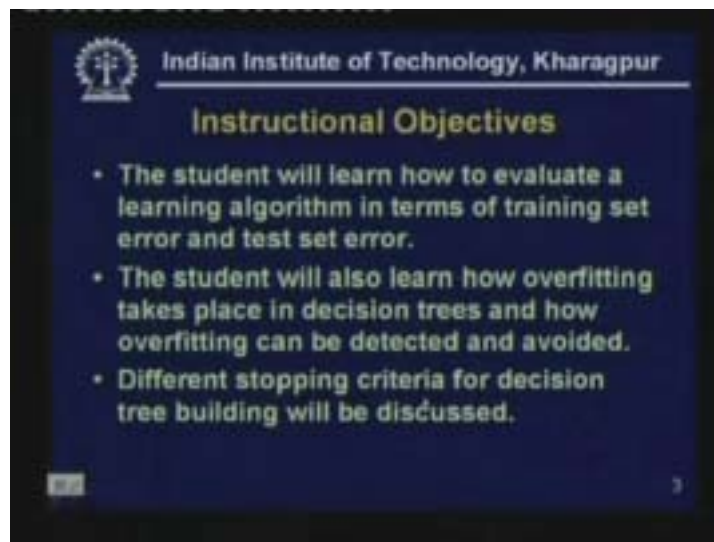


**Artificial Intelligence**  
**Prof. Sudeshna Sarkar**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture - 35**  
**Rule Induction and Decision Trees - II**

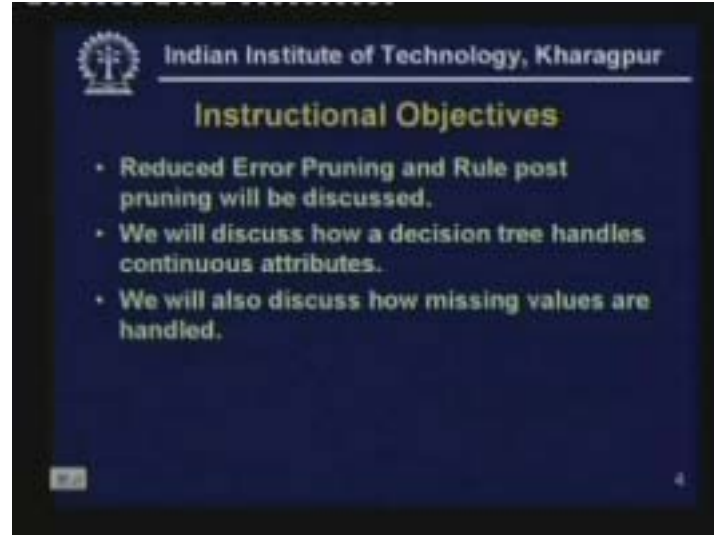
Welcome, today we start the second part of the lecture on decision trees. In the last class we looked at the definition of the decision tree and we also looked at an algorithm ID3 which helps in constructing decision tree or inducing the decision tree given some training data. Today we will further consider the decision tree learning algorithm.

(Refer Slide Time: 1:21)



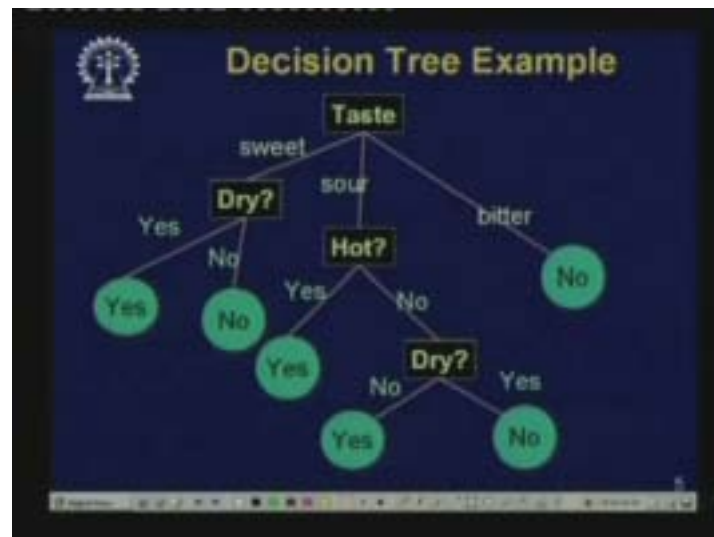
The instructional objectives of today's lectures are as follows. The student will learn how to evaluate the learning algorithm in terms of the error obtained on the training set and the error obtained on the test set. The student will learn how over-fitting takes place in decision trees and how over-fitting can be detected and avoided. We will talk about different stopping criteria for decision tree building. We will look at two different pruning techniques for pruning of the decision tree. First we will look at reduce error pruning and then we will look at rule post pruning.

(Refer Slide Time: 1:21)



We will discuss how a decision tree can be made to work with continuous valued attribute. We will also discuss how missing values are handled in training data where the data is incomplete so some attributes have missing values. Before we start let us review what a decision tree is and look at an example decision tree that we considered in the last class. If you look at this diagram every node in the decision tree the internal nodes in the decision tree test the value of an attribute.

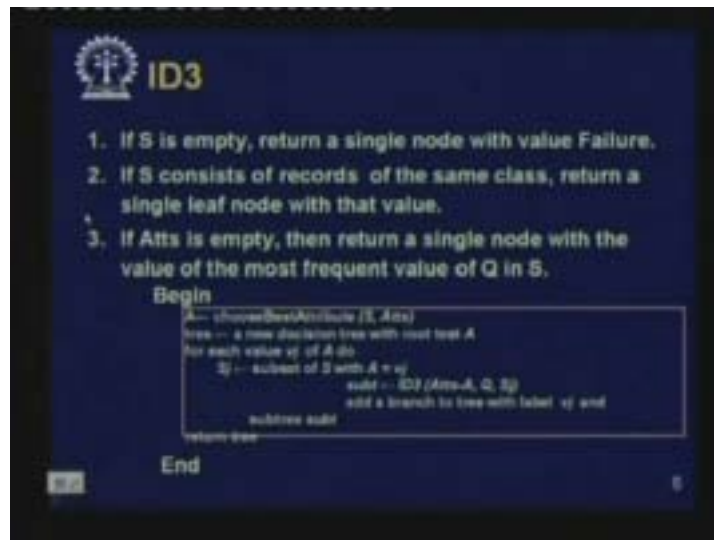
(Refer Slide Time: 2:52)



In this decision tree there are four internal nodes. This node looks at the attribute for test, this node for attribute of dry, this node attribute hot and this node attribute dry. Every leaf node corresponds to a classification. So this, this and this corresponds to yes values and this, this and this corresponds to no values. The branches that come out of an internal

node correspond to the different values the attribute at its parents can take. Taste can take the values sweet, sour and bitter so we have three branches from test, hot can take values yes or no so we have two branches from hot. ID-3 is the simplest algorithm for learning decision trees. This algorithm was developed in 1989 by Ross Quinlan in the machine learning community and parallelly by **Brian Minato** in the field of statistics. So let us look back at ID3.

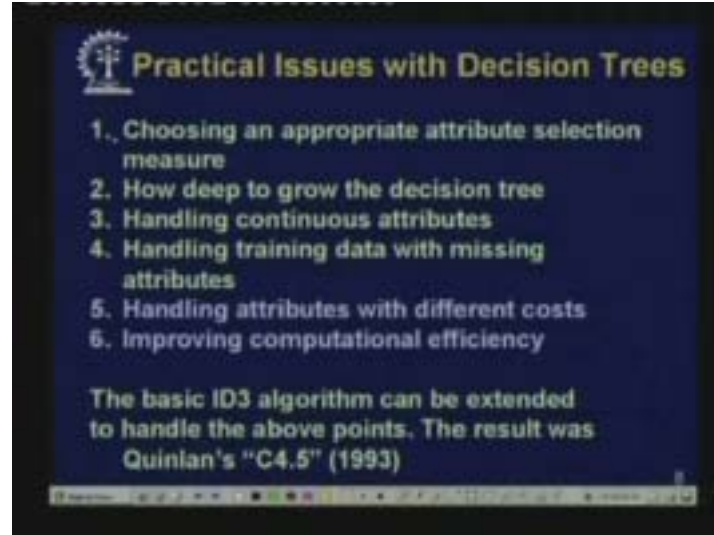
(Refer Slide Time: 4:32)



In ID3 we start with input, it is the set of training examples  $S$ . We have a set of attributes  $att$ , we have a training example  $S$  and  $Q$  is the attribute on which we are trying to classify the data. If the training example is empty ID3 does not proceed further. It just returns single node with a default plus or with the value failure. If  $S$  is not empty but all elements of  $S$  belongs to the same class, that is  $f$  is homogeneous we stop growing the tree and we label that leaf node with the class to which all the examples in which  $S$  belong to.

Thirdly, if  $atts$  is empty that is there are no more attribute left for testing then also we cannot grow the tree any further. In that case if  $S$  is not empty we label the leaf with the majority of the different elements of  $S$  belongs to otherwise we carry on this loop. So, when does one stop growing the tree? We stop growing the tree when either  $S$  homogeneous or when  $atts$  is empty. We also stop if  $S$  is empty. Otherwise if these conditions are not satisfied we choose the best attribute  $A$  by looking at the examples  $S$  and the attributes left. And then we branch on the attribute and grow a decision tree recursively. There are several practical issues that come up in developing a decision tree some of which or not properly handled in ID3 algorithm.

(Refer Slide Time: 7:11)



Some of these issues are:

1) We have to choose an appropriate attribute selection measure. As we have seen there can be different techniques for selecting the attributes the one we have been talking about is using information gain and entropy.

2) We have to decide how deep to grow the decision tree. The ID3 algorithm that we discussed stops when either  $S$  is empty or  $atts$  is empty or when  $S$  is homogeneous. But we might decide to stop growing the earlier based on other criteria. ID3 will also fail if we reach a situation where  $atts$  is empty and  $S$  is not homogeneous. In such situation ID3 will fail and we have to take care of the situations.

3) So far we have looked at how to handle attributes which have a finite number of values. Either they have Boolean values as true or false or they take nominal values, for example taste takes the values sweet, sour and bitter. But what about attributes which take real values? In most of the cases for many training sets the attributes will take real values e.g. the value of temperature, the value of height, value of income. They cannot be put in to a small number of classes. How do we handle continuous values? Decision tree algorithm can be modified to handle continuous values.

4) We want to handle training data with missing attribute values. Sometimes we get a data for which we do not get the values of all the features. Either they are corrupted by noise or the features could not be measured or they are unavailable. So we need to often deal with missing attribute values.

There are other issues which we will not talk about in this class. For example, sometimes attributes have differing causes. For example, in the decision tree once we have a decision tree, when we apply the decision tree we take the example and test with the root node of the decision tree then we follow the appropriate branch. Therefore, trying to

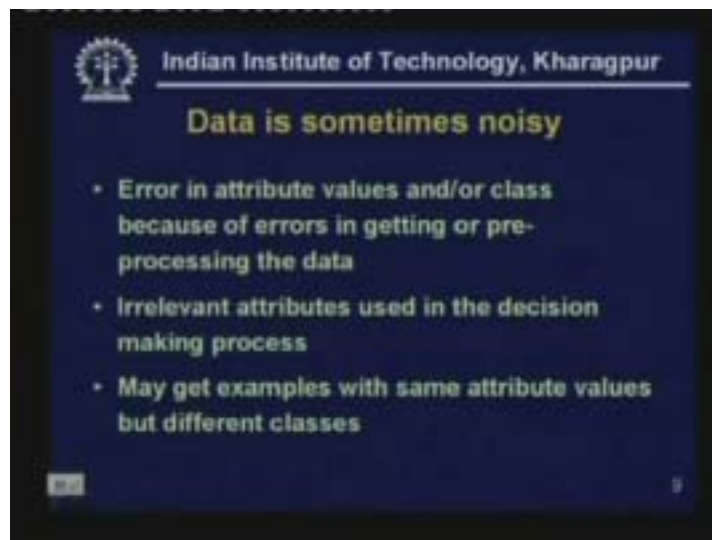
classify the example involves testing some of the attributes. Now it could be testing certain attributes are cheaper, testing certain attributes are more expensive.

Consider a medical diagnosis system. Each attribute can correspond to some tests which have to be carried out. Some tests are cheaper, some tests are more expensive. So we would prefer to carry out tests which are cheaper. So, growing a decision tree and apart from looking at the simplicity or small decision trees it will also take into account the cost of the different tests.

Improve computational efficiency:

These issues are dealt in the later versions of the decision tree algorithm. For example the C4.5 algorithm which was developed by Quinlan in 1993 and there are other versions of decision tree algorithm including C5.0 and then there is the cart algorithm. And then there are numerous other versions of decision tree algorithms which take care of these issues and grow beyond to the basic decision tree algorithm or ID3. Before we discuss some of these issues let us discuss the issue of noise which we have not talked about so far. When you get the training data the data may not be pure there could be noise in the data. The noise can come in because of the error in obtaining the data or error in measuring the data or error in processing the data. So due to this error in measuring or obtaining the data or data precision process can give us noisy data.

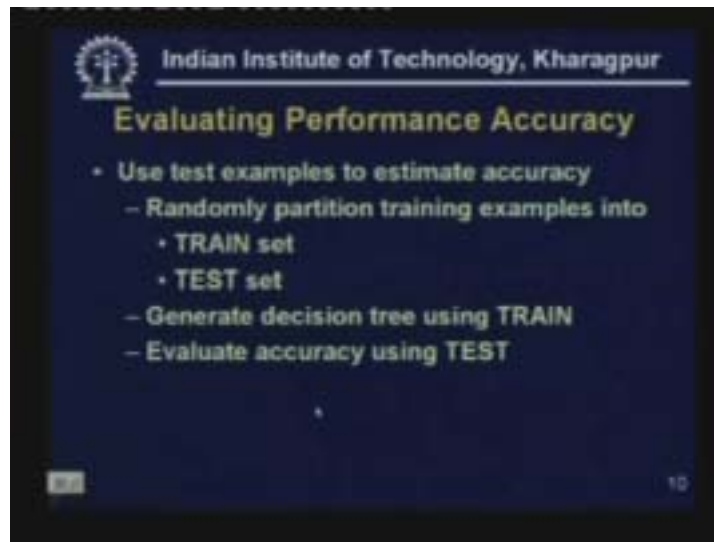
(Refer Slide Time: 11:55)



For example, if the color is actually white it may appear to us as grey and so on. Secondly, sometimes what happens is that we do not choose the feature set properly and our data set includes many features which are actually irrelevant to the classification task. The values of these features do not affect the actual classification. So such attributes actually introduce noise in the classification process. And because of the presence of irrelevant attributes or because of the error in measurement we may get two examples which have exactly the same values of all the attributes but they belong to different classes.

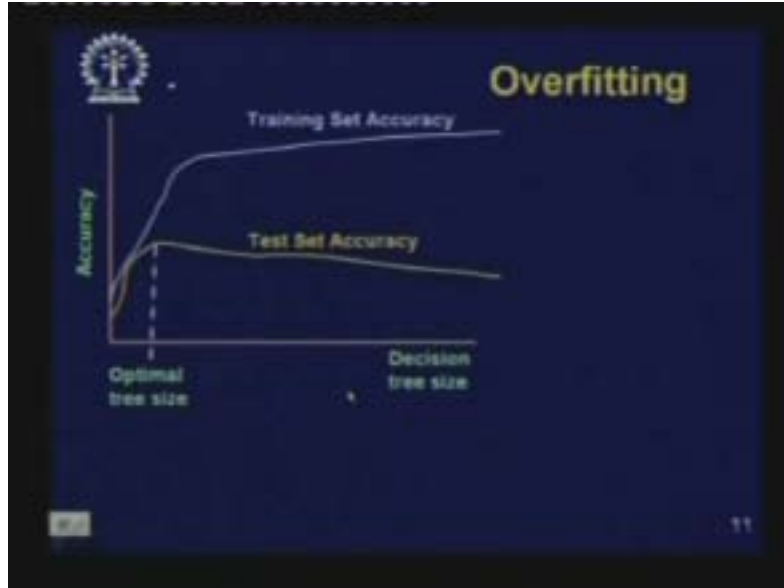
Certain classification problems by themselves are always noisy. If we are not able to look at all the features or if we skip some of the relevant features we might get two data sets which agree with all the features that we have selected but have different classes. This can also happen if some data is noisy. So, in any case there is a situation where no matter what decision tree or what classifier we construct that classifier cannot achieve 100% accuracy on the training set because the training set contains two examples which have identical values of the attributes for different classes. So any classifier that you construct will assign the same class will these two instances which will not be correct.

(Refer Slide Time: 14:07)



Let us review how we would estimate the accuracy of a classifier. As we have mentioned earlier we have some data  $d$  we divide  $d$  into two sets; the training set  $train$  and the test set  $test$ . We train the classifier using the training set and after we have learnt the classifier we apply it on the test set and because the classifier was trained as learned on the train set. It may be likely that the classifier is fit has been over-fitted to the training set. So to get better idea of the true accuracy of the classifier we should look at some unseen examples. And in order to get some examples we keep aside a portion of  $d$  to evaluate the classifier.

(Refer Slide Time: 15:44)



Issue of over-fitting:

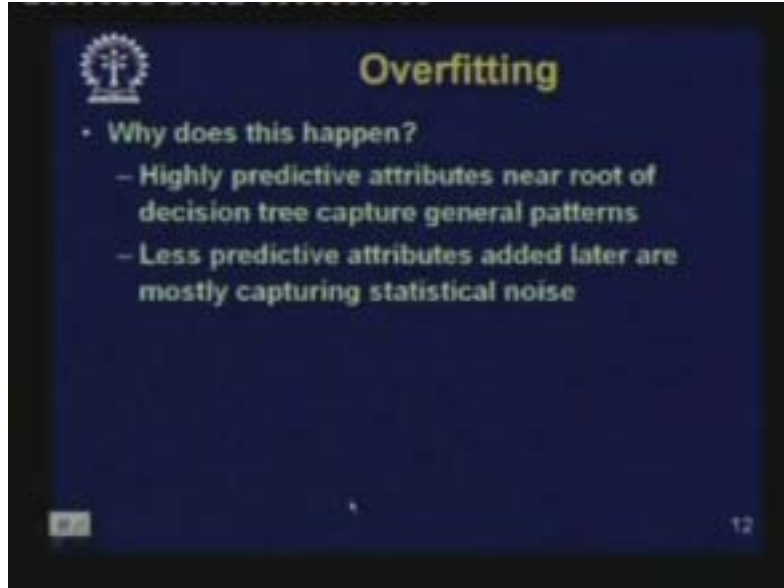
Typically when we learn any classifier what happens is that as we increase the complexity. For example, in decision tree as we increase the decision tree size whereas we grow more nodes the training set accuracy increases. We start with a decision tree having a single node and we gradually add nodes. As we add nodes usually the training set accuracy goes up. However, if we test the accuracy on the test set we see that the test set accuracy initially goes up and then it can slowly come down. And this phenomenon is known as over-fitting. And we notice that based on the test set this is the point where the classifier has highest accuracy on the test set so this is the optimal tree size.

Why does such over-fitting occur?

Such over-fitting can occur because there could be some regularity which is discovered in the data. As we have only a finite amount of data some regularity can be perceived in the data due to some random behavior of the irrelevant attributes. They can mislead us in making us think there is a pattern. And by taking advantage of this pattern our classifier can increase the accuracy of the training set but this pattern may be absent from the test set. Therefore the test set accuracy goes down.



(Refer Slide Time: 17:16)



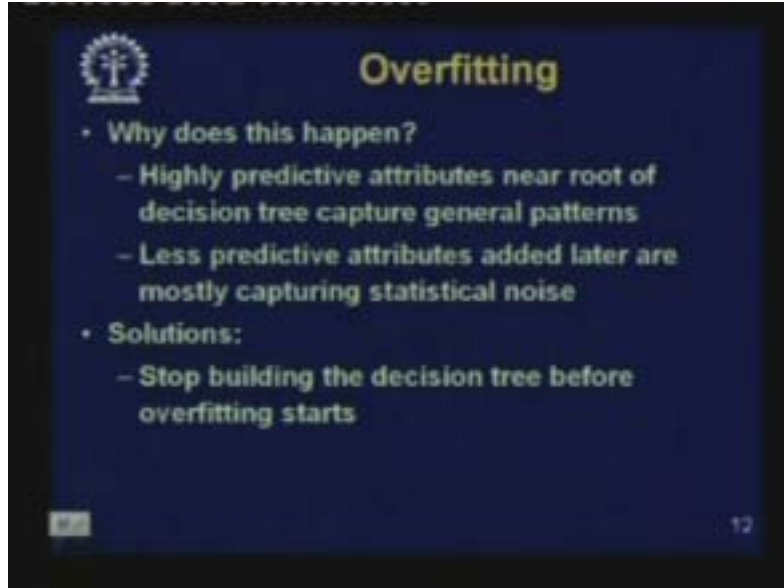
When the attributes are selected initially near the root of the tree the better attributes gets selected immediately. So, the attributes which are highly predictive are already taken care of near the root of the decision tree. After we have looked at the more important attributes then we look for other attributes to grow the tree and then we try to include these attributes which have less productive power and many of them are actually noisy. Therefore the highly predictive attributes occur near the root of the decision tree and they are able to capture the more general patterns. The less predictive attributes added later and they mostly try to capture overfit the tree to statistical noise.

How do we overcome the effect of over-fitting?

There are two major approaches to take care of over-fitting. Firstly we can stop growing the decision tree before over-fitting sort of kicks in and takes over the process. So we have to be careful to know how long to grow the tree.



(Refer Slide Time: 17:16)



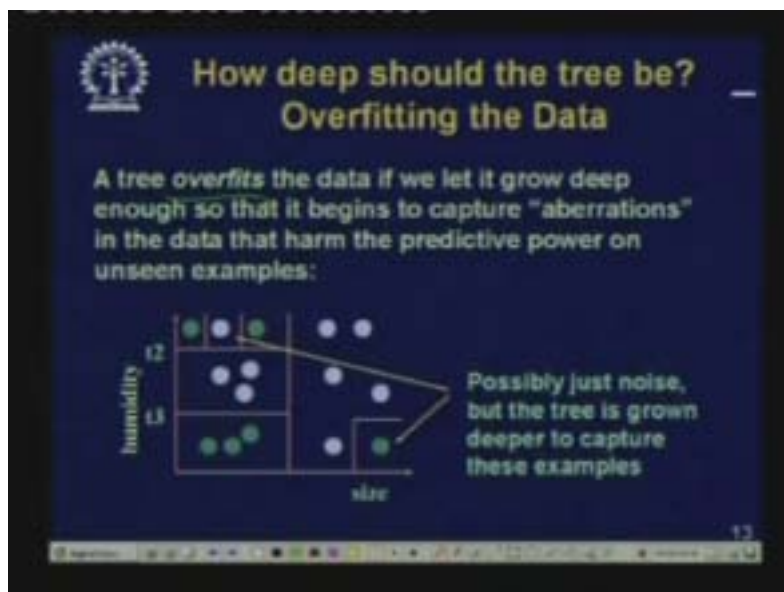
**Overfitting**

- Why does this happen?
  - Highly predictive attributes near root of decision tree capture general patterns
  - Less predictive attributes added later are mostly capturing statistical noise
- Solutions:
  - Stop building the decision tree before overfitting starts

12


We should not grow the tree in all cases till completion as ID3 prescribes rather we might to decide stop growing the tree earlier. The second approach involves pruning. So, we grow the tree quite deep and then we eliminate some lower portions of the tree as a post processing step. Therefore stopping the growth of the tree while growing is one approach and first growing the tree and then pruning is the second approach. Therefore these are the two approaches used to stop the phenomenon of over-fitting.

(Refer Slide Time: 19:30)



**How deep should the tree be?**  
**Overfitting the Data**

A tree overfits the data if we let it grow deep enough so that it begins to capture "aberrations" in the data that harm the predictive power on unseen examples:



Possibly just noise, but the tree is grown deeper to capture these examples

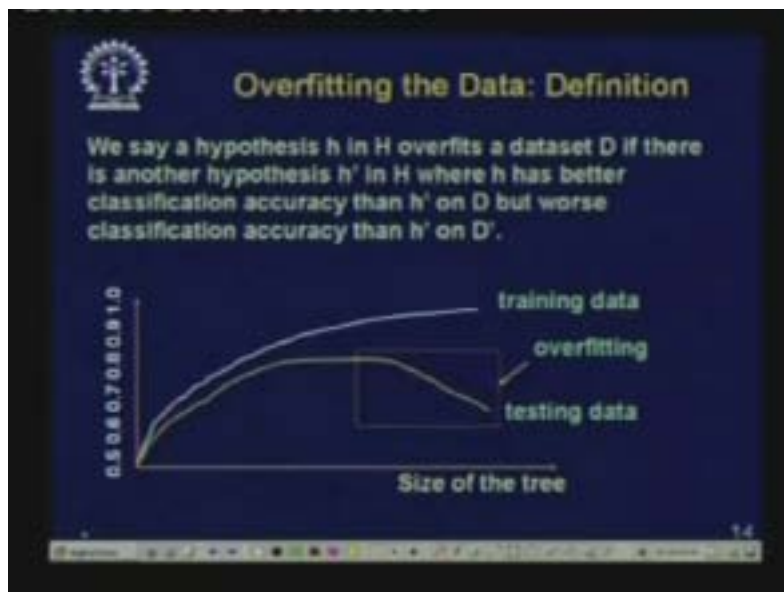
13

How over-fitting can be avoided?

A tree or indeed any classifier is set to overfit the data if we let this tree to grow deep enough so that it captures aberrations in the data. It tries to fit in the aberrations in the data to get perfect fit and this harms the predictive power on unseen examples. For example, suppose there is a feature called humidity and there is another feature called size, now we have two types of classes the blue class and the green class. So what happens is that in the decision tree the decision tree separates the feature space into different classes. In this class there are all blue except this green.

Initially the decision tree gets this node consisting of all the six examples then it further divides the set to separate this lone green class. And it may be that this is over-fitting the data. Possibly this is just noise but the tree is grown deeper to capture this noise. This is another example of data that could be classified as noise. Now the point is that if we treat this whole thing as blue, this thing as green, this as green and this as white may be we will get a tree which has better generalization power which works better with unseen data. But trying to fit all the cases and zero error on the training set may give us a classifier which has lower accuracy on the test set.

(Refer Slide Time: 21:39)



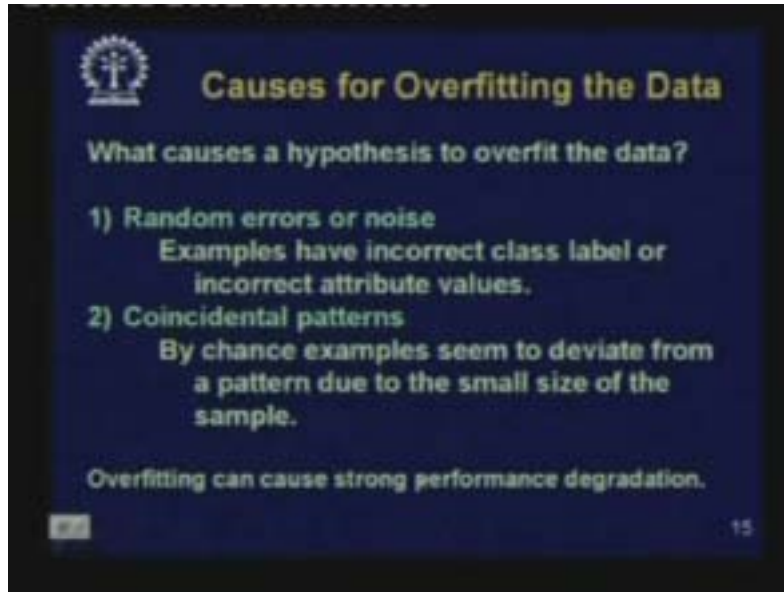
Formal definition of over-fitting:

We say a hypothesis  $h$  in  $H$  over-fits the data set. So  $H$  is the hypothesis space. We are trying to find the hypothesis ball  $h$  which belongs to the hypothesis space. We say the hypothesis  $h$  overfits the data set  $D$ . If there is another hypothesis  $h'$  so  $h$  is the hypothesis that we have obtained and  $h'$  is another hypothesis in the hypothesis space. If there exists an  $h'$  which has worse classification accuracy on the data set but which has better actual classification accuracy, that is  $h$  has better accuracy than  $h'$  on the current training set but  $h$  does not work very well on unseen data then we say that  $h$  has been over-fitted. We can see the phenomenon of over-fitting by inspecting the following curve. This curve is similar to the curve that we looked at earlier. As we have noted we plot

accuracy along the y axis. So this is 0.5, this is 0.6, this 0.7, 0.8, 0.9 and 1. So these are the values of the accuracy.

Now we see that the accuracy on the training set keeps going up as the size of the tree increases. But the accuracy on the test data initially goes up and then it goes down. So this illustrates the phenomenon of over-fitting. So beyond this point the decision tree has tried to overfit the data.

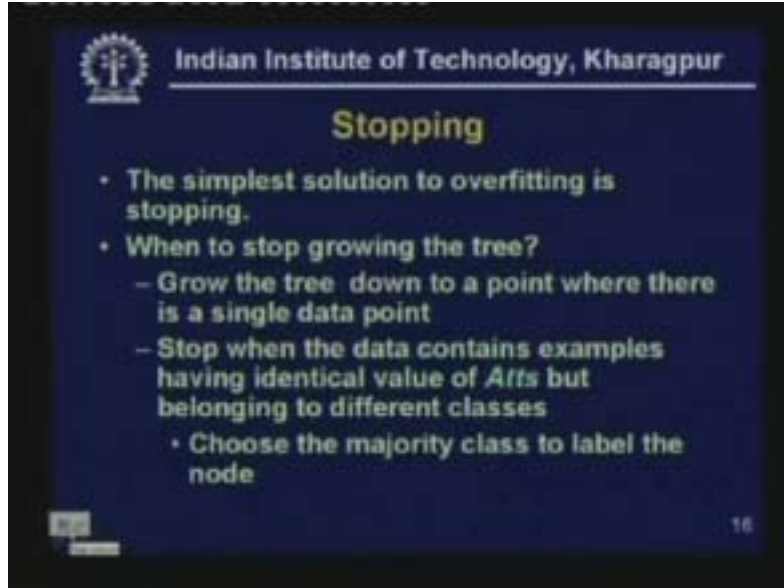
(Refer Slide Time: 23:47)



What causes a hypothesis to overfit the data?

As we have seen it may be due to random errors or noise or it can be due to coincidental patterns. So, random errors or noise happens when examples have incorrect class label or incorrect attribute values due to error in measurement. Secondly, there can be coincidental patterns. By chance the examples may seem to have to have a pattern due to the small size of the training sample. If you take a large training sample much patterns like to be present. But due to this phenomenon of over-fitting there can be strong performance degradation. So we have to deal with the effect of over-fitting for any learning algorithm that we inspect.

(Refer Slide Time: 25:00)



Indian Institute of Technology, Kharagpur

### Stopping

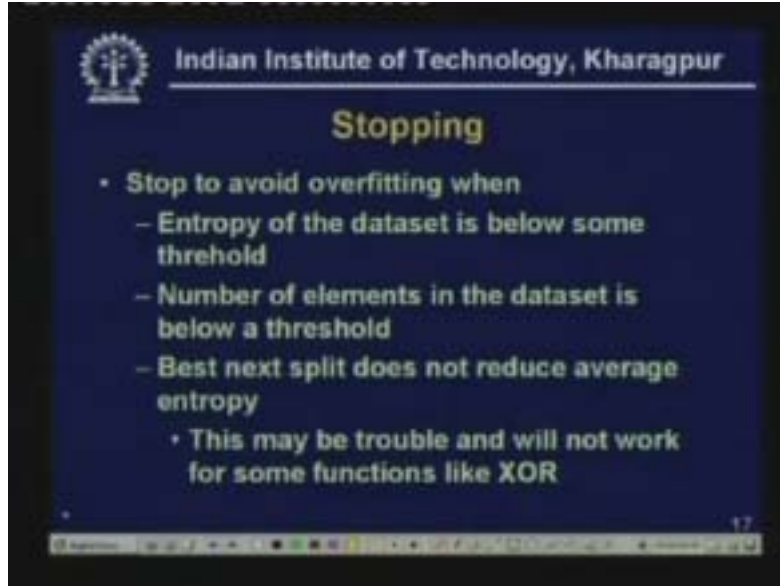
- The simplest solution to overfitting is stopping.
- When to stop growing the tree?
  - Grow the tree down to a point where there is a single data point
  - Stop when the data contains examples having identical value of *Atts* but belonging to different classes
    - Choose the majority class to label the node

16

So, as we mentioned there are two ways of over-fitting; stopping and pruning. The simplest solution to over-fitting is stopping. We want to stop growing the tree before over-fitting takes over. When should we stop growing the tree? There could be several things that we can do. First of all we can grow the tree, we have to use ID3 we can keep growing the tree until we get a single data point. A single data point obviously has a single class and we cannot proceed any further. But instead of doing that we can stop when the data contains examples which has identical values of *atts*. That is, we have some data which agree with the values of all the other attributes but they belong to different classes.

In this case we do not get a homogeneous class and ID3 will fail but we have reached a leaf and we cannot grow this tree further. In those situations what we do we is choose the majority class to label the node. So, in this node we have examples all of which belong to the same features but they have different class. We take the majority class and label it as the class of that node. Now, if you want to handle over-fitting we have to do something different. We have to stop growing the tree even earlier. In order to do that there are several techniques we can employ.

(Refer Slide Time: 26:40)



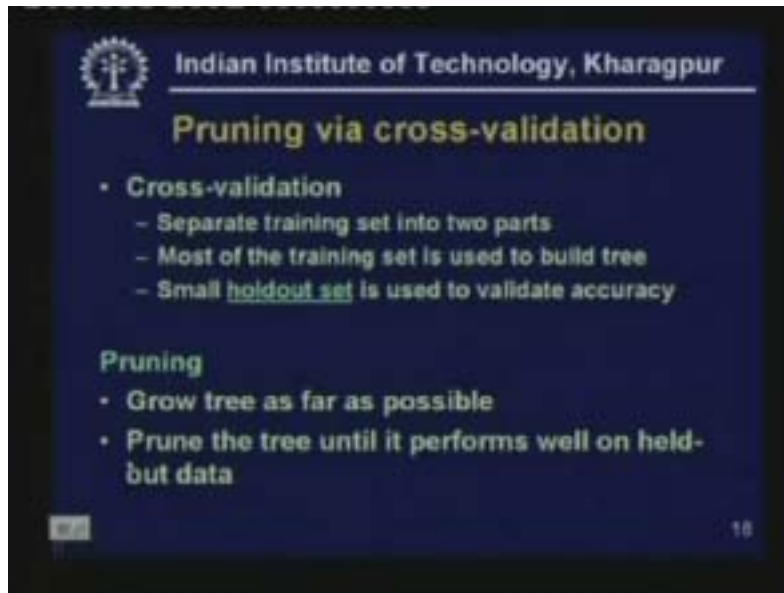
So, when we have a node we mention that we find the entropy of examples at this node and then for a particular attribute we use at the node we look at the children. So  $S$  is the data set here  $S$  has an entropy, entropy of  $S$ . Now due to this attribute  $A$  which takes values true and false we get the data sets  $S_1$  and  $S_2$  and  $S$  is the partitioned between  $S_1$  and  $S_2$ . We find the entropy of  $S_1$ , entropy of  $S_2$  and find the weighted average entropy of  $S_1$  and  $S_2$ . We ideally stop when the entropy of a data set is 0 that is they are all homogeneous. But instead of doing that we can stop when the entropy of the data set is low. So we can select a threshold which is likely more than 0 and we can stop growing the tree when the entropy is smaller than the threshold then we need not consider **this division**.

Secondly, we can stop growing the tree when number of elements in the dataset is below a threshold. If our  $S$  contains only three or four data items we do not want to keep dividing the tree. So only if  $s$  is large then only we consider growing the tree so we stop when  $s$  is small. When  $s$  is small we stop, when entropy is low we stop and thirdly we also stop when the best next plate does not reduce the average entropy. So, this plate does not reduce the average entropy. Therefore we have reached a point where splitting does not help in the short run. So these are three techniques which can be used to stop growing the tree.

Instead of stopping when entropy is zero that is the data set is fully homogeneous we stop when the data set is low entropy and secondly we stop when the size of the data set at that node is small, thirdly we stop when the best plate is not reduce the entropy. However, we can see that the third situation is not always wise. There are certain functions for which if we look at the immediate the gain in information that immediate gain is not there but there can be further gain as we go below the tree.

For example, if we consider the XOR function, suppose you are trying to find a XOR b if you either split a or split b at the top level the entropy does not reduce the entropy remains the same. The entropy only reduces when after a if you test on b then entropy reduces or after b if you test a. after two levels the entropy reduces. So this is not a wise decision in many cases.

(Refer Slide Time: 30:27)



Pruning:

This is the second alternative. For pruning what we do is, when we prune a tree usually the error in the training set will go up. But why do we prune the tree? We prune the tree so that the true error should go down. So, how do we test the true error? We test error on the test set or call the validation set. So we keep aside a validation set we grow the tree on the training set and then prune it and then check it on the validation set. This validation set is called the holdout set. This validation set is slightly different from the test set. The test set is used to test the final tree after pruning.

We have another set called holdout set of validation set which we use for validating the pruning. So we separate the training data into two parts. Most of the training data is used to build the tree and the holdout set or validation set is used to validate the accuracy. For pruning what we do is we grow tree as far as possible. Instead of stopping early we grow the tree quite deep. And then after going the tree we prune the tree until it keeps performing better on the held out data.



(Refer Slide Time: 30:27)

Indian Institute of Technology, Kharagpur

## Pruning via Cross-Validation

- Idea behind pruning
  - Portion of tree that models general patterns works well on holdout set
  - Portion of tree that fits random noise works poorly on holdout set

19

So the idea behind pruning is that the portion of the tree that models general patterns should work well on the holdout set but the portion of the tree that fits noise should not work well on the holdout set.

(Refer Slide Time: 32:31)

Indian Institute of Technology, Kharagpur

## Training and Validation

Dataset D

TRAIN (normally 2/3 of D)

TEST (normally 1/3 of D)

There are two approaches:

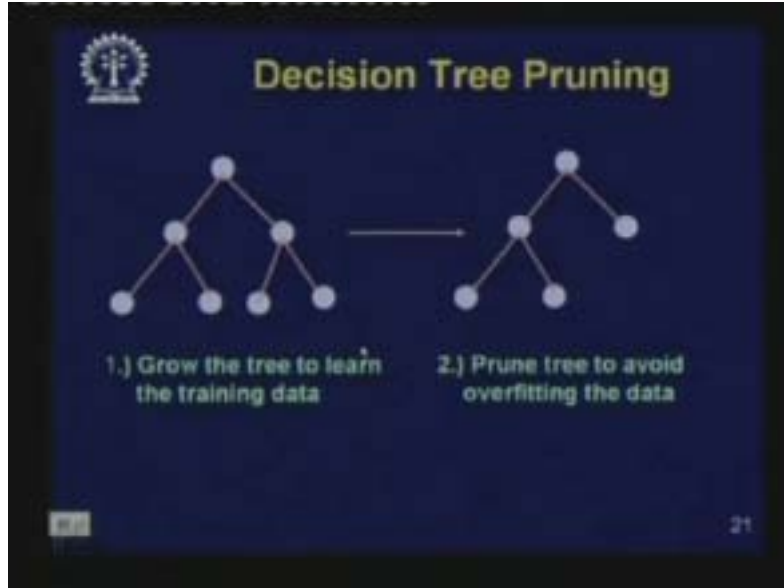
- A. Reduced Error Pruning
- B. Rule Post-Pruning

20

**This is the schematic.** We have data set D we divide it into training and testing or the holdout and we will consider two different approaches to pruning; reduced error pruning and rule post pruning.

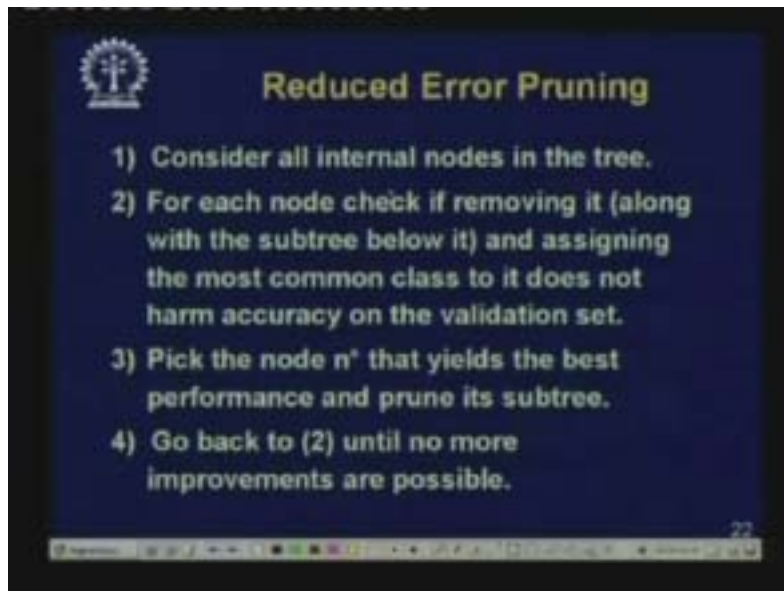


(Refer Slide Time: 32:55)



First we will talk about reduced error pruning. In pruning what we do usually is that we grow the tree to learn the training data and then after growing the tree we remove certain portions of the tree. So we prune certain portion of the tree and so we get this new pruned tree. This is the pruned tree that we get. And the expectation is that the pruned tree will not overfit the data. So reduce the pruning we the techniques for pruning of decision tree.

(Refer Slide Time: 32:55)

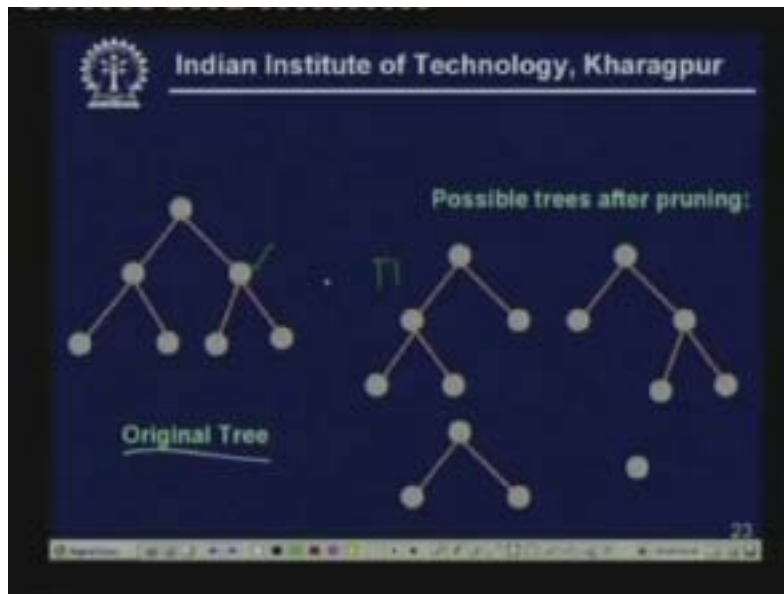


In reduced error pruning we first grow the tree and then we do the following:

We consider all the internal nodes of the tree. For each node we check if removing it gives us any advantage in accuracy. So we consider all internal nodes in the tree. For each internal node we check if removing the node along with the subtree below the node and assigning the most common class to the node whether that improves accuracy on the validation set. So, we find out those intermediate nodes for which there is an improvement in accuracy and let  $n^*$  be the node for which the increase in accuracy is maximum. We prune the tree below  $n^*$  and then we go back to two until we get a situation when none of the nodes can be pruned to better accuracy. So this is the essence of the algorithm reduced error pruning.

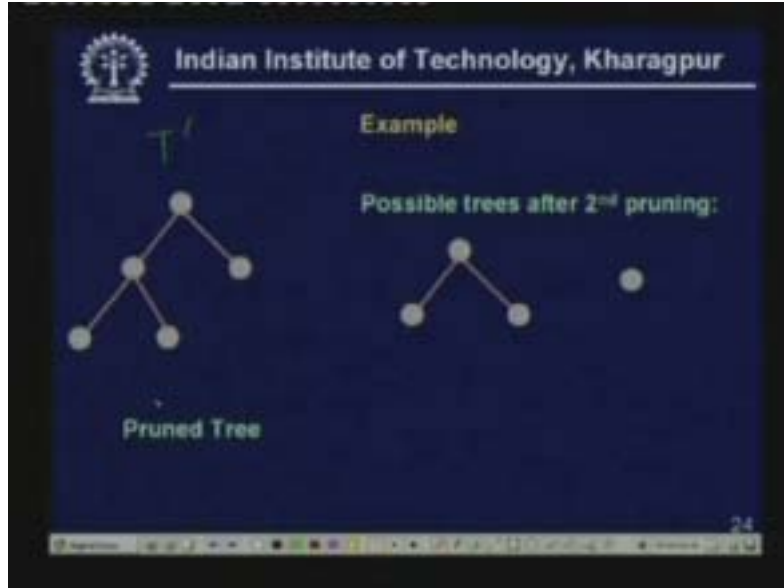
We consider all internal nodes. For each internal node we check if there is improvement accuracy by removing the tree below the node and by replacing it with the majority value. And for all such nodes for which there is improvement we choose the node with highest improvement which is  $n^*$ . If no nodes show improvement we stop, if  $n^*$  shows highest improvement that is the positive improvement then we prune below  $n^*$  and we continue this algorithm. Therefore reduced error pruning can be illustrated by this diagram.

(Refer Slide Time: 35:45)



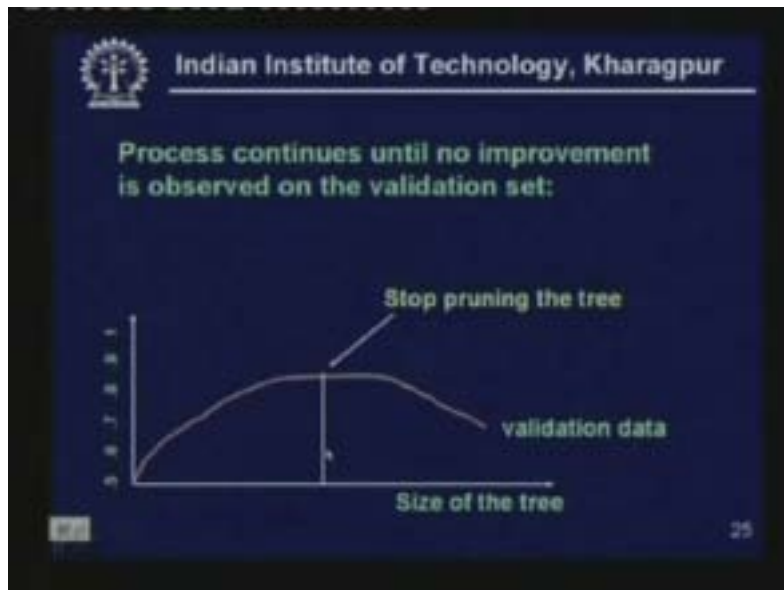
Suppose this is the original tree it has seven nodes and three internal nodes. So what are the possible trees we can get after pruning? If we prune below this node we get this tree  $T_1$ . If we prune below this node we get this tree  $T_2$ , if we prune below this node we get this tree  $T_3$ . Now  $T_4$  involves pruning both below 1 as well as below 2. So we will consider these three trees  $T_1$ ,  $T_2$  and  $T_3$  as possible candidates. We find out whether for these trees the accuracy on the holdout set is better than the accuracy of the original tree. Among them we find the tree which has a highest accuracy. Suppose  $T_2$  has the highest accuracy which is better than the accuracy of  $t$  then we will choose this tree and then we will proceed. Suppose we prune below this node we get this as the prune tree so  $t'$  is the prune tree.

(Refer Slide Time: 36:48)



Now we will again consider pruning this tree further. We can prune below this node to get this tree  $T_1$ , we can prune below this node to get this tree  $T_2$  and evaluate them with accuracy, take the one highest accuracy or if both of them have worst accuracy on the holdout set than  $t'$  then we retain  $t'$ . This is how reduced error pruning proceeds.

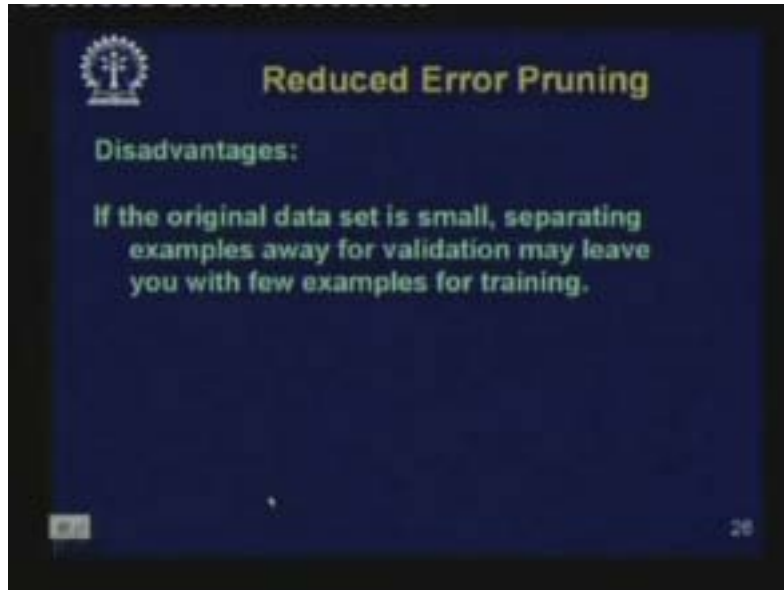
(Refer Slide Time: 37:20)



So, this process continues until there is no improvement in the validation set. So this can be illustrated by this curve, this is the size of the tree and as we reduce the size of the tree

we start with this tree where the accuracy is this as we prune that we see that initially the accuracy is going up and then the accuracy goes down so we stop pruning the tree here.

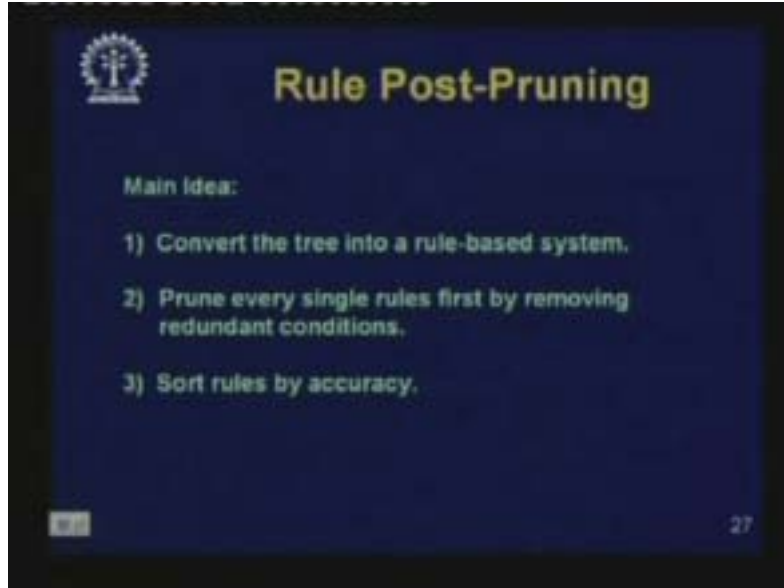
(Refer Slide Time: 38:06)



Disadvantages of reduced error pruning:

Reduced error pruning is a very good technique but it cannot be used if the training data set is small. If the original data set is small if we keep aside something for the validation set the training dataset will be further reduced. So we have seen that cross validation can be used to take care of less amount of training data but in general reduced error in pruning data is not very good when we do not have enough data. Now we will discuss the second pruning method for pruning of decision trees which is called rule post pruning.

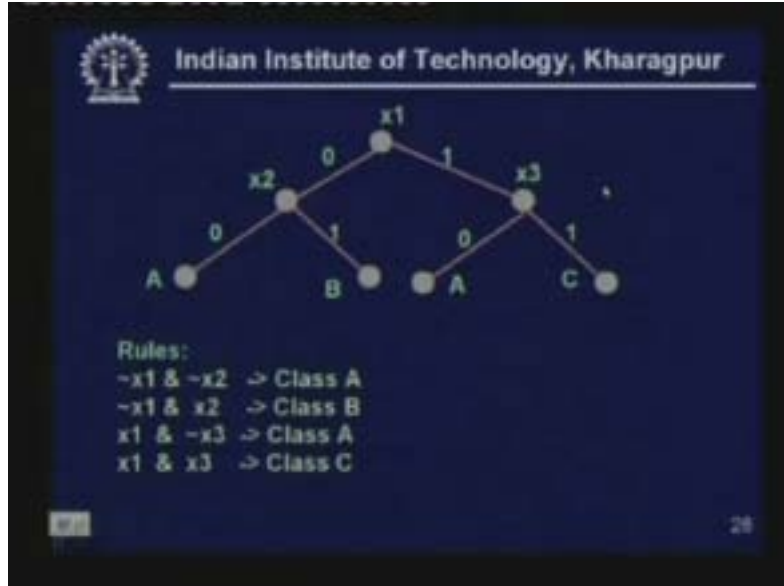
(Refer Slide Time: 38:06)



The basic idea behind rule post pruning is that after you construct the decision tree you dismantle the tree and write down a prevalent set of rules of the decision tree. A decision tree can be expressed as disjunction of all the paths along with a given class  $C$ . So we find that for all the leaves labeled by  $c$  we take the disjunction of all these paths. For each path the formula is the conjunction of all the attributes and the branch that was followed to get to this leaf. We can open out this tree in terms of each such path. So we can convert the decision tree into rules.

We can have rules corresponding to the positive class as well as rules corresponding to the negative class and together we have a set of rules. After we get the set of rules we can prune the rules independently. So what we do is that we look at each rule and see if we can drop some conditions in that and achieve a higher accuracy on the validation set. Therefore we first unroll the tree and we write it in the terms of rules, we sort the rules according to the accuracy and then to use the rules first we will use the most accurate rules and keep using the rules until we get a rule match and then we get a classification. This is the idea of using a rule set on a decision tree. Then we can **prune each of these trees**. So, as an example let us look at this decision tree which has three internal nodes  $x_1$ ,  $x_2$  and  $x_3$  and three classes  $A$  which corresponds to  $x_1$  is equal to 0 and  $x_2$  is equal to 0 or  $x_1$  is equal to 1 and  $x_3$  is equal to 0.

(Refer Slide Time: 40:57)



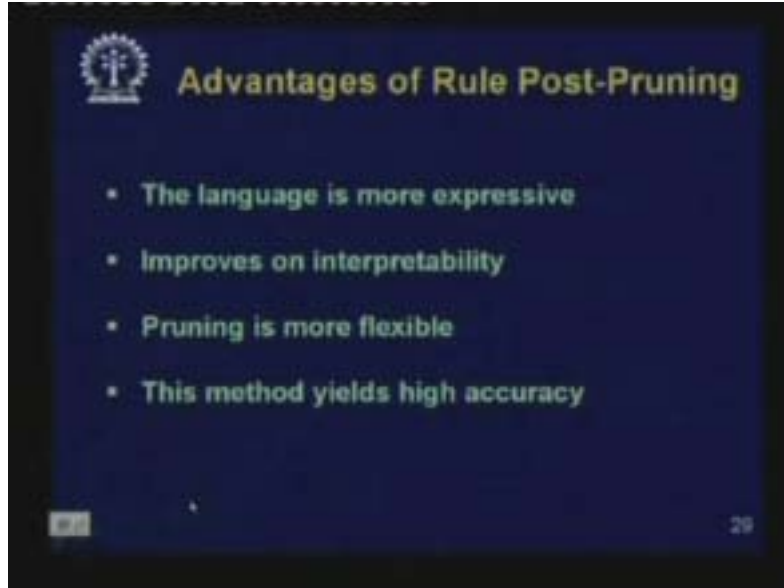
There is the class B which corresponds to  $x_1$  is equal to 0 and  $x_2$  is equal to 1. There is a class C which corresponds to  $x_1$  is equal to 1 and  $x_3$  is equal to 1. So, corresponding to this decision tree we can write out the rules as NOT $x_1$  and NOT $x_2$  implies class A, also NOT $x_1$  and  $x_2$  implies class B,  $x_1$  and NOT $x_3$  is class A,  $x_1$  and  $x_3$  is class C. So we get this set of rules.

Now, at these four rules we try to see we can drop conditions from these rules and get a more accurate classifier. For example, we can drop the condition NOT $x_2$  from this set so NOT $x_1$  implies class A, we have not pruned NOT $x_1$  and  $x_2$  so we prune this rule and we prune this rule so we drop the condition  $x_1$  from this rule and we drop this condition NOT $x_2$  rule so we get this new set of rules which is smaller than the previous set of rules and we evaluate the accuracy of this old rule set on the holdout set. And if this rule set has higher accuracy we adopt the rule set. So we drop conditions to the rule so that the resulting rule set has higher accuracy on the holdout set.

Can you spot the essential difference between reduced error pruning and rule possible pruning?

In reduced error pruning we only prune the bottom portion. When you prune a node we prune everything below that node. If you prune an intermediate node it can be shared by different paths. So when we prune the node we prune it along all the parts. When we write out a decision tree in the form of a number of rules and we look at each rule individually we can prune a variable. For example, we are pruning the variable  $x_1$  for this rule so we are dropping  $x_1$  from this rule. But we are not dropping  $x_1$  from the other rule. So we can independently drop conditions on different rules. And we can drop a condition at the top of a tree and not drop a condition at the bottom. In reduced error pruning when we drop a node we drop everything below that node.

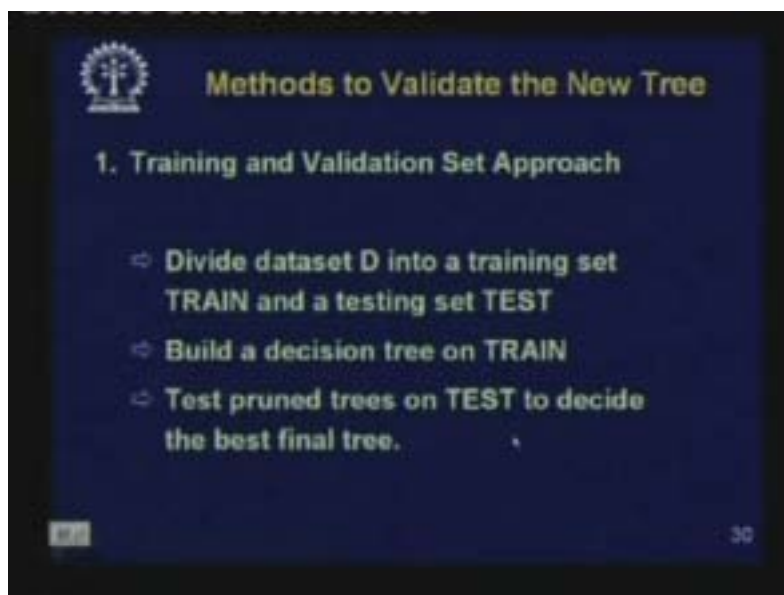
(Refer Slide Time: 44:19)



What are the advantages of rule post pruning?

The language is more expressive than a decision tree. When we get an arbitrary set of rules they cannot always be efficiently expressed using a decision tree. A decision tree can express a disjunction of rules. They can be expressed but not very efficiently. So when we get the set of rules we cannot reconstruct the decision tree back from the rules, we want to use these rules. Therefore this language is very expressive, rules are easy to interpret, pruning as we just noted is more flexible than we use rule pruning. And finally in practical application this method has been seen to have yielded high accuracy.

(Refer Slide Time: 45:16)





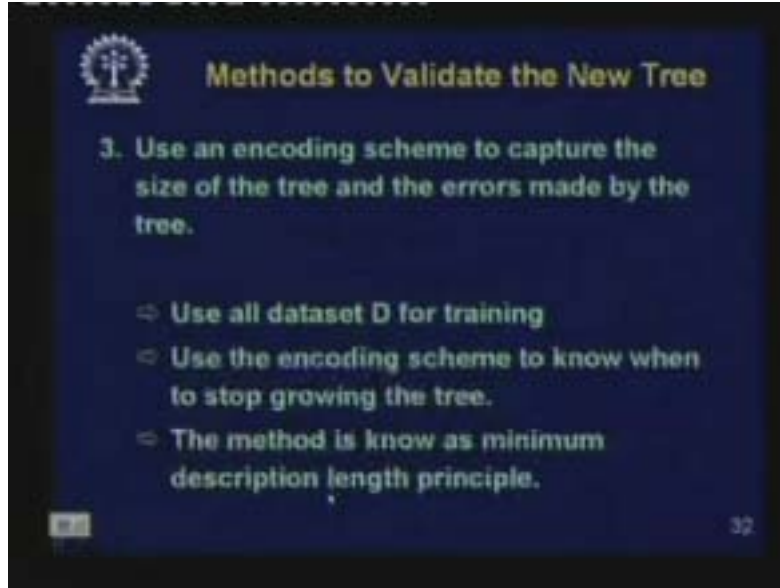
Now let us look back at the methods of validating the new tree that we get. So, the methods we have looked at so far are by using a validation set, the training set and validation set approach. So the first method that we have been following involves dividing the dataset  $D$  into two sets train and test. We build a decision tree using train and we test the prune tree using test. The second method does not use a validation set. So this method can be used even when you do not have enough data to keep aside a validation set.

(Refer Slide Time: 46:06)

The slide features a dark blue background with a white logo in the top left corner. The title 'Methods to Validate the New Tree' is written in yellow at the top. Below the title, the text '2. Use a statistical test' is displayed in white. Two bullet points follow: 'Use all dataset  $D$  for training' and 'Use a statistical test to decide if you should expand the node or not (e.g., chi squared)'. A simple tree diagram with three nodes is shown below the text, with an arrow pointing to the right child node and the text 'Should I expand or not?'. The slide number '31' is in the bottom right corner.

This second method uses a statistical test like the chi square test. The chi square test is an example, a statistical method people use than other tests. So, in this method we use all the data sets for training. Then we use a statistical test to decide whether we should expand the node or not. Suppose you have grown this tree and then you consider the tree statically in finding whether there is any benefit in expanding this node.

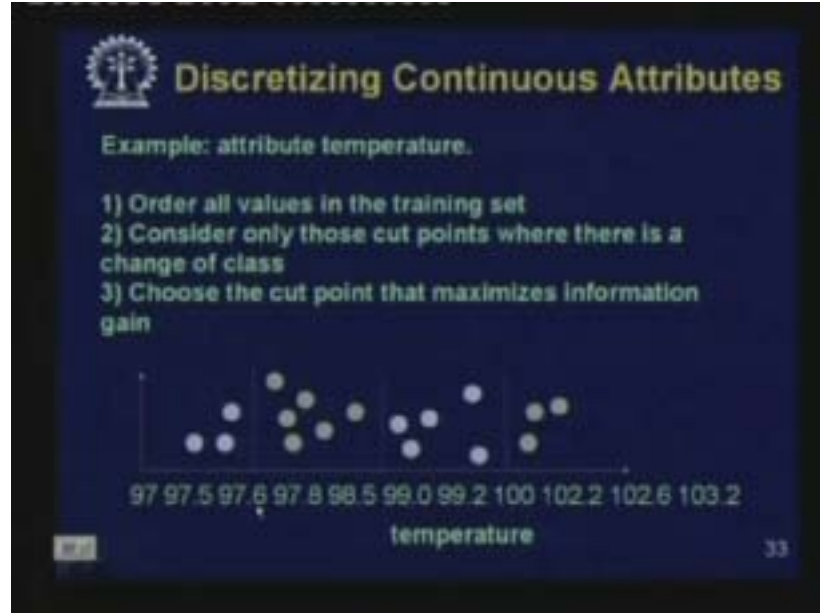
(Refer Slide Time: 47:04)



The third method is slightly different. It uses an encoding scheme to capture the size of the tree and the errors made by the tree. So we use something like the minimum description length principle. So the basic idea is that we use all the dataset  $D$  to construct the tree and we use the encoding scheme to know when stop growing the tree. Therefore this method is known as the mdl principle.

In the mdl principle usually what we do is, we say that the best tree we want is the tree where the size of the tree in bits plus the size to represent the misclassified examples. So the sum of these two sizes should be minimum. Suppose you have a tree  $t$  which is larger in size suppose it contains 20 nodes and it misclassifies four examples four misclassifications. Suppose  $t'$  has 6 nodes and it has six misclassifications so which one would you prefer? In order to compare these two schemes the mdl principle is used. You find out for any of these schemes the number of bits you need to represent both the tree as well as the **misclassified** examples. And of the different alternatives you have you select the classifier for which this is minimum. So this is the mdl principle which is also inspired by the occam's razor principle.

(Refer Slide Time: 47:04)



Additional issues concerning decision trees:

So far we looked at attributes which have either Boolean values or fixed values. Now what to do when we have attributes that take continuous values. For example, suppose you take the attribute temperature. So temperature can take different values. Now, what we do in this case is that we try to discretize continuous attributes and in order to do that we select a split. Suppose we say temperature greater than 30, temperature less than equal to 30 so we split the values of temperature into two sets. Therefore, in order to find out a good split what we do is we order all the values of temperature in the training set.

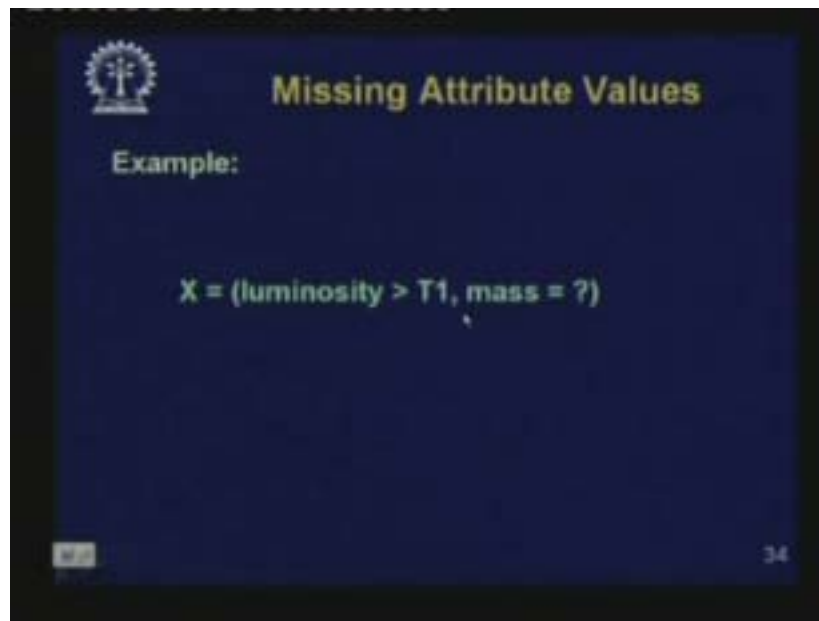
We have ordered all the values and we find out for each value whether we have positive classes or negative classes. So we find that here there is a positive class, positive class and a positive class and then these are all negative examples, these are all positive examples, these are all negative examples. So these examples are negative whereas these examples are positive and these examples are positive. So, when we decide to discretize the attribute we select the possible points where we wish to discretize the values. Hence we only consider those cut points where there is a change of class. We choose the cut point that maximizes this information gain.

To review, what we do is, we order the data set in terms of temperature values. And for each instance we note the classification against the temperature value. Now, when we discretize the attribute we cut it on one point. At this point if we cut this we find the information gain with respect to this cut. So, temperature less than 99.0 will be on this side,  $t$  less than 99 is on this side and  $t$  greater than equal to 99 is on this side. Therefore this amounts to getting a Boolean feature and with respect to this Boolean feature we can find the information gain. So we consider cutting at different points. We consider cutting at these points  $t$  less than 97.6,  $t$  greater than 99.6,  $t$  less than 100,  $t$  greater than 100. So

we can consider cutting at each of these points for which the gain in information is maximum.

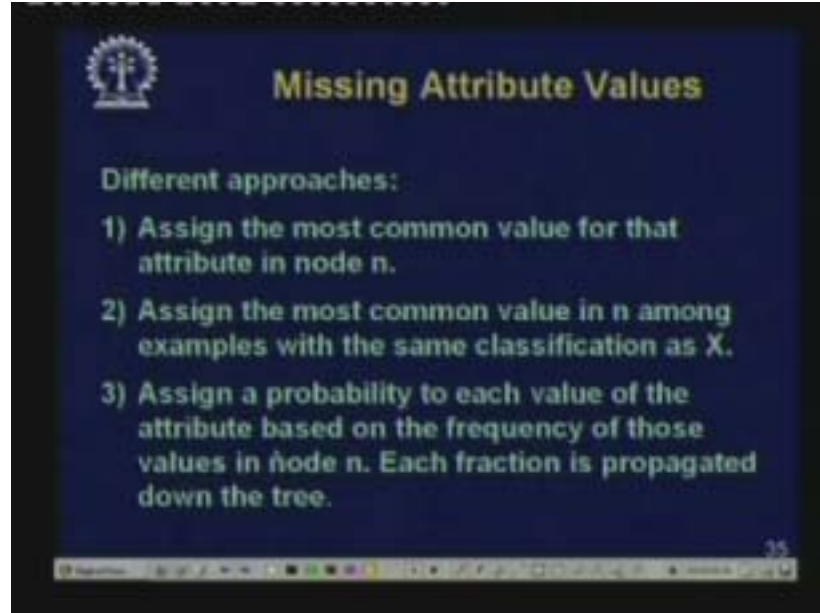
Secondly, we wish to handle cases where attribute values are missing from the data set. For example, we may have a data set for which the value of mass is missing.

(Refer Slide Time: 52:49)



We have the other feature but the value of one of the features is missing, what we do in this case? There are several things we can do.

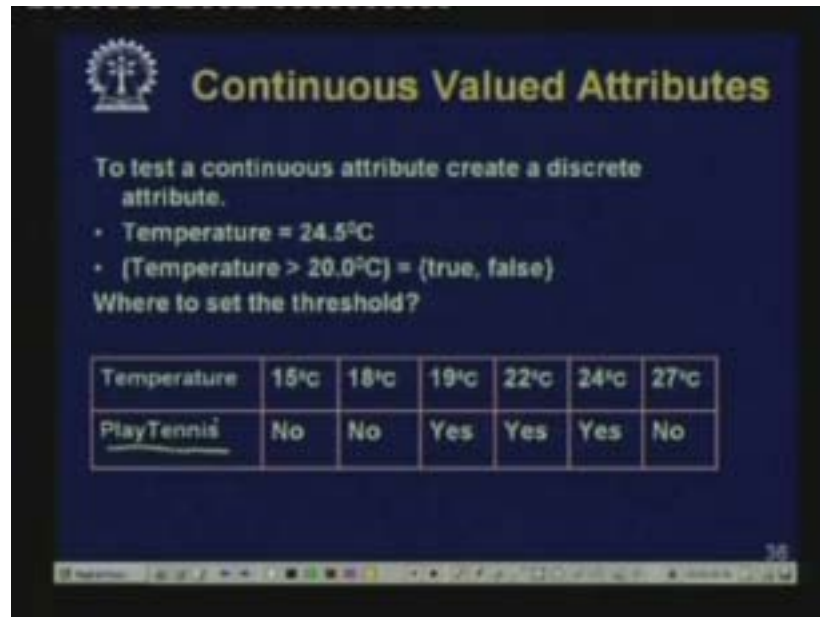
(Refer Slide Time: 53:06)



One possibility is to assign the most common value for that attribute in this node. Suppose Mars is missing we look at the other examples, we find out what is the most common value of Mars in the other examples we put that value in this example. This is one strategy. The second strategy is we look at only those examples which have the same class as the current example. Suppose the example is  $n$  it has a class  $x$  we find all other examples which belong to class  $x$ . For those examples we find which value of Mars is most common we use that value of Mars in this example, this is the second strategy.

The third possible strategy is; we assign a probability to each value of the attribute. Suppose Mars can take three different values  $m_1$ ,  $m_2$  and  $m_3$ . So, for each of these values of Mars we assign a probability and this probability is based on the frequency of those values you are going to take. Therefore each fraction is propagated down the tree. Suppose we find in the examples that Mars is the value of  $m_1$  with probability 0.7,  $m_2$  with probability 0.2,  $m_3$  with probability 0.1 we assign these values with this probability to the node  $n$  and we use that in the decision tree algorithm. This is another example of a continuous attribute. We have temperature, we have the class that we are trying to learn and it is the play tennis class, we have ordered temperature, sorted temperature and ascending order 15 to 18 degrees, 19 degrees, 20 degrees, 24 degrees, 27 degrees and for each of these instances we have written down the class. We see that these are the points where the temperature value changes.

(Refer Slide Time: 55:03)



**Continuous Valued Attributes**

To test a continuous attribute create a discrete attribute.

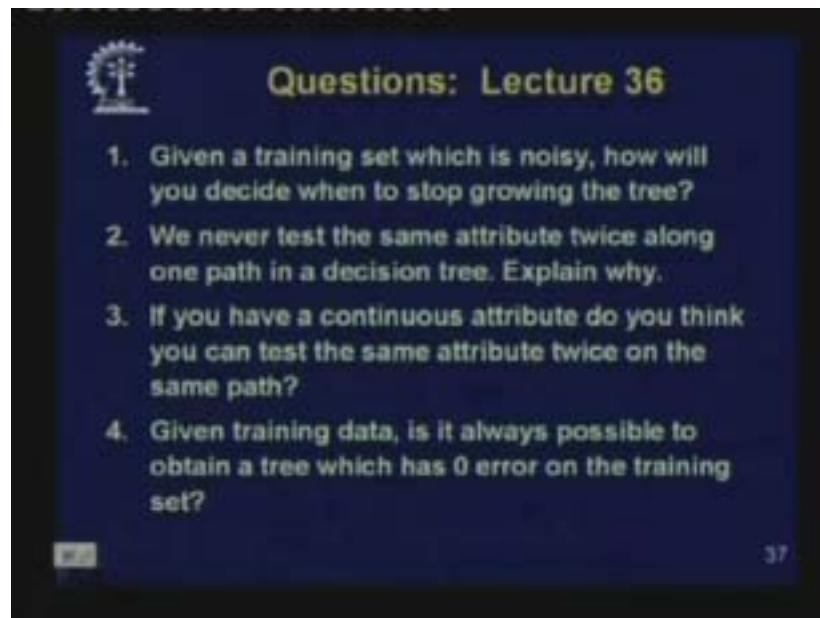
- Temperature = 24.5°C
- (Temperature > 20.0°C) = {true, false}

Where to set the threshold?

Temperature	15°C	18°C	19°C	22°C	24°C	27°C
PlayTennis	No	No	Yes	Yes	Yes	No

So we will be taking one of these as the cut point. For example, if we take this as the cut point then we will test if temperature is less than 18.5 degrees, if we cut at this point we will check if the temperature is less than 24.5 degrees centigrade. For each of these we will find the information gain and select the one with the higher information gain.

(Refer Slide Time: 56:06)



**Questions: Lecture 36**

1. Given a training set which is noisy, how will you decide when to stop growing the tree?
2. We never test the same attribute twice along one path in a decision tree. Explain why.
3. If you have a continuous attribute do you think you can test the same attribute twice on the same path?
4. Given training data, is it always possible to obtain a tree which has 0 error on the training set?

Questions:

1) Given a training set which is noisy how will you decide when to stop growing the tree?

- 2) We never test the same attribute twice along one path in a decision tree, explain why?
- 3) If you have a continuous attribute do you think you can test the same attribute twice on the same path.
- 4) Given training data is it always possible to obtain a tree which has zero error on the training set?