SHEPPARD: As part of our discussion on probability, I thought it would be interesting to discuss an algorithm that makes use of probability and in a particular, makes use of the properties of independence and conditional independence that we just covered as a way of solving a rather important problem. The problem that we're going to look at is the classification problem. We can think of the idea here being that if you are presented with a piece of data that characterizes through some set of features some object. What we would like to be able to do is based on those features, determine what the type of object is. We call this the classification problem. One of the classic examples would be, for example, if we look at the features of a plant where we start considering, for example the size of the plant, the color of some of the flowering parts of the plant, perhaps some smell. We can use that information to perhaps classify or identify what the plant or what family the plant belongs to. So we are going to look at this probabilistically using a specific kind of a model that we call a Bayesian model, and in particular, a Bayesian network. So we can define a Bayesian network somewhat informally as follows-- a Bayesian network or a Bayesian belief network is actually a graph. So as a data structure, we will represent these things as graphs where the vertices or the nodes of the graph correspond to random variables of a larger joint probability distribution. The edges or the links within this graph are directed. So we're dealing with a directed graph or digraph. And the links pair nodes together, where we interpret the direction of the links to identify causality. So if we have an edge going from node A to node B, we say that A some kind of a causal or probabilistic influence on B. In fact, we'll find later that this corresponds to a dependence of B on A. Furthermore, each of the vertices within our graph, each of the nodes, have associated with the probability tables. As shown here, when we talk about conditional probabilities, where the node is actually receiving one of the edges from another node. On the other hand, if we have a node for which there are no edges coming into it, than we would actually have an unconditional or marginal probability table associated with that node. What are the key properties of our directed graph for this type of a model is that the graph be acyclic. So we're dealing with a DAG, a directed acyclic graph. This shows a fairly basic example of a Bayesian network where our network has five nodes. And we're doing a little bit of abbreviation here. So for example, when we look at the node family out, we will abbreviate that fo. Has fleas will abbreviate hf and so on. In every case, in this example, these nodes correspond to Boolean random variables. So we would actually consider is the family out or not? And it'd be a true, false relationship that we assign to that. And looking at the overall structure of this network, what we're seeing is that we're talking about the relationship between the family been home or not home, there being fleas in the home, there being a dog in the home. Such that if somebody from the outside is observing this home, they might see a light on or not or they might hear the dog barking or not. So the influences that we're representing in this network show that the family being out will have an influence on whether or not they turn the light on and whether or not they put the dog outside or just leave the dog inside. The dog having fleas will also have an influence on whether they allowed the dog to remain inside or it gets put out. And finally the dog being outside will have an influence on whether or not or how well somebody outside the house could hear the dog barking. So with each of these, we associate probability tables. And at the top, the two nodes family out and has fleas-- you see have no edges coming into it. So in this case, we simply assign an unconditional probability. In the first case, the probability of the family being out and the second case, the probability of the dog having fleas. Looking at light on that has a single edge coming into it, and so we are going to consider a conditional probability. What's the probability of the light being on, given the family is out; the probability of the light being on given the family is not out. And we find a similar type of relationship down with hearing the bark. The dog out node is the most complicated node in this network because we actually have a joint dependence where now we're considering the probability of the dog being out, given both the families out and the dog has fleas; given the family is out but the dog does not have fleas; given the family is not out but the dog does have fleas; and given that neither the family is out nor the dog has fleas. So what is interesting about this particular network structure is that it actually is a compact way of representing the joint probability distribution over all five of those variables. Now since these are all Boolean variables, to fully represent the joint distribution in a Naive way would require on the order of two to the fifth or 32 probability values to be stored. However, what this network actually captures are conditional dependence and therefore conditional independence relationships. As an example, if we know the value of family out and we can actually assert that the light being on, the dog being out are independent, so there is an independence relationship with the light on and the dog out, given the family is out. And this allows us to reduce the number of probabilities they need to be specified for 32, down to 10. For very large models, you can see that conditional independence can indeed create a

tremendous savings in the representation of the underlying problem. So what we're going to do is look at a highly specialized version of this network. It's called a Naive Bayes network. And we'll see a graphic of this in a moment. What I'm showing here is actually both the learning algorithm, where if you have a bunch of data you can derive the probabilities that go into this network directly from the data. And then once you have that, we also the algorithm for using it to do classification-- so to solve our classification problem. So what we see for the actual learning problem is we have to estimate the probabilities of all those nodes that have no edges coming into them. And for the Naive Bayes algorithm, those nodes will correspond to the class nodes. So remember our example-- what we're dealing with determining the type of flower. That would be where we would identify the different types of flowers that we're going to be categorizing. And it's simply a matter now of going through the data and counting up all of the instances of each of the different types of flowers in the data and then divide by the number of examples that we're working with. And that becomes an estimate of the probabilities of each of those, this flower types. This slightly more difficult case occurs for the various features. So recall some of the features we considered were the size of the plant, the color of the flower, perhaps something dealing with the smell of flower. These are attributes which we would identify with these a sub i's. So we want to know what is the probability of a large plant, given this is a dogwood, or a small plant given this is a rose. And we do the same kind of thing. First of all, let's suppose we're considering all the roses. We consider only the subset of data corresponding to the roses. And then we go through and count up all the instances of large versus small plants, take those counts and divide by the number of roes that were in the data set and that gives us our estimate of the probability of a small plant given that it's a rose. Now the Naive Bayes algorithm is significant in this discussion because it is the most extreme case of exploiting this concept of conditional independence. Because what we were saying here is that all of the features, all the attributes in our classification problem are independent of one another, given the classes that we're working with. And this will become explicit when we look at the picture of the Naive Bayes network in a moment. The last thing to mention on this slide is given that we've built this network, what do we with it? So what we would do is we would come in with a particular example. So suppose we have a flower that's been classified to be large, the color is red, and there is no odor. So those are the three features. Then, what we would do is we would consider all the different types of flowers that we're trying to classify, whether it's a rose, a dogwood, a carnation or whatever. And we would calculate using this equation on the bottom. The probability of a rose, times the product of these conditional probabilities for the given features we're seeing. So the probability of large, given a rose; probability of a red, given a rose; the probability of odorless given rose. Calculate those products over all of the various classes that we have. So we're considering rose, carnation, dogwood, whatever. Take the resulting values and maximize. And then take the type of flower we're considering with the highest value, and that's what gets returned. That's the classification process using Naive Bayes. So this is an example of what a Naive Bayes network might look like. Where, if you look at the red node at the top, labelled d in this case-- and the labels actually don't really matter-- that's our class nodes. So it would be like the flower is it a rose carnation or whatever. And we associate because their no edges coming into d, the unconditional probabilities as we just described. Along the bottom, we have these yellow nodes that actually represent the attributes or the features. Actually, you would say observation. And here, we would have to associate the conditional probability of the yellow node, given the red node as a parent node. So this represents, graphically, in a very simple way this Naive Bayes network. One of the problems that we have, however, is that Naive Bayes makes very, very strong conditional independence assumptions that, in reality, may not apply. The advantage to the Naive Bayes network is that computationally, when we look at the complexity of the algorithm, both for learning and for doing classification, it's actually very efficient. But at a cost, we're dealing with efficiency but we may have a reduction in accuracy as a result. So what we'd like to do is actually see if we can identify where further dependencies might exist, for example, among the observations or the attributes and reinsert those dependencies in the network. Well, one idea that was proposed is to do this in a very restrictive way so we can still control the computational complexity of the algorithm. The idea is what's called a tree-augmented Naive Bayes network. So in this case, focusing just on the yellow nodes in our network, what we would have is rather than a bunch of disjoined nodes-- again, ignore the red, pretend that's gone. In Naive Bayes, those would all be disjoined. Now we're going to actually overlay a tree structure on this set of notes, recalling that every time we insert an edge here, we're actually saying there's additional dependence, now, between these. What that say is, OK, we're going to relax this strong conditional independence assumption to allow for some additional dependencies but we're only going to permit each of these nodes to have at most, one additional dependence. And that's where the

controller or computational complexity comes from. And now the question is, how do we know which of these edges to insert? Well, we want the edges, again, to correspond to stronger dependence relationships, as opposed to just putting an arbitrary tree here. And this is the interesting part of the algorithm. All right, so the tree-augmented Naive Bayes algorithm, the learning process is described here. It's actually first described in the paper back in 1997 by Friedman, Geiger, and Goldszmidt. The algorithm makes use of a technique that we will be discussing later in this course, where we're going to be finding a spanning tree of a graph. The one difference here is that we're going to be looking for a maximum spanning tree. The algorithm that gets presented later in the course is a minimum spanning tree. And honestly, the only difference between the two algorithms is in the sort algorithm that we apply. So to walk through the way the algorithm works, what we're going to do is we're essentially going to pretty much ignore the parent class node. And then we're going to take all of those disjoined observation or attribute nodes and we're going to connect them up into a complete graph, where every node is connected to every other node. But these edges are-- excuse me-- undirected, at least to start. And then we're going to calculate a measure called the conditional mutual information. And I'll give you the equation for this on the next slide. It's a relatively straightforward thing to calculate, and once again, basically focuses on doing counting in the data set. Once we've calculated this CMI measure, we will associate the CMI measure with each of the edges on the graph and use those edges as weights to find the maximum spanning tree. So what is a spanning tree? A spanning tree is a tree over the same nodes of the graph that you started with, such that if you were to add up all of the weights on the edges in that tree, for maximum spanning tree, that sum would be maximized. So if we were to, for example, remove one of those edges and replace it with another edge so that we still have a tree, the value of that tree would be less than or equal to what we had before. Once we have our maximum spanning tree, recalling that a Bayesian net requires the final network to be directed, we arbitrarily pick one of the nodes of the maximum spanning tree to be the root of our tree and then we direct the edges outward from that root. And finally, we go back and using a similar counting process to what we used in the Naive Bayes algorithm we calculate the conditional probabilities for-- this says test to test but again, the relationships between the attributes are the observations within the graph. And then finally, we add the-- this says diagnoses-- we add the class node back into the network. So that we can calculate the resulting probabilities there. As a comment, the reason that we have these terms "test to test" and "diagnosis" is that one very common classification problem-- and it's actually a classification problem that I deal with every day of my research-- is the fault diagnosis problem, where your attributes correspond to test, to measurements that you take relative to a system and the diagnoses corresponds to the classes that you're going to take is a particular part of the system faulty or not. So to calculate the conditional mutual information, what we have to do is consider joint probabilities of the various features occurring together with their classes, as well as conditioned on the classes. So when we look at the sum, you see that joint probability of attribute i, attribute j, and the class node v. And then in the numerator after the log, you've got the conditional probability of these attributes occurring together, given the feature-- or I'm sorry, given the class. And then in the denominator, we consider the product of the individual probabilities of each attribute, given the class. And what we find is that this is actually a pretty good measure to determine how dependent these pairwise features are on one another. In fact, we'll find that if they're completely independent, the resulting measure will be zero. And the higher the value, the more strongly dependent they are one another. This says nothing about which way the dependence goes, a sub i on a sub j or vice versa. And frankly mathematically, it's irrelevant because using Bayes rule, you could convert any probability of A given B to the other the probability of B given A. So let's walk through a simple example. So here is a Naive Bayes network where we have a single class node V and we have for attribute or feature nodes A1 through A4. The initial network would be specified with the unconditional probability distribution P of V, as well as the four conditional probability distributions P of A1, given V, P of A2 given V, P of A3 given V, P of A4 given V. Recalling once again, we're saying that the four attributes are conditionally independent, given the class node V. Now we're going to relax that assumption. To do that, we start by creating a complete graph that pairwise connects all of the attributes within the network. After they're connected, we will calculate the conditional mutual information. I of Ai Aj given V. And assign those values as weights on each of the edges. So let's assume that we have a bunch of data. We've calculated weights for this particular network. This might, therefore, reflect possible weights they get assigned to each of the pairwise edges between this complete graph, between the four attributes. We're now ready to find the maximum spanning tree. It is a greedy algorithm and we'll talk about greedy methods later in the course as well. And the resulting maximum spanning tree for this example is as follows. So you'll notice that we have here is a tree that has been overlaid on top of the

attributes. And if we were to remove the class node completely, we would still have a connected graph. And if you were to add up those three values and Compare it to any other spanning tree that we might get out of this, you would find that sum is the maximum possible for this example. Now what we need to do is direct the edges. Suppose we select A4 as the root of our tree and then we orient the edges down. This then gives us the new directed graph as shown here. What we now need to do is recalculate the different conditional probabilities on those attributes that have picked up an additional parent node. In this case, V remains unchanged because it's still our class node. A4 remains unchanged because it's the root of the tree. Looking at A1, A2, and A3 however, we have to calculate the probability of A1 given A4 V, the probability of A2 given A4 V, and the probability of A3 given A2 and V. And this can be done, once again, simply by counting through the data. There's a rather nice characteristics that come from the tree-augmented Bayes network. First of all, because any given attribute will have at most two parents, we're still able to control the computational complexity of this algorithm. The complexity doubles but is still going to be polynomial relative to the number of attributes. The other thing, and this is actually significant when we talk about general Bayesian networks, doing what's called exact inference in a general Bayesian network, where you're trying to derive the resulting probabilities from your evidence, is actually a very computationally complex problem. Something that is beyond the scope of this class, but would be addressed, for example, in a course in artificial intelligence. The nice thing here with this particular model is that we can still very efficiently do exact inference over this model. And as we discussed, we're still able to derive the probabilities in the probability tables directly from the training data. And we find that the classification rule is actually a very simple extension of the Naive Bayes rule. We're looking at this equation at the bottom. What has changed is the last product on the right. Well, the only thing that's really different here is that we have the extra dependence in these probabilities that were working. And it still ends up just being a product over these, the different specific values of the attributes. That completes our discussion of a Bayesian classification, the Naive Bayes method, and the tree-augmented Naive Bayes method. What I hope you got out of this discussion is that there is actually a practical use for working with conditional independence and for-- and you'll also get a bit of a preview of some of the types of algorithmic issues and techniques that we'll be considering later in the course.