the right-hand side. So this is a proportionality. To turn this into an equality, we normalize this thing by dividing over the sum of all possible values that 'c_i' can take, meaning we sum this numerator over all values 'k'. So we take this, we divide it by the sum over all possible values for the cluster assignment. And we get–we define this to be 'Phi$_i$' of 'k'. So this is serving identically the same purpose as with the weighted K-means algorithm. The only difference is that now we have a firm probabilistic interpretation of what we're doing. So that's just the first step of the EM algorithm. Remember that the first step of the E-step of the EM algorithm.
The second part of the E-step, after we update each of these 'q' distributions, is to calculate this expectation. So, in that case, we want to calculate this thing where we are now summing over the possible values for each 'c_i'. And so that turns into this. In the previous slide, we had a sum where we had this term here. Now, we take this term and we replace it with what we defined it. So this is just our shorthand. But now, we have the sum over each observation where we evaluate the log of the likelihood of the data coming from each cluster– so that's this term, the log of the likelihood of the ith data point coming from the 'kth' cluster–plus the log of the prior probability of it coming from the kth cluster. So that's equivalent to the log of the joint likelihood of the ith observation coming from the kth cluster and it coming from any point coming from the kth cluster.

We take that log and we Multiply it by the weight that we assign to the kth cluster and sum it up. So this term is simply equal to this term where, again, we are going ignore this part because it's simply a constant. It's not going to change anything. So we can write this as the log of 'p' minus the log of 'q'. We can separate these two terms and then this term is not going to involve the parameters that we're interested in updating, so we're going to ignore it and simply look at the log of this term here. And again, we replace 'q' with what we've defined 'q' to be. Okay, so we can actually write this out. We can expand what this is. We can write out the Gaussian form for it and then get this function that we want to maximize– remember, we called this 'L' last time, not 'q'. And so then we maximize 'L' with respect to Pi, 'Mu$_k$', and 'Sigma$_k$' for each 'k'.

Video 7 (06:23): EM Algorithm

Okay, so let's look a little bit closer at the M-step and ask how has EM made our lives easier here. We can see this by first looking at the original objective function which is the log of the likelihood of the data given Pi, Mu, and Sigma where we integrate out 'c'. So, we get something that looks like this. A sum over each data point of the log of the probability of 'x' given Mu, Pi, and Sigma. And, we could also write that as a sum over 'c' where we integrate out 'c' first. So, we're integrating out 'c', and we get this. So, this term is equal to the sum from 'i' equals one to 'n' the log of 'p' of 'x$_i$' given Pi, Mu, and Sigma. This is the original thing that we want to maximize. Remember, on the left- hand side, right! And so, we have a log sum term here, and this log sum, this sum appearing inside of the log is what makes it difficult. If you took the derivative of this with respect to Mu or Sigma or Pi, you would see that this sum inside is what's the thing that's making everything complicated for us. So, with EM, what we have, again, this is one, is now we have an M-step after we do the E-step. We have an M-step that looks like this. We have a sum over all the data points, and then a sum over each of the clusters of a log term. So, this term is a log of something.
So, now, we have a sum log. Before, we had a log sum. EM turned it into a sum of a log. If we, now, take the derivative of this and try to maximize it with respect to Pi, Mu, and Sigma, we'll find that it's very easy to do that. And so, that's what we do. We actually take the derivative of this thing with respect to Mu, the means, the covariances, and also Pi conditioned with the constraint that Pi is a probability distribution. And, we maximize this function with respect to those three unknown parameters, and we'll find that we get a closed form solution that looks like this. So, here's a summary of an EM algorithm for doing maxiMum likelihood for the Gaussian mixture model. This is the solution that we end up with. So again, we're given data, 'x$_1$' through 'x$_n$' where each point is an R^d. Our goal, now, is to maximize the log of the marginal likelihood of each data point over Pi, Mu, and Sigma where there is no cluster assignment at all to be found because it's integrated out. And now, we're going to iterate between the following two steps, the 'e' and the M-step. First, we perform soft clustering of the data by assigning it, by taking each data point and calculating the conditional posterior probability of that data point coming from each of the 'k' different clusters. And, we can do that this way where we calculate the prior probability of the kth cluster times the likelihood of the observation given it comes from the kth cluster for the ith observation. And then, divide by the sum of that over each cluster.

That way, we get a probability distribution of the probability of the ith data point coming from the kth cluster. Now, here, for the M-Step, this is the solution that I haven't worked out. But, this is what it ends up being. For each cluster, 'k' equals one cap 'K'. First, let's just for simplicity, for notation reasons, let's define 'n_k' to be the sum over each data point of the probability of that data point coming from the kth cluster. So, this sum is like the expected number of points that we're going to see coming from the kth cluster in the, according to the current iteration. Then, we can update the three parameters of a Gaussian mixture model in this way. We, first, update the probability, the prior probability of the kth cluster by simply taking 'n_k' and dividing by the number of data points. So, this is the empirical distribution on the clusters according to our assignments, our soft assignments. So, this makes intuitive sense. For the mean, we, then, take each data point. We multiply it by the probability of that data point coming form the kth cluster. We sum it up and divide by the total, expected total number of points coming from that cluster. So, notice that this actually is identical to weighted K-means.
This is exactly the same as weighted K-means. The only difference is the way that we calculated this Phi. So, this Phi, the way we calculated Phi with the Gaussian mixture model, used this equation. If you look at the way we calculated Pi for weighted 'k' means, it used a slightly different equation, but then, once we have the soft clustering of the ith observation of each observation, we can calculate the mean exactly in the same way. But, then, in addition to that, with the Gaussian mixture model, we get a covariance matrix for the kth cluster which is also like the weighted empirical covariance of the data coming from the kth cluster. So, for 'Mu_k' here, this is the mean of the kth cluster. We use the updated Mu here. So, we have to first update Mu, and then take Mu and plug it into this equation here. We calculate this outer product. We multiply it by the probability that the ith point came from the kth cluster to begin with. Sum that up over each data point, and then divide by the expected total number of points. So, this is a soft, like a soft covariance matrix.

Video 8 (08:56): M-Step Close Up

In the K-means lecture, we looked at this exact same example of iterating through an example run of clustering with the K-means algorithm, clustering data into two clusters. Now let's look at the same example of clustering this data into two clusters except we're now going to do a Gaussian mixture model. So, in this case, we define an initial Gaussian– two Gaussians. So for clarity we'll put them way out here. This would be the first Gaussian. This is the second Gaussian. And the circles indicate the covariance matrix. So this circle indicates that it's a spherical covariance matrix for each of these Gaussians. And also, they're equally probable. The weight is not being shown here on the probability of each of these clusters. Okay, so this is our initialization. The E-step then, we assign –we take each data point, and we calculate the posterior probability of that point coming from each of the clusters. So for example, this point here has a certain probability of coming from this cluster and a certain probability of coming from this cluster and now we're going to color code it based on the weight of that probability. So, this point here is almost entirely red to indicate that with almost probability one it's coming from this cluster.

This point here is almost entirely blue to indicate that it's almost entirely coming from this cluster. And then as we move from this side to this side, we get shades of averages between blue and red. So right here, in the middle, it's '50-50' on either of these two clusters. And so that's how we are going to visualize the clustering in the E-step. Remember, for K-means what we had was something that looked like this. We had a line basically that went like that and everything on one side was entirely red. Everything on the other side was entirely blue. Now we have the soft clustering. Then the M-step of the first iteration is to update the Gaussians. And so we take the red –we take all the data, we calculate the empirical– the empirical weighted mean and the weighted covariance for each of the clusters and we get something that looks like this. So this is the first cluster. This is the second cluster. And this is their updated means and covariances. And so we step through this and now we'll skip the E and the M-steps and just go through an entire iteration. After one iteration, you know– so we start here after one iteration, we get this.

Two iterations, we get something like this. Up to five iterations, we're here. And then by 20 iterations, we've converged and we have this clustering. So it turns out that, for this particular data set, we're very confident on our clustering. All of these points are basically– entirely this cluster and all of these points are entirely this cluster. Maybe there's some difference in shading that's hard to see. We get two clusters. We get a Gaussian with this mean and this covariance and a second Gaussian with this mean and this covariance. And we also get a probability of each of the Gaussians. So this Gaussian has a certain probability based on the empirical proportion of data coming from that cluster and then this cluster has the remaining probability. I want to quickly return to the Bayes classifier to again make some comparisons between these two methods because, intuitively they feel very similar but there are some very significant differences. So we have a GMM, a Gaussian mixture model with 'K' clusters. It feels a lot like the K-class Bayes classifier that we learned where each class–we had a class-specific Multivariate Gaussian density. Remember, with the K-class Bayes classifier, what we did was we took a data point 'x_i' and we declared the label of that data point to be the arg max over 'k' of a class prior, so the prior probability of a data point coming from class 'K' times the likelihood of that particular data point coming from the class 'K' specific Multivariate Gaussian. So we did this arg max and assigned 'x_i', to be–to have that label.

So, in that case, we also learned Pi, Mu, and Sigma, and we had similar looking updates for those parameters except, if you remember, we didn't have an algorithm in that case. We just simply gave an equation to say this is the maximum likelihood for Pi, for Mu, and for Sigma. Now, in this case, suddenly we have an algorithm. So what has changed? The key thing that has changed is that, with the Bayes classifier, we already had the– if you will, the cluster assignments. So, the clusters were like the classes and we had already assigned each of the data points to their specific classes because we had labels for them in our data set. And so we didn't need to ask which cluster or which class any data point came from. We simply assigned it to its specific class.

Now, with clustering, with this Gaussian mixture model which has a very similar form, we suddenly don't know for any individual data point which cluster it came from. And so what that tells you is that conditioned on the cluster assignment we can do maximum likelihood exactly. But when we don't have the cluster assignment, when we have to iterate between updating the cluster assignment and then updating the model parameters, we have to run an iterative algorithm because we can't optimize both

simultaneously. So, in a sense, this 'Phi$_i$(k)' is almost like encoding a probability of what label the ith data point has. If you were to view a cluster as like a label for that data point, we're inferring both the labels and the class specific densities simultaneously. Although it's not typical to think of this sort of unsupervised mixture modeling as a classification problem, we don't ever usually discuss the clusters as being classes but intuitively, the math is very similar. Now, a natural question is to ask, how many clusters are there in my data set? Doing a maximum likelihood inference algorithm for the Gaussian mixture model is very similar in this case to the K-means algorithm in that the number of clusters that you give it is going to equal the number of clusters that it learns.
So, for example, if we have a data set that looks like this and we do maximum likelihood EM for the Gaussian mixture model, and we assign–we define there to be three clusters, we'll get something that looks like this where we have these three clusters, where we have these three Gaussians, and we have a distribution on each Gaussian. And we assign each data point softly by its conditional posterior distribution to one of the three Gaussians. If we now assigned there to be, say, '20' Gaussians or many more Gaussians and we do maximum likelihood EM, we'll get something like this where we just simply partition the data into these many different Gaussians. And you can also think of it as we overfit our data. So there are a set of Bayesian techniques to handling this that we won't discuss in this course, but they're all based on using the Dirichlet distribution, or mostly based on using the Dirichlet description where we take the cluster prior Pi and put a probability distribution on it and then infer the posterior where the probability is going to encourage there to be as few clusters as is necessary to model the data. So these are called Dirichlet priors and they're often used to fix this issue but that's just to give you a pointer to how these methods are developed to handle these sorts of situations–these sorts of modeling issues like overfitting.

Video 9 (04:12): Gaussian Mixture Model: Example Run

Finally, I want to quickly discuss how this can be generalized to mixture models using different distributions besides Multivariate Gaussians. We, actually, already discussed everything necessary to give an outline of the algorithm for doing maximum likelihood EM for any generic mixture model and it looks like this. So, now we have data 'x$_1$' through 'x$_n$', where we assume each data is in some space, and this space can be anything. It can be quite complicated if we want. It doesn't have to be R^d. And our goal now is to maximize the log of the likelihood of the data. And because we, again, make an *iid* assumption it's the sum of the log of the likelihoods of each observation given the mixing weights Pi. So, this does not change at all. And also given some parameter set Theta, where we're going to be generic here.
This cluster specific likelihood of 'x' given the kth cluster is going to be generic. With the Gaussian mixture model, this was a Multivariate Gaussian and 'Theta$_k$' was the kth mean and the kth covariance for that Gaussian. Now this is any probability distribution, and 'Theta$_k$' is a set of parameters for that distribution. The EM algorithm in this case looks like the following. For each iteration we have the E-step. This is identical to before where we say, "For each observation 'i', we calculate the conditional posterior probability of the ith observation coming from the kth cluster that's equal to the prior probability of a point coming from the kth cluster times the likelihood of the ith data point coming from

the kth cluster, where we use the parameter for the kth cluster to evaluate this likelihood." This is now different from before. Before this was a Multivariate Gaussian. Now it's whatever distribution that we've defined on 'x_i' according to the problem that we're looking at. So, we calculate this, and then we normalize to get a distribution, and that's the conditional posterior probability of the ith point coming from the kth cluster.

The M-step then is similar to before in the sense that, for the kth cluster, we now sum the probabilities of all of the data coming from that cluster. So, we get the expected total number of points. We're going to see from that cluster, and then we update the prior probability of any point coming from the kth cluster as taking that sum 'n_k' and dividing by the total number of points. So, this is no different from before. It's the same exact thing. This now is...has to remain generic. So, when we want to update the parameter to the distribution for the kth cluster, what we have now is this problem where we want to find the arg max over all values Theta of the sum over each data point of the probability of that point coming from the kth cluster times the log of the likelihood of the ith point given Theta.

So, for example, before this was a Gaussian where we wanted to maximize this thing over the mean and the covariance, and we took the derivative and, you know, we found out we could do that in closed form, and I gave the equations on the previous algorithm. Now, depending on what this distribution is, we might need to run an algorithm to update... to maximize this, or maybe we can take the derivatives and set to 0 and solve, but we'll have a different equation given the model that we're considering, given the densities, the likelihoods that we're using for the model we're considering.