Week 12
Video Transcripts

Video 1 (9:02): Association Analysis and Rules

In the last nine lectures, we've been discussing unsupervised learning. We first discussed, clustering models, followed by matrix factorization approaches and finally sequential models, and in all three of these types of unsupervised learning frameworks, we were looking to uncover interesting patterns underlying the data, without necessarily, having a prediction task in mind. In this lecture, we're also going to discuss unsupervised learning, but a fundamentally different approach, from what we've been discussing previously, today we'll discuss something called Association Analysis. So, at a high-level, association analysis is going to be the problem of finding highly probable subsets of data, and this is something we'll make more concrete throughout this lecture. To set up the problem, we can imagine that many businesses have massive amounts of customer purchasing data. So, for example, Amazon has your order history, a grocery store would know objects purchased in each transaction if you give them your rewards card, for example, they'll know the purchase is made by whoever's identity is on that card. Other retailers also have similar data on purchases at their own stores. So, using these types of data, we might want to find subgroups of products that tend to co-occur, in either purchasing or viewing behavior. So, for example, retailers might want to cross-promote products through different deals by knowing that these two products are very highly correlated in the way, that they're purchased, so we might want to have deals on both products at the same time. Grocery stores might want to strategically place objects. So, this is the classical example where a grocery store would want to strategically place objects next to other objects because it would encourage a consumer to purchase them both at the same time. Online retailers can use this information to recommend content. We've already discussed one approach for recommending movies; in that case, we had actual user feedback on their beliefs, their thoughts about the movie through a rating. In this case, we might want to recommend content, based only on viewing behavior. So, we don't get any feedback, but we know that users tend to view different objects together in the same viewing session, and it's also something that can be generalized more fully. So, let's look at the basics of association analysis, through something called market basket analysis. So, this is the canonical way of thinking about finding associations among data. So, for example, let's think of...imagine that we have five market baskets. What this literally means is that we're assuming that somebody has a basket at a market or a grocery store, and they have different objects in their basket at checkout time.

And we're going to look at five of these baskets, meaning we're going to look at five different people at checkout time, in this very simple example. So, what...what this would say, is that, the first person, or the first basket that we have, contains bread and milk in it, so it contains those two objects and nothing else. The second contains bread, diapers, and beer and eggs. The third basket contains milk, diapers, beer and cola, etc. So, imagine now that we have a lot of this data. We have millions of these

transactions and they're across thousands of products; we now might want to use this type of data to analyse patterns of co-occurrence. So, which products tend to co-occur together? Which objects tend to appear in baskets at what percentage or what rate, and also to use these sorts of patterns to define association rules? And it's just another word for conditional probability. So, for example, and this is the classical example, that apparently was found in one of the first uses of this technique. We might find that people who purchase diapers also purchase beer. So, we want to learn an association rule, and this one's kind of a joke, where, given that they've got one object in their basket, we're going to predict that it's more likely to have a second object in their basket.

So, let's make this problem a little bit more mathematically abstract. So, what we're trying to do is to perform association analysis and learn association rules. And so, first let's imagine that we have 'p' different objects, for example, there could be 'p' different items for sale in a grocery store, and each of those items has a unique index from one to capital 'P', saying which object it is, that's for sale. So, then we assume that we have a collection of subsets of these items, for example, 'X_n' for the n^{th} subset would be a subset of indexes from one to 'p', where we can think of 'X_n' as giving the index of the things purchased by the n^{th} customer. And we're going to let 'n' range from one to capital 'N'. So, 'X' subscript 'n' is like the market basket for the n^{th} customer, that says–that gives the index of which items are being purchased by the n^{th} customer. It doesn't give the number of items purchased, for example, if milk is being purchased, it's not going to distinguish between one or two bottles of milk, it's just going to give an index that's saying milk is being purchased.

So, now there are two objects of interest that we want to discover. The first is to perform an association analysis. This is simply to try to find subsets of items that have a high probability of co-occurring together. So, to this end, let's define 'K' to be a subset of index values between one to 'p', so this is a subset of items. And then 'P' of 'K' is now going to be the fraction of market baskets, that contain those items indexed by 'K'. So, if 'K' is an index of a subset of indexes, then the probability of that subset, 'K' is simply found by counting how many of these baskets contain those items in it, indexed by 'K'. So, we want to count the number of 'n''s, such that 'K' is a subset of 'X_n'. The N^{th} market basket could contain more items, we're only interested in whether the items indexed in the set 'K' are contained in the n^{th} basket. So, we count the total number of baskets where this subset appears in, and then we divide by the total number of objects or subsets to begin with. So, we want to find the subsets where 'K'– where this probability is large. For example, we might define 'K' to be peanut butter, jelly and bread, and then we'll go and count how many of these baskets at purchase time contain those three objects together, and then divide by the total number of baskets to get the fraction of customers, that purchase those three objects together. That would be 'P' of 'K'. So, our goal is to find these subsets, where this is a large number. A second goal is to learn association rules. So, this is the problem of finding objects that are highly correlated, where knowing that a certain subset of objects appears in a set 'X_n' makes it more likely that another object is going to appear in that set as well.

So, to this end, let's let 'A'and 'B' be two disjoint subsets of the integers one to 'p'. Then 'A' right arrow 'B' can be thought of as meaning that purchasing 'A' increases the likelihood of purchasing 'B' as well. So, we want to learn these rules. For example, if we define 'A' to be this set, peanut butter and jelly, and 'B' to be the set bread, we want to learn that purchasing these two items increases the probability of purchasing this item.

Video 2 (10:06): Processing the Basket

So, in order to do this, we need to learn how to represent each of these baskets. Here is the list of baskets as presented a few slides ago. Instead of representing it this way, we think of it as an indicator of an item being present in a particular person's basket. For example, if we look at the first person, then we would have a one for bread, a one for milk and a zero for the other objects. If we assume, that there are only these four objects that can be purchased, this person purchases bread and milk and so there's a one under those two items and zero under everything else. And that corresponds to the bread and milk. Similarly, if we look at the fifth person, this person bought bread, milk, diapers, and cola, but did not buy beer and eggs. And so, there's a one for the items that appear in that person's basket and a zero for the items that don't appear in that person's basket. So, this is how we think of representing the data that we want to now process. So, our goal now is to find the subsets of these six items in this toy example that tend to co-occur together with high probability, meaning in this case the high probability is, simply saying a probability above some threshold that we're going to set. So for example, we might ask do the objects bread and milk occur together relatively frequently in a person's basket. So, what do we do? We simply go to each of the five baskets and we count the number of times that bread and milk co-occurred in both. So, we have one time here, a second time, and a third time. So, three out of the five baskets contain both bread and milk. That's '60' percent. And so therefore, we would say, okay, these two objects co-occur with high probability from this very simple data set. So, notice that in this case, a person's basket can contain more objects than just those two. For example, the fifth person also has diapers and cola but we're only checking for the subset bread and milk and so we disregard everything else. So, it seems like a straightforward and trivial problem and when the data is small, it is, we simply have to brute-force count every subset. However, in real life, the data sets that we're looking for don't have five different objects and six items.

We might have 100 million users or customers and there might be on the order of thousands of objects that they could purchase or view. So, there's a very large number of users and objects making up this chart. So, we can do some quick combinatorics to figure out why brute-force search is not going to be possible in this case. For example, we can ask how many different subsets of items are there that we could possibly check. So, this is saying, how many of these subsets 'K', of the integers one to 'p' are there that we have to check as a potentially interesting subset of objects? To answer this, we can represent each subset 'K' as a binary vector of length 'p', where a one indicates that the object is in that subset and a zero indicates that an object is not in that subset. And so, the total number of these binary vectors, meaning the total number of potential subsets 'k' that we would have to check is two to the 'p'. So, this is a very large number as soon as 'p' gets even moderately big. So, it's not possible to check every single possible subset, by brute-force counting. Ok, so then we might say that, well clearly nobody will have a basket with every item in it, and if 'p' is very large we could upper bound the number of items that we want to check to 'K'. So, we only care about checking subsets of 'k' objects picked from 'p', a superset of 'p' total objects.So, in this case, we can still do the combinatorial analysis to say, to ask how many different subsets of size 'k' can we construct from a superset of size 'p'? And the answer to this is, 'p'

chose 'k', which is 'p' factorial divided by 'k' factorial times 'p' minus 'k' factorial. So, this is also a huge number. For example, if 'p' is 10,000, so if there are 10,000 total objects, and 'k' is only five, meaning that, for example if a store has 10,000 total objects for sale, but we're only going to check subsets of five objects, as being a potential interest, then 'p' choose 'k' is still '10' to the '18'.
So, that's '10' to the '18' different subsets that we would have to check, by a brute force search, which again is too large to do in a reasonable amount of time. So, the takeaway is that intuitively, what we're trying to do is extremely simple, we have 'N' of these different baskets, for example, this there we could have several millions of rows, in this table, and each row is going to say that this customer purchased a certain subset of items and all we're looking to do is find subsets that tend to occur a lot, that tend to be purchased together very frequently. The problem is that we can't do a brute force search over all of the subsets to actually count these, the number of times that these subsets co-occur with each other. And so, we need some sort of an algorithm in order to help us do an efficient counting. And so, we can't do a brute-force search, so we need some sort of a clever approach to handling this counting problem. So, before we discuss the algorithm for finding all of the subsets of potential interest, we first need to ask what is it that we want to count. So again, let's let, 'K' be a subset of 'p' objects. So 'K' is a subset of the 'p' integers. And we're going to let 'A' and 'B' be two subsets of 'K', such that the union of those two subsets is 'K' and 'A' intersected with 'B' is a null set. So, in other words, we let 'K' be a subset of the superset one to 'p'. And then, we define 'A' and 'B' to be a partition of 'K', into two disjoint sets. There are three probabilities that we're interested in computing. The first is simply the probability of the set 'K', which can also be written as the join probability of 'A' and 'P'. This is called the prevalence or the support of the items in set 'K'. So, we want to find all subsets 'K', such that, that combination in that subset co-occurs very often. So, it co-occurs with a high frequency in the data that we have. Another thing we would want to compute is, the conditional probability of the set 'B' given the set 'A'. So, if you tell me that the set 'A' is in my basket, what is the probability that the items in the set 'B' are also in my basket? So, this is the conditional probability. This can be written as the joint probability of 'A' and 'B' divided by, the marginal probability of 'A', and since 'A' and 'B' is simply 'K' that's the probability of 'K' divided by the probability of 'A'. So, this is called the confidence. This is the confidence, that 'B' will appear in the basket given, that 'A' is in the basket. We use this confidence to define rules. So, we want to find high probability sets 'A' and 'B' such that, if you tell me 'A' is in the basket, then with high probability 'B' is also in the basket. We want to learn from the data what are these sets. We want to construct all of these potential sets of interest, with high probability.
Finally, we define this function 'L', of 'A' and 'B'. This is the joint probability of 'A' and 'B' divided by, the marginal probability of 'A' times the marginal probability of 'B'. So, we can simplify this first fraction 'A' and 'B', divided by 'A' by recognizing that this fraction is simply this marginal distribution. So, in other words we want the marginal-conditional probability of 'B' given 'A', divided by the marginal probability of 'B'. So, this is called the lift of the rule 'A' implies 'B'. This is a measure of how much more confident we are in 'B' given that, we see 'A'. So, the set 'B' has a certain prevalence in the data set by itself. And now, we want to know, how much more confident are we in the set 'B' being contained in the basket given that you tell me 'A' is in the basket compared to simply the prevalence of 'B' overall.

Video 3 (6:31): Processing the Basket: Example

So, let's look at an example, a simple intuitive example where we define 'K' to be the set of three objects peanut butter, jelly, and bread. So, 'K' has three items in it, peanut butter, jelly, and bread. Then we partition this, so that, 'A' is peanut butter and jelly and 'B' is bread. So, 'A' and 'B' do not share any items and the union of 'A' and 'B' is equal to 'K', so this is the partition of 'K', 'A', and 'B'. So, if you tell me that a prevalence of '0.03' that the set 'K' has a prevalence of '0.03', that means, that if I look at all of the data that I have, all of the purchases that I have, the three items, peanut butter, jelly, and bread, appeared together in three percent of those purchases. If you tell me that the confidence of 'B' given 'A' is '0.82', what that means is that, when peanut butter and jelly were purchased then, '82' percent of the time, bread was also purchased. So, if I simply restrict myself to looking at the baskets that have peanut butter and jelly in them, and then I throw away every other basket in my data set, then in '82' percent of those baskets, that I'm restricting myself to, bread will also be contained in that basket. And then finally, if you tell me that the set 'B' and 'A' have a lift of one '0.95', so if you tell me that, 'L', 'A', and 'B' equals '1.95' then, what that means is, that it's '1.95' times more probable, that bread will be purchased given that peanut butter and jelly were purchased, compared to the probability of bread being purchased at all.

So again, this is 'P' of 'B' given 'A' divided by, 'P' of 'B'. And so, what I am going to do is, I'm going to look at all 'N' data points that I have, I'm going to count, how many times bread was purchased, in those 'N' objects. So, the fraction of the 'N' baskets, that have bread purchased that's 'P' of 'B'. And then, the fraction of the baskets that have bread purchased, restricted to the baskets that all ready have peanut butter and jelly purchased, that's 'P' of 'B' given 'A'. This fraction says, what's the lift or the bump in probability that I get. If you tell me 'A', how much more confident am I in 'B' compared to whether, to how confident I was in 'B', without any knowledge. So, with those three probabilities of interest in mind that leads us to the apriori algorithm. So, this is the name of the algorithm that allows us to quickly find, very quickly find all of the subsets 'K', that have empirical probability greater than a predefined threshold 't'. So remember, there are two to the 'P', different subsets, that I can construct in 'K'. And, I want to find all of those subsets that have an empirical probability greater than 't', which I'm going to set that number 't'. The apriori algorithm is going to give me a way to find every single one of those subsets, without actually having to search all two to the 'P' subsets.

So, if I find a subset 'K' that satisfies this probability, what that means is, that the items in subset 'K' will appear in at least 'N' times 't' of the 'N' baskets, that I have. And, we believe that a very small fraction of subsets should satisfy that so we're going to set 't' such that, that's the case. So, what the apriori algorithm is going to do is, it's going to restrict the number of subsets that we actually need to check by using simple rules of logic. So, it's going to start by constructing all subsets of size one. So, it's going to first check every single subset of size one. It's then going to have some subsets surviving, some not surviving, and then based on that it's going to move on to subsets of size two and so on. And, only the subsets—when we're at subsets of size 'K', only the subsets of size 'K' minus one that survive this process are going to factor in, when constructing subsets of size 'K' to check. And so, what's important

about the algorithm we're going to discuss is, that it's going to find every single subset, such that the empirical probability of that subset is greater than 't'. But it's going to again use certain properties about probabilities to eliminate subsets without even bothering to check them. So, this structure can be put into a lattice representation and we'll return to this figure later. So, we start with subsets, the null subsets, so this is the subset with nothing in it and then we can move on to subsets of size one. So, if we have five objects, if 'P' is equal to five, then we label them 'a', 'b', 'c', 'd', and 'e', then we have all of the five subsets of size one to check. And then we move on to all subsets of size two, and so here we have, all subsets of size two.
We can then move on to all subsets of size three, all subsets of size four, and finally all subsets of size five and because there are only five different objects in this example, there's only one subset there. And so, we can represent all of the subsets that we want to check as this lattice structure.

Video 4 (13:47): Frequency Dependence

So, we're going to use two very simple properties, about the subsets in order to develop an algorithm for efficiently counting. So, for example, if the set 'K' is not big enough, meaning, if we have a certain set 'K' and that set 'K' does not appear in the fraction 't' of the baskets that we're looking at, so the probability of 'K' is less than 't', then we immediately know, that any subset 'K' prime that we can construct by taking 'K' and adding another object to it–another element, so, we take 'K' union with some other set 'A' is also not goingto be big enough. So, in other words, if 'P' of 'K' is less than 't', then we know that the 'P' of 'K' prime, has to also be less than 't'. So, for example, if 'K' is equal to 'a' and 'b', and if these items appear together in 'x' baskets, then the set of items 'a', 'b', and 'c' have to appear in less than or equal to 'x' baskets. If we have 'x' baskets, that contain both 'a' and 'b', we can't have more than 'x' baskets that contain 'a', 'b', and 'c '. This is simple logic and that follows, because the set 'K' is a subset of the set 'K' prime. Mathematically, what that means is that we can write the probability of the set 'K' prime, as the joint probability of the set 'K' and the set 'A' where, these two sets are disjoint.
We can write this joint probability as the conditional probability of 'A', given 'K', times the marginal probability of 'K', because a probability has to be less than one. We know that the product of these two numbers must be less than 'P' of 'K'; we take 'P' of 'K' and multiply it by a number less than one, that has to reduce the size of 'P' of 'K', so the product of these two numbers has to be less than or equal to 'P' of 'K', If this is one, then it would be equal, and since we know that 'P' of 'K' is less than 't' by assumption, we therefore know that 'P' of 'K' prime also has to be less than 't'. So this means that any set 'K' that we know occurs in fewer than the fraction 't' of the baskets, we can avoid having to construct any larger subset 'K' prime that contains 'K', as a subset of that larger set. So, we can immediately eliminate a lot of checking. By the converse, if 'P' of 'K' is greater than 't' and 'A' is a subset of 'K', then we automatically know that 'P' of 'A' has to be greater than 'P' of 'K'–has to be greater than 't'. So if 'K' satisfies our threshold, then all subsets of 'K' have to also satisfy our threshold, that's the kind of the inverse of the first logical rule. So, the first property says this– if 'a' and 'b' don't co-occur enough times to pass our threshold, then we can immediately eliminate all of these other subsets that contain 'a' and 'b' in them.

So all of these sets that contain 'a' and 'b' as part of that set, we immediately know that any of these subsets also will not occur enough times, to pass our threshold.
So, this is the first property. If we eliminate this set, we immediately can eliminate all of these subsets for checking. The inverse property of that, says that if, for example, the set 'c', 'd', 'e' co-occurs enough, so if 'c'–if object 'c', 'd', and 'e' occur together enough times to be above our threshold, then all subsets of 'c', 'd', 'e' also must co-occur together, to satisfy our thresholds. So if 'c', 'd', 'e' appear in enough baskets together, then 'c' and 'd' also must appear in enough baskets together, because the probability of all of these subsets has to be greater than the probability of this subset. So this leads us to the most basic version of the apriori algorithm. So, this can be improved in different ways to make it more efficient, but already we've made quite a bit of progress by reducing the two to the 'P'– different sets we have to check to something that can actually be done in finite and reasonable amount of time. So, first we're going to set some threshold, 'N' times 't' where, 't' is a number between zero and one, but it's relatively small enough that we don't expect there to be many subsets that co-occur together more than 'N' times 't' times, in our data set. The apriori algorithm then has one iteration for each size of subsets that we might want to check. So, in the first kind of step, we would want to check all subsets of size one. So, in this sense, we're going to take every object, and we're going to keep all of those objects that appear in more than 'N' times 't' baskets. So, those are all the sets of size one that occur frequently enough. Those are the objects that are bought frequently enough that they have more than 'N' times 't' purchases among the 'N' total purchases. Then at step two, we want to check all of the subsets of size two.
In this sense, we're going to take all pairs of objects that have survived step one. And so, this is where the properties that we discussed come into play. If something didn't pass step one, then taking a set of size two containing one of those objects is not going to occur enough. So if the third object, doesn't appear in more than 'N' times 't' baskets, then the third object plus the fifth object isn't going to appear in more than 'N' times 't' objects. We immediately know that; so, we don't even have to check it. So, we take all of the sets of size one, that pass this threshold and construct subsets of size two, among those objects and then check those, and see which subsets of size two appear in more than 'N' times 't' baskets. And keep the ones, that appear more than 'N' times 't' times, and then, throw away all subsets of size two, that don't occur enough times, and then we keep doing this. For example, once we get to subsets of size 'K', what we do is, we take all subsets of size 'K' minus one, that passed the threshold, that appear in more than 'N' times 't' baskets, and then, we add a new object, by incrementing each of those 'K' sets of size 'K' minus one, with an object that survives step one, but not all really in the set to make a set of size 'K'. We then construct that potential set of interest of size 'K'. We check in our data set whether or not, that set of size 'K' appears more than 'N' times 't' times. If it does, we keep it, if it doesn't, we throw it away. So, it's still a brute-force search, but it's reducing the amount of searching that we need to do by not fruitlessly searching.
We're only searching subsets that could potentially be of interest– things that we apriori know are not going to be of interest, that they won't satisfy this threshold, we don't even bother checking. And so, it should be clear, intuitively speaking, that as 'K' increases, we can hope that the number of sets, that pass on to the next size 'K' plus one is going to be decreasing. So, as 'K' is increasing, we're having fewer and fewer sets, that satisfy this threshold, and at a certain value of 'K', no sets are going to survive. So,

at a certain value of 'K', much less than 'P', we're not going to be able to find any subset of that size that occurs in more than 'N' times 't' of the data points that we have. At that point we're done, we don't need to check any sets-size larger than 'K' because we know that they can't exist, so that's the apriori algorithm, that's the steps that we follow for checking whether different subsets appear more than fraction 't' number of times. Now the question is, does this algorithm check every subset? So, we're going to assume, that if we present a subset for checking, then we follow the brute-force counting, we literally go through all 'N' objects that we have– all 'N' data points– and we check, whether that particular subset appears in each one of those data points and count the number of times it appears. We then divide that count by 'N' and if that fraction then, is greater than 't', we keep that subset, if it's not, we throw it away. So, we're going to assume that we can find every- that if we're presented with a subset–we'll be able to then verify whether it's something we should keep or not. So, really the only question is, will we check every subset that passes this threshold? So, the proof is fairly straightforward, and it's a proof by induction. So, we're going to imagine, we're going to assume, that we know every single subset of size 'K' minus one, for which the empirical probability, of that subset is greater than 't'. Then we need to show that if we know every subset of size 'K' minus one, we're going to check every potential subset of size 'K', that could possibly have empirical probability greater than 't'. And so, let's look at this through an example, of subsets of size three.

So, for example, assume that we know that the set 'a', 'b', and 'c' appears in more than 'n' times 't' baskets. The question now is, will we check it? So, if we know that 'a', 'b', and 'c', appear in more than 'N' times 't' baskets, we know that all subsets of this set of three objects has also appeared in more than 'N' times 't' baskets. So, for example, we know that 'a' and 'b' have to appear, co-occur, in more than 'N' times 't' baskets. We also know that 'c' has to appear in more than 'N' times 't' baskets. So, by assumption again, we want to prove for 'K' equals three- by assumption, we're going to assume that, we found the set of size two, 'a' and 'b' and we know that this set 'a' and 'b' appears in more than 'N' times 't' baskets by assumption. So, we know that this satisfies this threshold, from the apriori algorithm, because the first step checked all subsets of size one, we also found the set 'c' and we know, that this set also occurs more than 't' fraction of the baskets. So, we're going to check the union of 'a' and 'b' with 'c'. So, we know that we will check 'a' and 'b' and 'c' when at the third step in this algorithm, and so, now we follow by induction, we know that we have all subsets of size one by the brute-force search, and by induction. We therefore know that we have all subsets of size two by brute-force search; and therefore, because we know we have all subsets of size two. Again, by these rules, we know that we have all subsets of size three and so on.

So, it's an inductive proof. So I want to point out that, as written, the apriori algorithm can lead to duplicate sets for checking. So for example, if we have the set 'a' and 'b' and the set 'c' we can take their union. And also if we have the set 'a' and 'c' and the set 'b', we could take their union, but those two sets are the same. So, we could actually, by the apriori algorithm, check these two sets, as if they were different sets, when in fact they're the same set. So, indexing methods can be used to make sure that we don't do this. Also for each proposed set 'K', should we iterate through every basket for checking? Again, there are tricks that we can use to make this faster, that take structure into account, so we don't have to actually iterate through all 'N' baskets to check whether a proposed, 'K' appears in that basket or not.

Video 5 (12:51): Finding Association Rules

Okay! so the Apriori algorithm has shown that we can find all sets 'K', such that the empirical probability of that set in our data is greater than 't'. Now we want to use this to find association rules. So, these rules are of the form, the probability of 'A', given 'B', is greater than some other number 't'—'t_2, where we split 'K' into two subsets 'A' and 'B'. So, the union of 'A' and 'B' is equal to 'K', and the intersection of 'A' and 'B' is the null set. And now, for every single set 'K', we want to check all possible splits of that set into sets 'A' and 'B', such that we can find the conditional probability. And then, if that conditional probability is greater than 't_2', we keep that as an association rule. If it's less than 't_2', we throw it away. So, the question now is, how can we do this efficiently? So, first let's notice that the conditional probability of 'A', given 'B', is equal to the joint probability of 'A' and 'B', which is the probability of the set 'K', divided by the probability of the set 'B'. So, if the set 'K' is greater than 't', and if 'A' and 'B' partition 'K', then we know that the probability of 'A' and the probability of 't'—'B' are also greater than 't'. And since the Aprior algorithm found all subsets, such that the probability of that subset was greater than 't'. The Apriori algorithm also found the set 'A' and the set 'B', separate from their union, and also found the probabilities of those two sets. So, the take-home message is that once we have run the Apriori algorithm and found all subsets such that their empirical probability is greater than 't', we store those subsets and we store those empirical probabilities. And then we can quickly check all conditional probabilities by taking their stored probabilities.

So, for example, if we're interested in taking 'K' and splitting it into two subsets 'A' and 'B', then we know from Apriori that we have found the probability of 'K'. So, we simply look that probability up and put that in the numerator. From Apriori, we also found the set 'B' of—as being of interest, separately from 'A' and separately from 'K'. And so, we look up the set 'B' from our Apriori algorithm and readoff the probability that we found for that one, and then put that in the denominator to get this conditional probability. So, we don't need to go back to the data. After the Apriori algorithm, we stored all of the probabilities of the sets of interest. We don't need to then go back into the data to find these conditional probabilities. We can simply take the probabilities that we got from the Apriori algorithm and divide the probabilities of interest to get the conditional probabilities. So, let's look at an example of what the Apriori algorithm learned on a questionnaire problem. So, for this problem, we have 6876 different questionnaires. So, that's the number of market baskets that we have. Each questionnaire had 14 different questions and those are shown here. For example, one question was the person's sex. One question was about the person's marital status. Another question was about the person's age and education, occupation, and so on. And then, those were given different values. So, for example, sex had two possible answers. Marital status had five possible answers. Age had seven possible answers and so on.

So, for example, for marital status, those five possible answers were five categorical values, for example, single, married, divorced, and so on. For age, they were seven ordinal values, meaning that age ranges

were given, and then you simply said which age range you fell in. So, categorical here means that—simply that if you are in the first category, there's no ordering of your relationship to the second category. So, the categories have no ordering at all. They could be completely reordered. However, the age response has an ordinal relationship. So, for example, if you're in the first age bracket, you're younger than the second age bracket, and the second is younger than the third age bracket, and so on. So, there's ordering to these responses. So, what we did was, created '50'— for this problem, we created '50' different items. And, we did that based on whether the response was categorical or ordinal. So, if it was a categorical response, then however many categories were there were, there were that many items. So, for example, for marital status, there are five different categories, and so we turn that into five different items. And so, you can think of a person's questionnaire as being their basket, and for the marital status, they picked one of five different items and put it in their basket. So, in this case, only one of those five items could appear in the basket, and there had to be one of the five items in the basket. For the ordinal responses—those were turned into two different items based on whether the response was greater than the median or less than the median. So, for example, for age, those seven different responses were binned into two different responses, for example, the first three values and then the next four values. And so, we reduced those into—to do two different responses. And so, based on this type of an item encoding, it's clear that no basket can actually have every single item. Each basket can only have one of the categorical values for each question, and can only have one of the two possible ordinal values based on whether it's less than the median or greater than the median.
And so, this makes it clear that this concept of association analysis is actually more general than simply analyzing customer behavior or purchasing behavior. So now, let's imagine that we want to find different rules. We want to find different subsets of high probability. Already, with this many items and this many responses, we can't use a brute- force search of all the possible subsets. So, we have to use the Apriori algorithm. So, that was, what was done. And then, let's look at two different association rules that we learned from this data set. So, for example, based on the questionnaires, there was one association rule that was learned that said, if your language at home was English and you own your own house, and your occupation is a professional or a managerial category, then there's a good chance your income is greater than 40,000 dollars per year. So, let's look at for example, for this association rule, this had a prevalence or a support of '13.4' percent. What that means is that in the 6876 different questionnaires, '13.4' percent of those questionnaires had these four responses selected. So, there were other responses that the people gave in the questionnaires, but '13.4' percent of the questionnaires of the respondents said that they spoke English, they owned their own home, that their occupation fell in this category, and they earned more than 40,000 dollars per year. So, that's the prevalence that we learned from Apriori. This subset of four objects was one subset that we learned as being of interest. We then took this four-object subset and broke it into two...two sets, one 'A'—set 'A' of these three objects and one set 'B' containing this one object, and then learned the rule, learned this rule with a confidence of '80.8' percent. What this means is, that among all of my questionnaires, if the person responded that they spoke English, that they own their own home, and they selected this as their occupation, then '80.8' percent of the time, they also said that their income was greater than 40,000 dollars per year. '19.2' percent of the time, they said that their income was less than this. So, that gives the confidence. So, we can be very confident that if these three things are true about a person, then we can be '80.8'

percent confident that this is also true based only on the questionnaire information. This association will also add a lift of '2.13'. What this means is that I am about two times more confident that a person makes more than 40,000 dollars per year, if you tell me that these three things are true about that person.
If I don't get to know anything about the person, then I am only half as confident that, that person will make more this amount of dollars per year. So again, this was only information that we could learn based on the questionnaires. So, if the questionnaires are not representative of a population of interest, then all of these rules are not—are bogus. They're not of interest. These rules only apply to the population who answered the questionnaires. And if, that population is representative of the population we care about, then these rules will make sense and can be useful to us. If they aren't, for example, if we want to predict what's going on in one state based on questionnaires from people in another state, our rules might be biased in some way, and so we might not be able to trust what we learned from the questionnaires. We also learned a second association rule— so I should say these examples are from the book, 'The Elements of Statistical Learning', and what the second rule is— simply says that if a person speaks English at home, their income is less than 40,000, they're not married, and they have zero children, then with '82.8' percent confidence, their education is less than a college degree. So, this is again a rule that we learned simply from the questionnaires that we had. This rule supports '26.5' percent. Meaning that '26.5' percent of the questionnaires satisfied all five of these properties, and the lift was about two again. Meaning that I'm twice as confident that a person does not have at least a college degree, if you tell me this information about that person, than I would be otherwise. So again, these are all things that we simply can find by counting from the data. We count how many times different things occurred. We count—we calculate these empirical probabilities based on our data. The reason why this is not a trivial task is because we have many responses. We have many data points or observations, and they occur among many different possible objects. So, they contain subsets of many different possible objects, meaning that the number of things that we have to check is countably impossible. It would take essentially an infinite amount of time for a computer to check every single possible thing, such that we could confidently say that we found all of them, the things of interest. By running this Apriori algorithm, we have a way to very quickly find every single probability of interest, guaranteed by simply eliminating— counting things that we already know in advance will not satisfy our threshold.

Video 6 (5:57): Model Selection

So, we've already seen, with many of the models that we've been discussing that there are certain parameters that we need to set in advance. And we haven't discussed in too much detail, how to go about setting those parameters, but we did briefly touch on the general approach of cross-validation. And so, we saw how we could hold-out parts of our data, and then learn models on the other parts of our data for different parameter settings, and then see how we—well we do on the held-out test set. So, that's the general idea of cross-validation. Another model selection problem is determining the model order, and this is a little bit different from determining model parameters. In that, we have to

actually decide the model to begin with. So, for example, with the Gaussian mixture model, we have to decide how many Gaussians are we going to learn. With matrix factorization, we had to determine the rank of the matrix factorization that we were going to learn in advance. And for Hidden Markov models, we had to define how many states we were going to learn in our model. So these problems are a little bit more, than just simply learning how to set a particular parameter for the same model.
It's learning how to choose the model to begin with. So, this is the model selection problem. There are many ways that we can handle this issue. Cross-validation is actually still one of those ways. In this lecture, I'm going to focus on two new ways for handling the model selection problem. So, in each of these problems of determining the model order, we can't simply look at the log-likelihood of different models for different orders because we can always fit our data better with a more complex model. So, we need new techniques. So, in this lecture, we're going to discuss two approaches to selecting the appropriate model order. This is different from selecting the appropriate model type as follows. So, for example, here are two potential data sets that we could have. If we want to learn a Gaussian mixture model on these two data sets—in the first one, we've picked an appropriate model. We could model this with a Gaussian, but we've picked an inappropriate model order by learning several more Gaussians that are necessary. So, the data requires three Gaussians and we learned six Gaussians in this case. So we picked the wrong model order, but had an appropriate model type. In this plot on the right, if we wanted to cluster this data into three clusters using a Gaussian mixture model and the original data space, we could argue that we haven't even picked the correct model type to begin with. So a Gaussian mixture model on this type of a data is inappropriate to begin with. And so, selecting the correct number of Gaussians or the correct model order is not even an issue that we should discuss. We should first find the correct model. So, in this lecture, we're assuming that we've picked an appropriate model type. We're assuming that if we're working with a Gaussian mixture model that is an appropriate model to be working with. And now we're interested in picking the correct model order for that model type. To do this, we're going to work with this type of a model setting. So, let's set some notation. We're going to write for 'L', the log-likelihood of a parameter under a model. So, we have this type of a model assumption. We're going to assume that we're getting data 'xi', drawn *iid*, from some probability distribution on 'x', given a model parameter. And now, we want to think of the log-likelihood of this data. So, this is something we've discussed very early in this course, which we can write in this way. So, the log-likelihood of a sequence of 'N' observations, that are *iid* from this probability distribution is just the sum over each of the log-likelihoods on each data point. And now the maximum likelihood solution for this, as we discussed in the first lecture, is simply to find the value of Theta that maximizes this log-likelihood term. So, if this is the setting that we're working with and we are, for example, wanting to learn how many clusters we should learn in our Gaussian mixture model, so in that case, this likelihood would be the Gaussian mixture model likelihood, and Theta would be all of the parameters for the different Gaussians as well as the mixing weights on those Gaussians. The incorrect ways to go about trying different Gaussian mixtures, different model orders, maximizing the log-likelihood, and then picking the best one. So, we can't simply try a Gaussian mixture with three Gaussians, four Gaussians, five Gaussians, in each case learn the log- likelihood–learn the maximum of the log-likelihood and then pick the best performing one, because we're always going to be able to fit our data better and better with a more complex model.

So, for example, if you want to immediately see this, we can perfectly fit the data by simply picking 'N' Gaussians in our Gaussian mixture model, if we have 'N' data points, and putting a Gaussian centered on each data point, and then letting the covariance go to zero. In that case, we can perfectly fit all of our data. We can make our log-likelihood go to positive infinity. But that is going to over-fit our data. And so, a more complex model over-fits our data. And so, we can't simply look at the log-likelihood, in order to pick the best model order. And so, we need a more technique.

Video 7 (8:10): Number of Parameters

So, to generalize this problem, what we have— the issue that we're facing is that a model with more degrees of freedom is more prone to over-fitting. And, by degrees of freedom, we mean roughly the number of scalar parameters in our model, which we're going to call 'K' in this lecture. And so, by increasing 'K', by increasing the number of model parameters that we're giving ourselves to fit the data, for example, by increasing the number of clusters and the number of model parameters that goes with that or increasing the rank or the number of states, we can add more degrees of freedom to the model, and therefore fit our data better and better, and eventually over-fit our data. So, what are some common solutions to handling this problem? The first two, which we aren't going to focus on in this class but I still want to bring up, involve stability issues. So, with this, what we can do is, bootstrap our data set and then learn a model on the bootstrapped data set, and see how it generalizes to the original data set. And so, we can do this multiple times for multiple bootstraps, multiple model orders, and then see which model order is the least sensitive to the bootstrapping.

So, which model order is the most stable under that bootstrapped approach, and then we can pick that model order. So, this issue of picking the most stable model using the bootstrap is one approach.

Another approach involves setting a specific prior on the model order using something called Bayesian nonparametric methods. And what this does is actually set a prior distribution on every possible value of 'K'. So, for example, for the Gaussian mixture model, there are potentially an infinite number of Gaussians. So, a Bayesian nonparametric approach would place aprior distribution on each of those infinite clusters or potentially finite number of clusters, and then pick the best number of clusters through posterior inference. So, in that sense, we would learn the model order as simply a parameter, like any other in the model, and then the posterior distribution will give us the best model order. So, this is another set of techniques that's very developed that we won't discuss in more detail. Another approach uses penalties on the log-likelihood, which makes adding more and more parameters, more expensive. So, we work with the log-likelihood and then we add an additional parameter, penalty that's not part of the log-likelihood that involves a number of parameters, and as we increase the number of parameters, we pay more and more of a price for learning that number of parameters. And therefore, in order to overcome the penalty that we pay by increasing the number of parameters, we have to have a decrease in the log-likelihood that's greater than that penalty.

And so, this is the approach that we are going to discuss in this lecture. So, in the rest is lecture we are going to focus on two techniques, where we add a penalty to the number of parameters that we are going to learn in our model. So, in general, we have to define a penalty function, that is a function of the

and also potentially a function of the data size as well, and then instead of maximizing the log-likelihood 'L', we are going to, in this case, minimize the negative of the log-likelihood plus the penalty that we add. So, typically these are presented with negative log-likelihoods and a penalty that we now want to minimize, instead of maximize. And so, the two popular approaches in this text—set of techniques is the Akaike Information Criterion, AIC, and the Bayesian Information Criterion, the BIC. And so, what the AIC does is it takes the negative log-likelihood of the model and it simply adds the number 'K', where 'K' is the number of free scalar parameters that I can set— that I can vary in my model. So, 'K' is potentially a very large number. The BIC is a simple modification of that, where we take the negative log-likelihood. So, this is what we've been working with all throughout the course, the negative log-likelihood, plus we add to that one-half times, the total number of scalar parameters in our model, times the log of the number of observations that we have in our model.

So, the two approaches are very similar. In the AIC, we add a penalty 'K', which is the number of scalar parameters in our model. The BIC takes 'K' and it multiplies it by one-half times log of the number of observations. And so, we can see that when one-half times log 'N' is greater than one, the BIC is going to encourage a simpler model than the AIC. So, with the BIC, by increasing the number of model parameters 'K', we are paying more of a penalty, because 'K' is being pre-multiplied by a number greater than one, whereas with the AIC, 'K' is being multiplied by one. And so, this happens when 'N' is greater than or equal to eight. So, when we have eight or more observations, the BIC encourages a simpler model than the AIC. So, what's an example of this? For example, we have discussed the Nonnegative Matrix Factorization model, NMF. And so, if we have this model with 'M_1' rows in the observation matrix and 'M_2' columns in our observed matrix. So, the data matrix that we want to factorize is 'M_1' by 'M_2', and we decide on a rank 'R' matrix factorization that we want to learn. So, we want to learn a matrix factorization that is 'M_1' by 'R', times 'R' by 'M_2'. In that case, the total number of model parameters that we want to learn, 'K', is equal to 'M_1', times 'R', plus 'M_2' times 'R'. So, we have this situation, where we have our data matrix, that's 'M_1' by 'M_2'. We want to approximate that with a factorization that's, 'M_1' by 'R' and 'R' by 'M_2'. So, we have to learn every single set of model...model parameters in these two matrices, and so the total number of values that we need to learn is 'M_1' times 'R' for this one, and 'M_2' times 'R' for this one.

So, 'K', in this case, is equal to 'M_1' plus 'M_2' times 'R'. So, it's a very big number and for each increment in 'R', we are adding 'M_1' plus 'M_2' new parameters that we have to learn. The BIC simply takes that, and then multiplies 'M_1' plus 'M_2', times 'R', by one-half. And then, the total number of observations that we have is the total number of values in this observed matrix, which is 'M_1' times 'M_2'. So, in this case, 'N' is equal to 'M_1' times 'M_2'. So, the BMC—BIC penalty would add this. The AIC would add this. And then, the log-likelihood is simply whichever penalization that we choose. Remember, with the NMF, we had this Frobenius Norm Error, the 'L2' error, and we also had the divergence penalty. So, we pick one of those and plug that in for minus 'L', and then add the appropriate penalty, whether we're doing AIC or BIC.

Video 8 (8:59): Examples

So, here's an example of what the AIC outputs. So, this is–we can keep the model abstract, and simply say that this dimension, 'x' dimension is a function of the model complexity. And then, this 'y' dimension is the log-likelihood. And so, as we increase the model complexity, the log-likelihood on the training data is monotonically improving. So were fitting the data better and better, and better. But then, if we add the number of parameters that we have to learn as a function of this 'x' axis, the AIC, as you can see, is still getting better. So, the improvement that we get by making a more complex model, in terms of the log-likelihood, is more than enough to compensate for the penalty that we have to pay by introducing new model parameters. So, we're still decreasing in this part of the plot, but then at a certain point, the penalty that we're paying by adding more and more parameters is greater than the decrease that we observe in the log-likelihood. And so, we've-at some point, our AIC penalty is going to increase. And, also shown on this plot is the testing performance on a held-out test set, and you can see that this also follows the expected pattern, that we improve our testing performance and then at a certain point, we start over-fitting our data, and the testing performance or the testing log-likelihood starts to get worse. And so, here, for the same problem, is the '0-1 Loss', so this is a classification problem. And so, we're plotting here the misclassification error. So, we want this to be small. And again, as you can see– so, this is the same problem, just showing the plot for a different 'y' axis. As we increase the complexity of the model, the training performance is getting better and better. So we're classifying our training data better and better and having fewer and fewer errors on our training set, as we increase the model complexity. For the testing set, this blue line, we see what we expect to see. As the model gets more complex, we get a better testing performance. But then, at a certain point, we start to over-fit the training data, and so, our testing performance starts to get worse and worse. And the AIC-and so, this is an ideal case. This is-in this situation, it works out very nicely that the AIC penalty is mimicking that pattern.

So, the AIC-using the AIC penalty, we would want to pick this model complexity. We would basically use the green line to choose the model, not the orange line. Here's another example of the AIC versus the BIC, so, these two combined on an HMM problem. So, again with the HMM, we have to decide how many states are we going to learn in our HMM. And so, what this plot here is showing, is the negative log-likelihood of the data, the sequences that we're modelling for a '1-state' through a '6-state HMM'. And so, what we can see here is that the negative log-likelihood is monotonically decreasing as we add more and more states to our HMM. We're fitting our training data better and better, and so, the negative log-likelihood is always going down–the log-likelihood is always going up. Also shown here, and this is to compare two different models, is a Gaussian mixture model. So, if you remember our discussion about continuous HMMs, they're like Gaussian mixture models, except the Gaussian mixture model generates the cluster assignments *iid* from a mixing weight on the Gaussians, whereas the HMM has a transition distribution. So, instead of using an HMM to learn the sequences of data, we disregarded the sequential information and just learned Gaussian mixture model with two, three, or four Gaussians. We also have our negative log-likelihood, which is equal to these three numbers.

So, we see that, according to the log-likelihood, the HMM is better than the Gaussian mixture model for this data. So, we're modeling sequential information with the HMM. However, in both cases, as a model complexity increases, the log-likelihood is better, meaning the negative log-likelihood is always decreasing. And so, we can't use the log-likelihood to pick out the model or the model complexity, in this case, because we can always pick a number of Gaussians to make the negative log-likelihood smaller than HMM for certain number of states. So, we need to use some additional criterion to pick the correct model order, and in this case, we could even compare the two models directly. And so, here's what's being shown in these two columns, the AIC and the BIC. And so, for example for the '1-state HMM', we have this value '391' as the negative log-likelihood, we then add to that the number of parameters, that we have to learn for this particular '1-state HMM', and that's equal, and that brings the final penalty to this number. For the BIC, we add the same thing except, we also have one-half times the log of the number of observations, and so our BIC penalty is this number. And then, as we increase the complexity of the model, we are improving the negative log-likelihood by decreasing it, but we're adding more and more of a penalty.

And so, taking the negative log-likelihood for a '2-state HMM' and adding the penalty for the AIC or the BIC, produces this number. And so, you can see that by increasing the HMM from '1-state' to '2-states', we're modelling the data so much better that we still improve the AIC or the BIC, so our improvement in the log-likelihood is more than the penalty we're paying for adding more parameters. However, at some point, in both of these cases, we start to pay more of a penalty. For example, when we go from a three to a '4-state HMM', we're paying more of the penalty for adding those parameters, than we're gaining by improving the log-likelihood. So, here's a plot of that. And we can see that as we add more states, the AIC and the BIC are both improving. But then at some point, we start over-fitting our data and both the AIC and BIC start to increase. And because the BIC penalizes model complexity more, the BIC is always greater than the AIC, and also looks like it's potentially picking a less complex model. In the rest of this lecture, I want to discuss the derivation of the BIC, in more detail to show where the motivation is for adding this penalty. These numbers aren't simply pulled out of thin air.

There's a mathematical reason why we're adding these penalties for these two criteria. Their-they both have their separate motivations. So, again let's recall what is the AIC, and what is the BIC for doing model selection. The AIC is going to look at the negative log-likelihood and add 'K', which is the number of parameters in the model. The BIC is going to look at the negative log-likelihood, and add one-half times 'K' times the log of 'N', where 'N' is the number of observations from the model. So, first let's notice, before we go into more mathematical detail, that algorithmically there's actually no more work that needs to be done. What we do is, we simply find the maximum likelihood solution for various model orders. So, we run our Gaussian Mixture Model or HMM etc. for different model orders for different model complexities, find the best model in each complexity class, and then find the log-likelihood or the negative log-likelihood for each of the those solutions. And then, we simply add to that result either the AIC or the BIC penalty, which we can calculate in advance. This-these two terms are independent of whatever parameter values the algorithm actually converges to. So, the natural question is to ask, where these penalties are coming from, because currently they seem a bit arbitrary. So, next we're going to discuss the BIC, and just simply mention that the AIC has its own theoretical motivation. But we aren't going to discuss those in this course.

Video 9 (6:42): Deriving the BIC I

So, focusing on the BIC for model selection, let's imagine that we have 'r' different candidate models, 'M_1' through M_r'. So, they could be Gaussian mixture with one Gaussian up to a Gaussian mixture 'r' Gaussians, or an HMM with 'one-state' versus 'two-states', up to 'r' states. Let's also imagine that we have data. We're going to call that 'D', which is 'N' observations x_1' through x_N'. And now, what we want with the BIC—the starting motivation is that we want to find the posterior distribution on each of the models. So, for the i^{th} model, we want to find the posterior distribution of the model 'i', given the data. And so, what this is asking in using Bayes' rule is for us to calculate the likelihood of the data, given the model, times a prior on the model, divided by a sum of the numerator over all the different models. And this way, we're getting a posterior distribution on 'r' models. We're not getting a posterior distribution on their parameters, but we're actually getting a distribution, a posterior distribution on the model itself.

So, this is saying, a posteriori, which model do I think generated this data. So, from the standpoint of motivating the BIC, we're going to assume that each model has a uniform prior that—a priori. We aren't biased towards any model. They're all equally probable. And then, because the denominator is simply a normalizing constant, what we can then say is that by finding the most probable model, we want to find the arg max of the log-likelihood of the data, given each model. So, we picked model 'i'. We calculate the log-likelihood of the data, given that model. And then, we're going to pick the most probable model, according to that likelihood to tell us which is the model order that we should select. And so, notice that there are no model parameters here. So, where they've gone is indicated by this integral. Actually, by writing the log-likelihood of the data given the model that we've chosen, what we're really saying is that we want to calculate the log-likelihood of the data given the model and a particular parameter setting, times the prior of that parameter setting given the model, and then integrated over all possible model parameter settings. So, for example, if 'M_i' is a Gaussian mixture model with 'i' Gaussians, 'M_i' is simply—indicates that, that is the model that's chosen. A Gaussian mixture model with 'i' Gaussians. Theta, then would actually contain the parameters for that 'i' Gaussian mixture model. It would contain the 'i' different means, the 'i' different covariances, and the 'i' dimensional probability distribution on those 'i' Gaussians.

So, all of the model parameters would be contained in Theta. We define some prior probability on those parameters, and then we integrate them out. And I apologize, this integral should be inside of the log. So, what we're choosing with the BIC, in principle, is the model with the largest marginal likelihood of the data by integrating out all of the model parameters. However, this is, in almost all cases not an integral that we can solve, and so we have to come up with some sort of an approximation. And, we'll see how the BIC criterion arises by making an approximation of this log marginal likelihood of a data set, given the model class as equal to the log-likelihood of the data, given the maximum likelihood parameter setting of the model, and given the model class, the model order, minus the BIC penalty,

minus one-half, times 'K', times the log of the number of observations. So, the BIC is a mathematical technique that motivates approximating this log marginal likelihood by this sum of the log-likelihood, given the maximum likelihood parameter setting, and this additional penalty that we're going to pay for the model complexity of the i^{th} model.

So, in the remainder of the lecture, I want to simply show, why this value can be approximated by this value, in which case, if we replace our goal of picking the best model according to this marginal likelihood with picking the best model according to this value, we simply have to find the maximum likelihood parameter setting for each of the models that we're checking over and then, add the appropriate penalty for the model complexity. So, first let's notice that the difficulty is with the integral. So, I'm rewriting the integral with the correct ordering here, where we say that the log of the likelihood of a data set, given the model order, given the particular model that's chosen is equal to the log of the integral of the data, given a parameter setting, times the prior probability of that parameter setting that we define. And so in here, again, both of these probabilities are going to be conditioned on 'M_i' as well, the model that we've actually chosen. But in the rest of the lecture, I'm going to suppress the conditioning on the model. So, when you see this likelihood and this prior what's implied is that we've actually picked a model to begin with. We've picked, for example, an 'i' Gaussian mixture model and then this is the appropriate prior on the parameters of that model. And we've also—in picking an 'i' Gaussian mixture model, this is the appropriate

Likelihood of a dataset coming from that model. So, what we're going to do is very similar to what we did with the Laplace approximation towards the beginning of the course. We're going to approximate this integral using a second order Taylor expansion.

Video 12-10 (11:54): Deriving the BIC II

So, let's work through the steps of deriving the BIC. Again, remember that what we want to do is calculate the log of the marginal likelihood of data, given the model. And we're going to write that as the log of an integral over the likelihood of the data, given a model parameter setting, Theta times the prior probability of Theta that we are going to define integrated over Theta. So, this is the marginal likelihood of a data set represented as an integral over the model parameters. And then, we're going to replace this likelihood of the data, given Theta, with an equivalent representation of the exponential of the log of that likelihood. So, of course, this term is equal to this term because the exponential and the log cancel out. However, when we represent it this way, we can then approximate the log of the likelihood using the second order Taylor expansion. So, if you'll recall in an earlier lecture when we discussed the Laplace approximation, we did something very similar to this. So, our goal now is to take the log of the likelihood of the data given model parameter setting, Theta, and approximate this with the second order Taylor expansion, and we get to pick any point to approximate this function at, but we're going to pick the maximum likelihood point.

So, if our-of course– if our objective is non-convex, we can't necessarily say, we found the maximum likelihood solution. So, we could run the algorithm multiple times, at multiple starting points, and pick the best solution. Each time it will converge to a mode, it won't necessarily be the best solution.

However, we can pick the best one and then call that the maximum likelihood solution. So, we find a value for Theta at a mode and call that a maximum likelihood solution. And we now approximate the log of the likelihood of 'D' given Theta, with the log of the likelihood of 'D'given–evaluated at the maximum likelihood solution plus a first order term, which notice involves- is a free parameter of Theta, and in both of these two locations, we evaluate the gradient at the maximum likelihood solution; and we take the difference of Theta with the maximum likelihood solution here, plus a second order term, which is one-half times the difference of Theta from the maximum likelihood solution on these two sides, and here, we have this-the matrix of second derivatives of the log of the likelihood, evaluated at the maximum likelihood solution. So, these two terms are free functions of Theta, and this term is a constant. And everywhere else, we're evaluating Theta using the maximum likelihood solution. So, notice that by evaluating this at the maximum likelihood solution we can cancel out this term, because the gradient at that location is equal to zero. And so, this term goes away, and we're left with these two terms. And now this matrix, I'm going to call negative 'J', and it's going to be a function of maximum likelihood solution.

So, I'm going to call this shorthand negative 'J'. Then if I take my log-likelihood in this integral and I replace it with these two terms, what I find is that I can approximate the log-likelihood of a data set given a model, by the log-likelihood of the data set where I evaluate that model at the maximum likelihood solution, plus the log of this integral. So that I found this approximation by simply approximating this log likelihood with these two terms, plugging them in, I get that this is approximately equal to this. So, now let's look a little bit closer at the integral that we have to evaluate, at this integral here. We notice, and this is exactly what we saw with the Laplace approximation as well, that this integral is simply the normalizing constant for a Multivariate Gaussian, with mean equal to the maximum likelihood parameter setting and precision matrix equal to 'J', this matrix of second derivatives.

Therefore we can say that this integral is equal to two Pi, divided by the...the determinant of the matrix 'J', raised to the power 'K' divided by two. And so, where is 'K' coming from? Remember Theta is every single model-parameter-scalar- one-dimensional parameter in our model that we can freely change. So, for a Gaussian mixture model, we have all of the Gaussians-means and covariances for all-for the different Gaussians, as well as the dimension as the distribution on those Gaussians. So, Theta is a set of many different parameters. And we're mathematically approximating that with this Gaussian, even though if you'll recall, what the Laplace approximation, we motivated not doing that, we're not-we're actually not going to do this in practice, it's just a mathematical construction. So, Theta can be very high-dimensional. This could be a several thousand-dimensional Multivariate Gaussian; but it's okay, because we're never going to calculate any of these things. Or, I should say, we're never going to have to calculate this huge precision matrix. Okay! So, that's where this 'K' is coming from, because we have 'K' different model parameters in this vector's data. Now let's remember our definition that negative 'J' is equal to the second derivative of the log of the likelihood evaluated at the maximum likelihood solution. So, because–and this is going back to the beginning of the lecture–we are making the assumption that our 'N' observations are generated *iid*. from this model on the data. We can write the log of the likelihood of the data as the sum– so, I'm doing two things here. We can write this as being equal to the sum over each observation of the great-the matrix of second derivatives of the log of the likelihood of

each individual observation evaluated at the maximum likelihood solution. So, originally we can say this is equal to this. But then, if we multiply and divide by 'N', we can then say, that this term is converging to something and then goes to infinity. So, notice we've simple done a trick here. We've multiplied and divided by 'N', so, these two things cancel out. However, if we now focus only on this sum of matrixes, divided by 'N', we're seeing that we have a sum of 'N' things divided by 'N'. And what we can show, which we won't prove in this course but what we can show, is that this converges to a matrix as 'N' goes to infinity. And so, we have 'N' out here, times a matrix that's converging as 'N' increases. So, we can wrap this up now by simply plugging it all back in. So, we take the log of the likelihood of the marginal likelihood of the data given the model, and we approximate that as the log of the likelihood of the data evaluated at the maximum likelihood solution plus–and then here's where we have the integral– the log of the integral, which was the log of the marginalizing constant of a Multivariate Gaussian.

So, the penalty now is the log of two Pi, over the determinant of 'J', raised to the 'K' divided by two power. And so, remember if we have the log of a value raised to a power, we can simply bring this out front and divide- and multiply by 'K' over two. And then, notice what's going on with this term. So, I'll write it out to be more clear. So, this term is equal to 'K' over two, times the log of two Pi, minus 'K' over two times the log of the determinant of 'J' evaluated at the maximum likelihood solution. So, I've simply taken this value and expanded it out, like this. Now notice a couple of things. First, as the number of observations increases, this thing is not increasing very quickly. So, what we're going to do, is simply ignore this term. And now notice what we can say here, that this determinant is equal to 'N' times the determinant of this average matrix, where we sum the matrix of second derivatives over each observation and then, we divide by the number of observations, and so, this is a matrix that's converging. And so, we can further set this to be equal to minus 'K' over two times log 'N', minus 'K' over two log, times the gradient of-and then one over 'N', and then I have a sum 'N' items here, like this. Okay! So, I now have taken this one term and broken it into a sum of three terms, where I have this term minus this term, minus this term.

I've already gotten rid of this term, and now notice that as 'N' increases, this term is converging as well. It's not increasing very fast. So, these two terms are order- are growing order one and 'N', as number of data points increases. And now we are left with only this term, which is increasing in 'N'-as 'N' increases. So, finally what we get at is that we've shown by using the second order Taylor expansion, we can approximate the log of the marginal likelihood of a data set given a model choice, as being equal to the log of the likelihood of the data set evaluated at the maximum likelihood parameter setting for that model, minus one-half times 'K' over two, times a log 'N' plus something that's not growing in 'N', so, something that's order one and 'N'. And so, this is where the BIC makes a second assumption. The first is using the Laplace approximation. The second approximation is saying that since it's not growing in 'N', even though it's growing in 'K', I only care about how it grows in 'N'. And so, I'm going to ignore this term and arrive at my final approximation, which is something that's growing, which is a penalty that grows in both 'K' and grows in 'N'. And so, we see that this additional penalty that we're going to pay, is motivated by a Laplace approximation, which is like a Gaussian approximation on our model parameters.