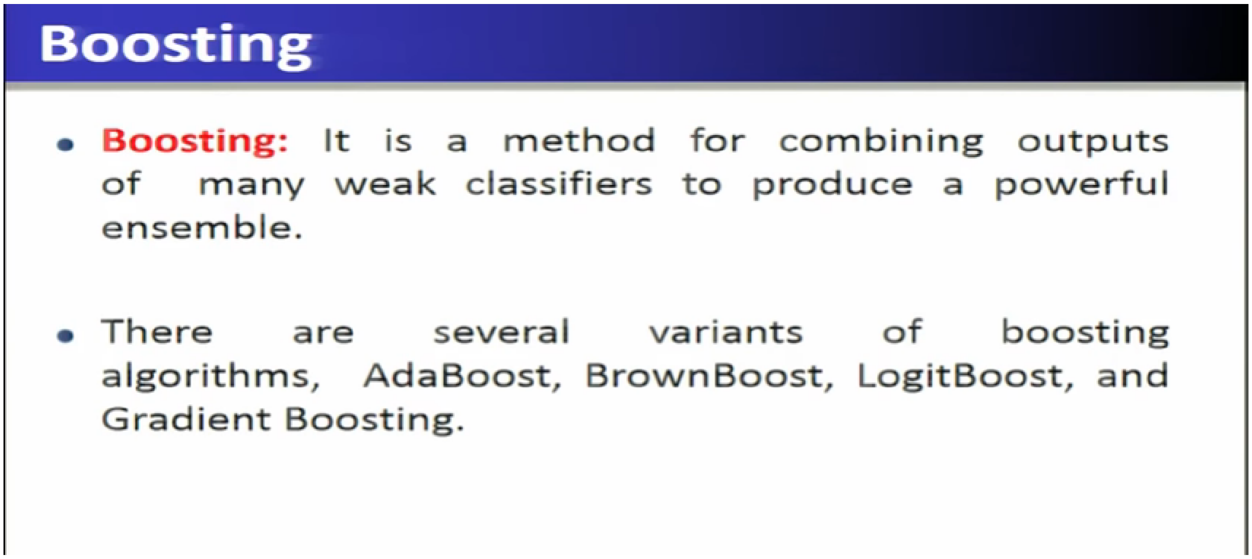
Lecture - 29
Big Data Predictive Analytics
(Part-II)

Gradient boosted decision trees for regression.

Refer Slide Time :(0:18)

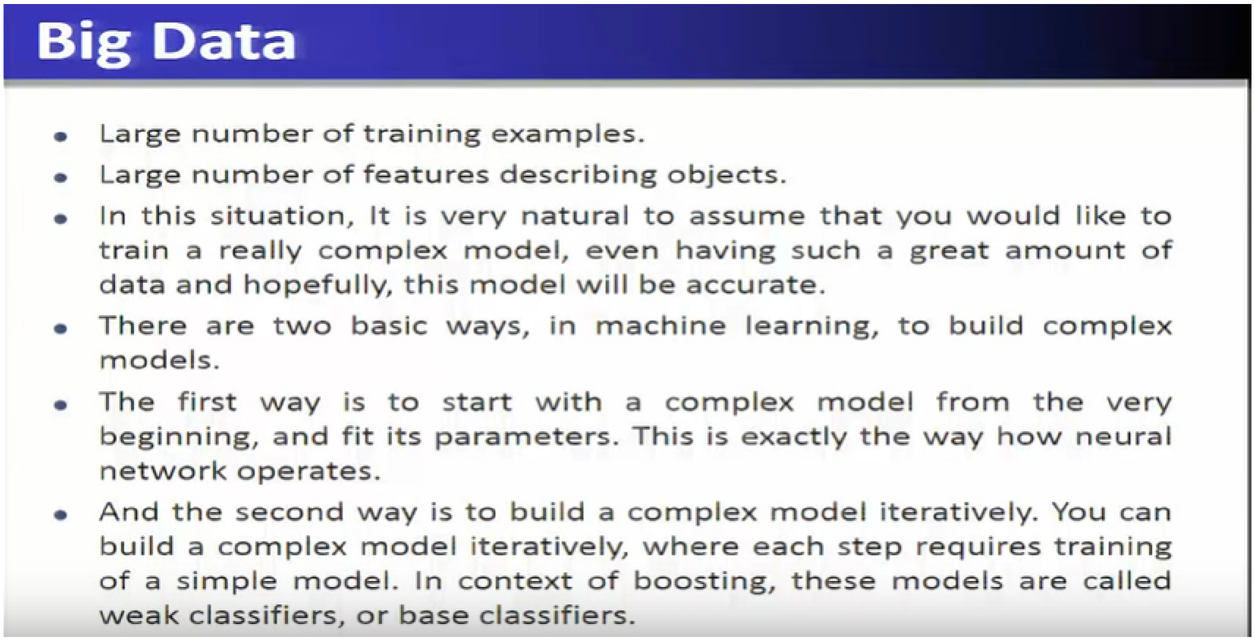


Boosting

- **Boosting:** It is a method for combining outputs of many weak classifiers to produce a powerful ensemble.
- There are several variants of boosting algorithms, AdaBoost, BrownBoost, LogitBoost, and Gradient Boosting.

Boosting: Boosting it is a method of combining outputs of many week classifiers, to produce a powerful and ensemble. There are several variants of boosting algorithms, AdaBoost, Brown Boost, Logic Boost and Gradient Boosting.

Refer Slide Time :(0:37)

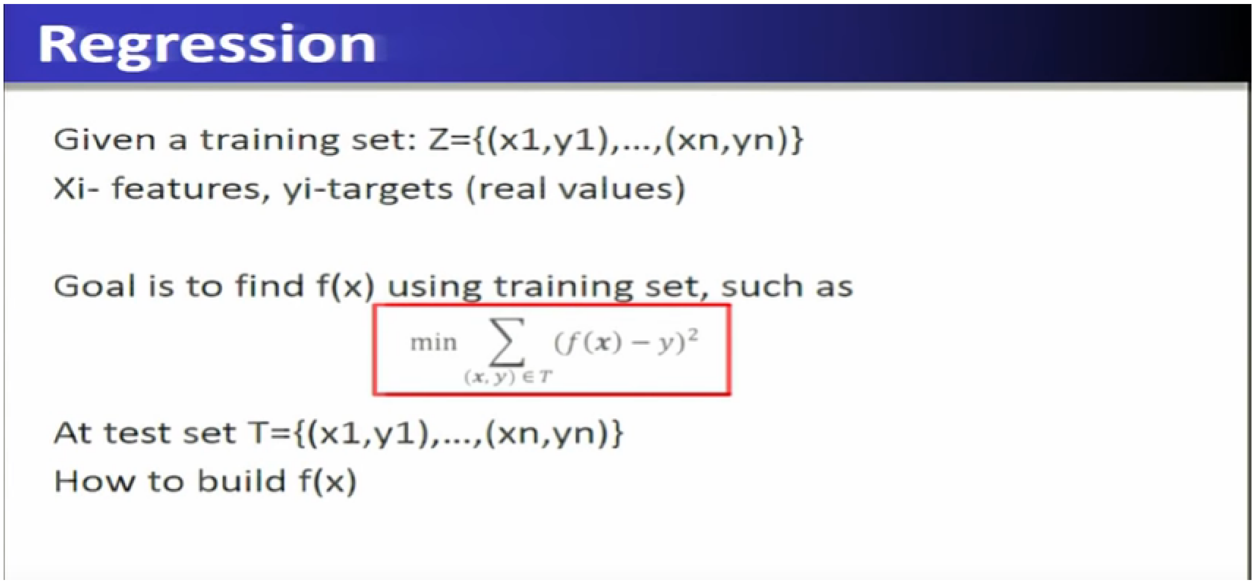


Big Data

- Large number of training examples.
- Large number of features describing objects.
- In this situation, It is very natural to assume that you would like to train a really complex model, even having such a great amount of data and hopefully, this model will be accurate.
- There are two basic ways, in machine learning, to build complex models.
- The first way is to start with a complex model from the very beginning, and fit its parameters. This is exactly the way how neural network operates.
- And the second way is to build a complex model iteratively. You can build a complex model iteratively, where each step requires training of a simple model. In context of boosting, these models are called weak classifiers, or base classifiers.

Now, we see in the big data that, the size of the training examples are a very large. And also, a large number of, large number of features describing the objects are present. So therefore, this one kind of scenario occurs in a situation of a big data. So, in this situation it is very natural to assume that, you would like to train a really complex model, even having such a great amount of data and hopefully, this model will be accurate. There are two basic ways, in the machine learning to build the complex models. The first one is that, you have to start with a very complex model from the very beginning, and fits its parameters, with the data. So, this is exactly the way how the neural networks operates, so they operates with the, with the very complex model, in the beginning itself and they will fit the data, they will fit the parameters and the data will be now, then predicted based on that fitting of its parameters. Now, the second way is to build a complex model iteratively. So, we can build a complex model iteratively that is with each step requires, the training of a simple model, so in the context of good boosting, these models are called, 'Weak Classifiers or the Base Classifiers. Therefore, it is an ensemble of weak classifiers and which makes, this particular model iteratively.

Refer Slide Time :(2:20)



Regression

Given a training set: Z={(x1,y1),...,(xn,yn)}
Xi- features, yi-targets (real values)

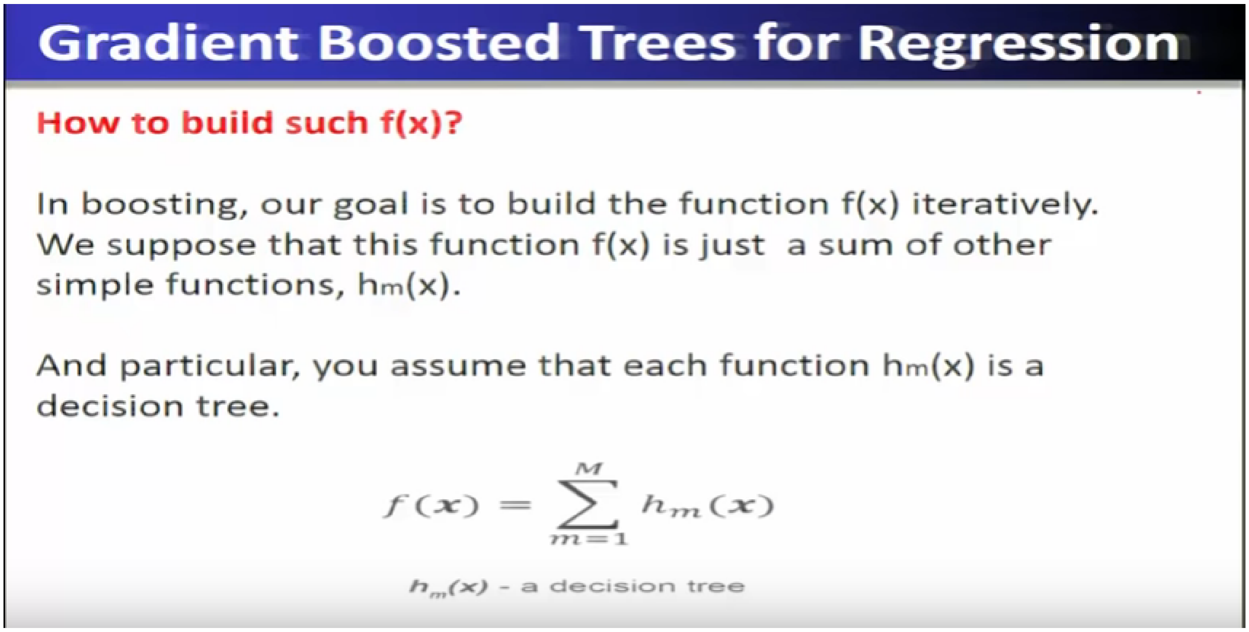
Goal is to find f(x) using training set, such as

$$\boxed{\min \sum_{(x,y)\in T} (f(\boldsymbol{x}) - y)^2}$$

At test set T={(x1,y1),...,(xn,yn)}
How to build f(x)

Let us see, the regression. How this particular concept can be applied in the regression, in a gradient boosted our decision trees. So, let us assume that there are samples which are given which are specified by Z, where x1 is the set of features, in the data set and y1 is the label and x1 to xn are n different samples, in the training data set. Now, the goal here is to find n FX, using this training set such that, this particular minimum of FX minus y Square and the summation of this is, the minimum such error will be there and so at the test set, given the test set this error should be the minimum one, here in this case. How to build this FX is a question?

Refer Slide Time :(3:28)



Gradient Boosted Trees for Regression

How to build such f(x)?

In boosting, our goal is to build the function f(x) iteratively. We suppose that this function f(x) is just a sum of other simple functions, hm(x).

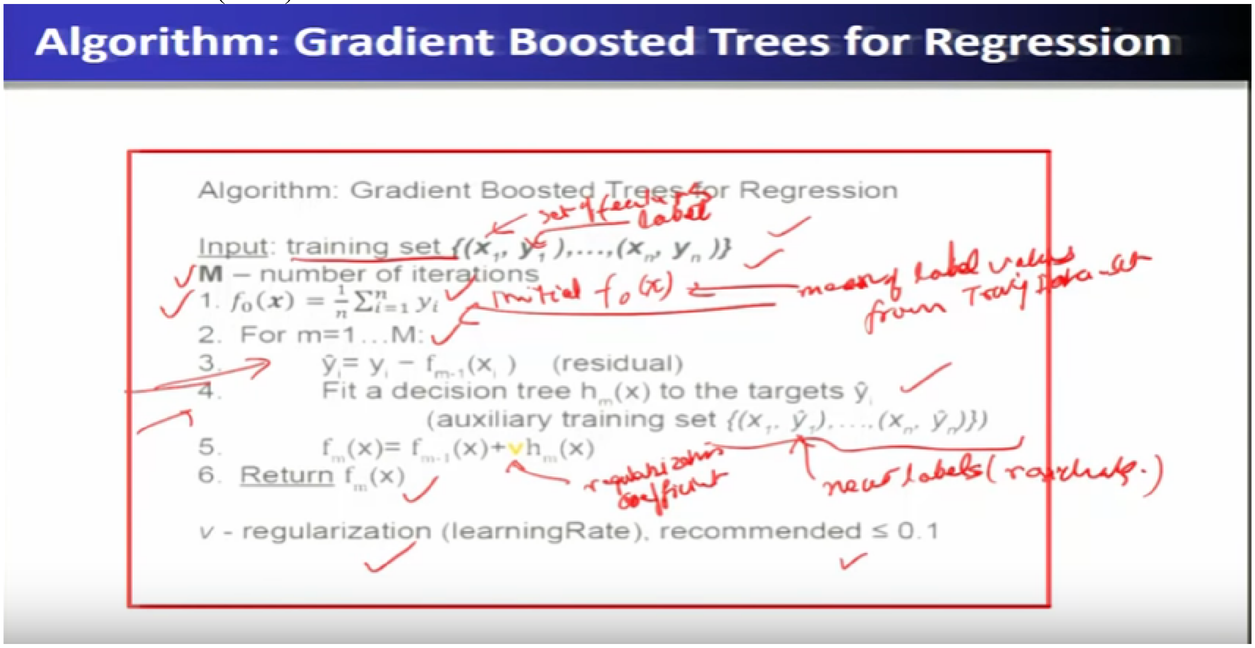
And particular, you assume that each function hm(x) is a decision tree.

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} h_m(\boldsymbol{x})$$

$h_m(\boldsymbol{x})$ - a decision tree

So, how to build such an FX? And boosting our goal is to find is to build, the function FX iteratively. So, we suppose that, this function FX is just a sum of other simple functions, hm of X. So, in particular you have a let us assume that, each function hm is a decision tree. So here, each function is a decision tree. And the, the aggregation of it is basically the FX function.

Refer Slide Time :(4:00)



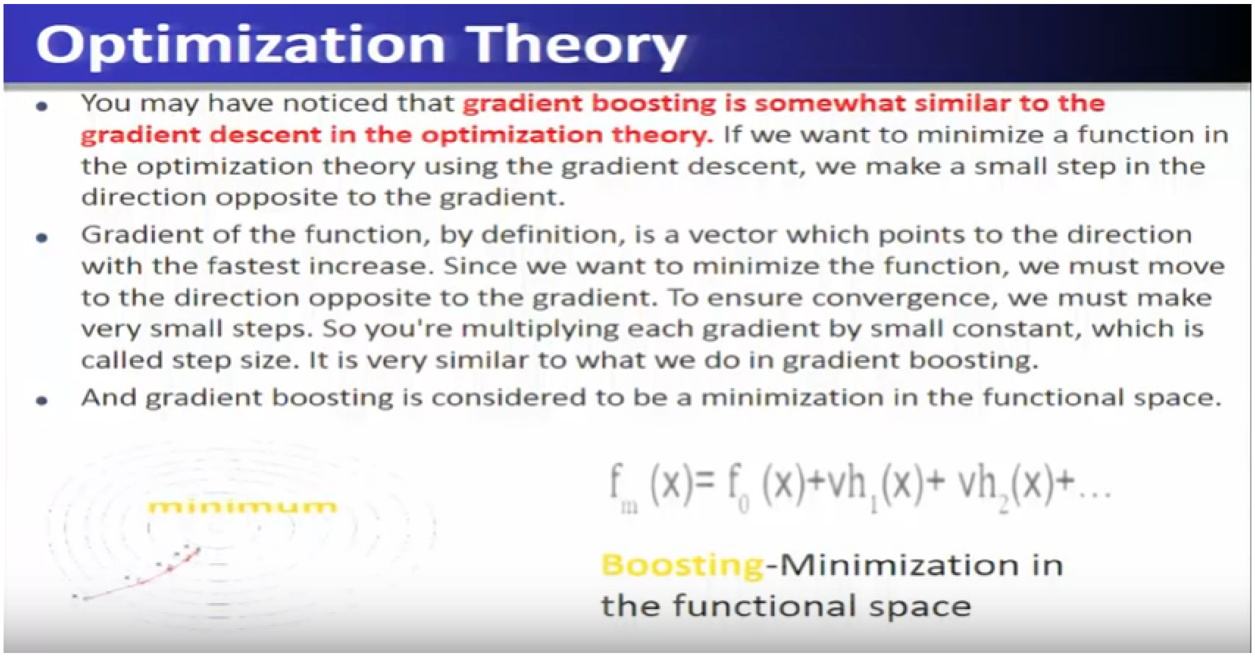
Algorithm: Gradient Boosted Trees for Regression

```
Algorithm: Gradient Boosted Trees for Regression

Input: training set {(x_1, y_1),...,(x_n, y_n)}
M – number of iterations
1. $f_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$
2. For m=1...M:
3.        $\hat{y}_i = y_i - f_{m-1}(x_i)$    (residual)
4.        Fit a decision tree $h_m(x)$ to the targets $\hat{y}_i$
               (auxiliary training set $\{(x_1, \hat{y}_1),....,(x_n, \hat{y}_n)\}$)
5.        $f_m(x) = f_{m-1}(x) + v h_m(x)$
6. Return $f_m(x)$

v - regularization (learningRate), recommended ≤ 0.1
```

set of feature
label
Initial $f_0(x)$
mean of label values from Train data set
regularization coefficient
new labels (residuals.)

Let us understand: The gradient boosted decision trees for regression in more details. So, gradient boosted trees for regression problem. So, let us take the input set of n different data that is training data set, which consists of X is the set of features and y:i consists of the labels, so in turn this is the training data set of n different samples which are given, as an input to the gradient boosted trees for regression. And M is the number of iterations and now, the step number one assumes the initial calculate the initial, f0x. So that will be calculated by summing up all the label values and their average or their mean is assigned over here. So, it will take the mean of, of the label values from the training data set and that becomes F 0 or the initial F 0 function, initial effects function. Now, then it will iterate for M iterations, wherein it will calculate the residual, which is Y I cap is nothing but, Y I - this residual of M minus 1 that is of a previous iteration and so here, it will be 0 residual and so residuals are nothing but, the differences of values in the actually labels and the predicted labels that is called the, 'Residual'. Now, this after that, the next step would be to fit a decision tree hm, to the targets that is Y I cap. And then, it that means, it will generate an auxilary training set, out of this particular set wherein, Y I cap will be replaced here as the labels, as the new labels which are nothing but the residuals. And then, F M, F X will be calculated by giving up particular regularization, coefficient and therefore, it will calculate the value of FM and in this manner, it will iterate and compute all that things. Now, as far as the regularization, coefficient that is nothing but I mean, it is recommended to be less than 0.1 here in this case, this is going to be important parameter, in this gradient boosted trees for regression.

Refer Slide Time :(7:37)



Optimization Theory

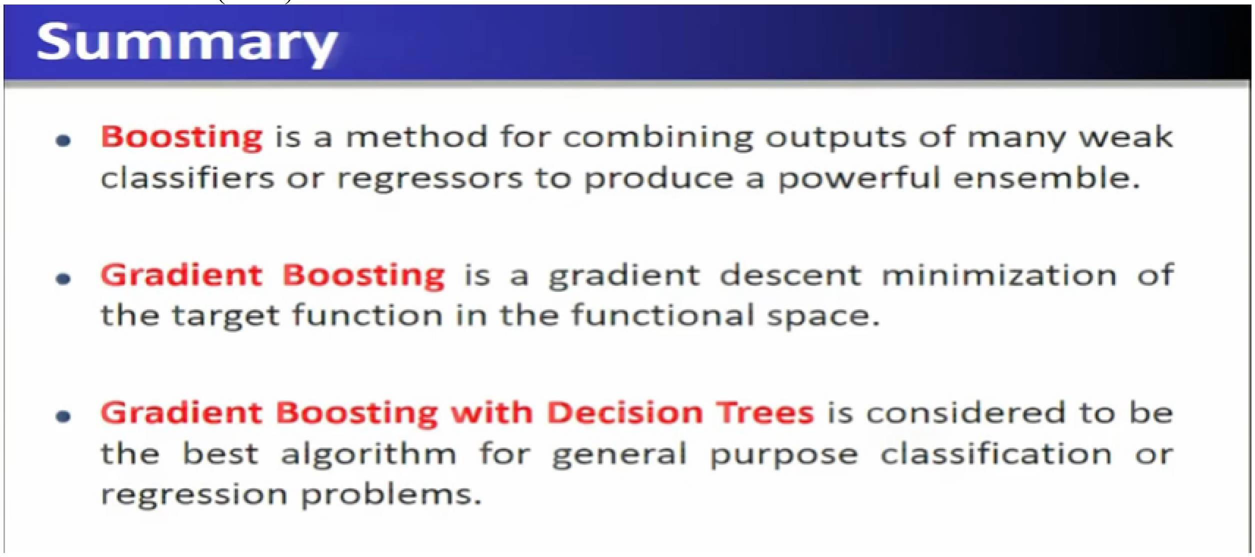
- You may have noticed that **gradient boosting is somewhat similar to the gradient descent in the optimization theory.** If we want to minimize a function in the optimization theory using the gradient descent, we make a small step in the direction opposite to the gradient.
- Gradient of the function, by definition, is a vector which points to the direction with the fastest increase. Since we want to minimize the function, we must move to the direction opposite to the gradient. To ensure convergence, we must make very small steps. So you're multiplying each gradient by small constant, which is called step size. It is very similar to what we do in gradient boosting.
- And gradient boosting is considered to be a minimization in the functional space.

$$f_m(x)= f_0(x)+vh_1(x)+ vh_2(x)+\dots$$

Boosting-Minimization in the functional space

So here, you might have noticed that, gradient boosting is somewhat similar to the gradient descent in the optimization theory. That if we want, to minimize the function in the optimization theory using gradient descent, we make a small steps in the direction opposite to the gradient. So, gradient of a function, by definition, is a vector which points to the direction, with the fastest increase. Since we want to minimize the function, we must move to the direction opposite to the gradient, to ensure the convergence, we must make very small steps, so you are multiplying each gradient by a small constant, which is called a, 'Step Size'. It is very similar to what we do in the gradient boosting. So, gradient boosting is considered, minimization in the functionality space. That is FM of X is nothing but, F 0 of X plus V times H 1 of X plus V times s 2 of X and so on. So, boosting is the minimization in the functional space.

Refer Slide Time :(8:38)

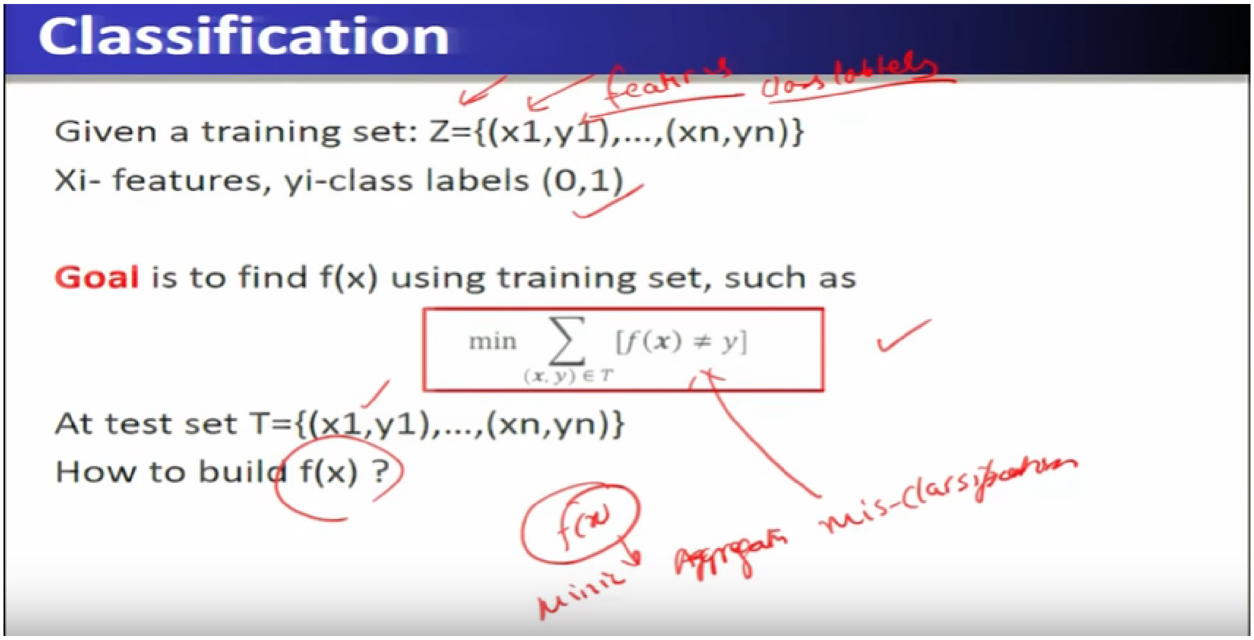


Summary

- **Boosting** is a method for combining outputs of many weak classifiers or regressors to produce a powerful ensemble.
- **Gradient Boosting** is a gradient descent minimization of the target function in the functional space.
- **Gradient Boosting with Decision Trees** is considered to be the best algorithm for general purpose classification or regression problems.

So, let us see the summary and boosting is the method for combining the outputs of many weak classifiers or the regressors to produce a powerful and ensemble. And gradient boosting is a, is a gradient decent minimization of the target functions, in the functional space. So, gradient boosting with the decision tree is considered to be the best algorithm for general purpose classification or the regression problem. Now, let us see the gradient boosted decision trees for classification problem,

Refer Slide Time :(9:13)



Classification

Given a training set: Z={(x1,y1),…,(xn,yn)}
Xi- features, yi-class labels (0,1)

Goal is to find f(x) using training set, such as

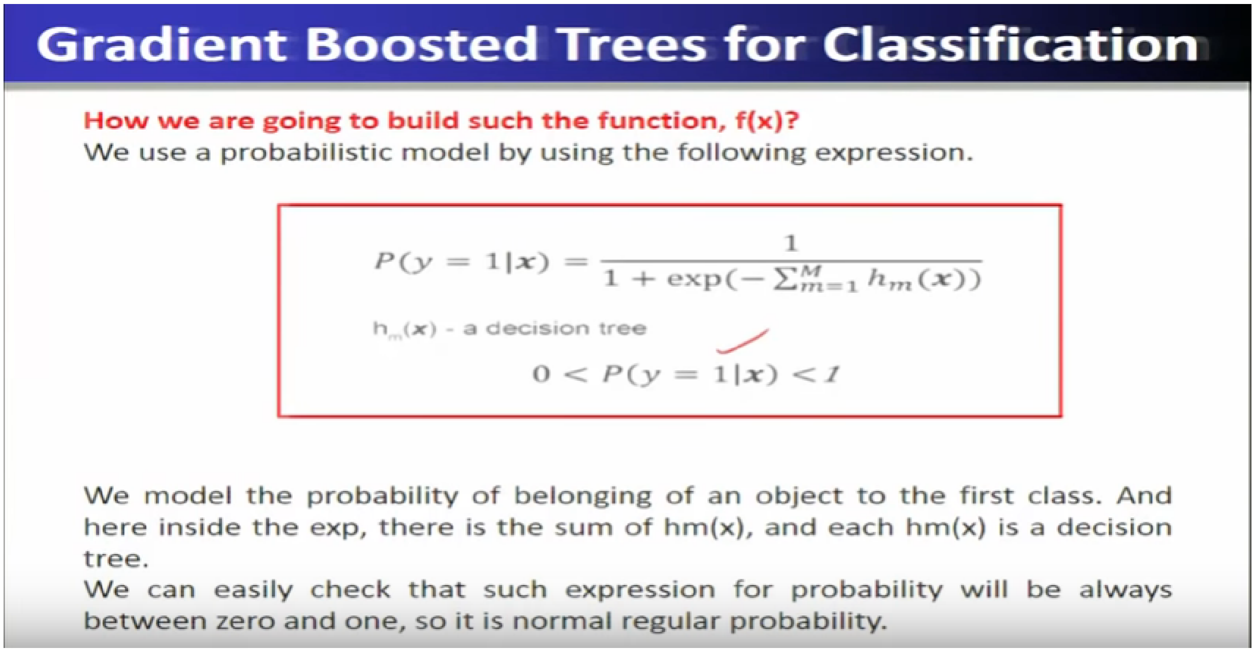
$$\boxed{\min \sum_{(x,y)\in T} [f(x) \neq y]}$$

At test set T={(x1,y1),…,(xn,yn)}
How to build f(x) ?

features
class lablels
f(x)
minimize
Aggregate mis-classification

from let us assume that, the training set which is given that is Z, comprises of X 1, y 1 and to y2 ends 1 up to X & Y and we're in X 1 is nothing but they are the features and y1 is nothing but, the class labels. So here, there will be a class label Y because this is a, classification problem so the classes, lies between lies 0 & 1. So, class labels let us assume that it is 0 & 1, so these will be assigned as the labels, in the training dataset. Now, goal here is to find out that, effects using the training data set such that, it will minimize this particular function that is the summation of, summation of FX, which is not equal to the label of y. So that means, the label FX is the predicted value and Y is that, the target label if it is not matching, so that becomes an error and the summation of all such miss classification. So, the summation of, so the aggregation of miss classification is to be minimized, so that should be the value of FX, which can achieve this so that, this particular prediction can be applied on, the test data set. So, test data set consists of x1 y1, x2 y2 and so on up to xn. So, how to build this FX is? The problem of gradient boosted decision trees.

Refer Slide Time :(11:12)



Gradient Boosted Trees for Classification

How we are going to build such the function, f(x)?

We use a probabilistic model by using the following expression.

$$P(y = 1|\boldsymbol{x}) = \frac{1}{1 + \exp(-\sum_{m=1}^{M} h_m(\boldsymbol{x}))}$$

$h_m(\boldsymbol{x})$ - a decision tree

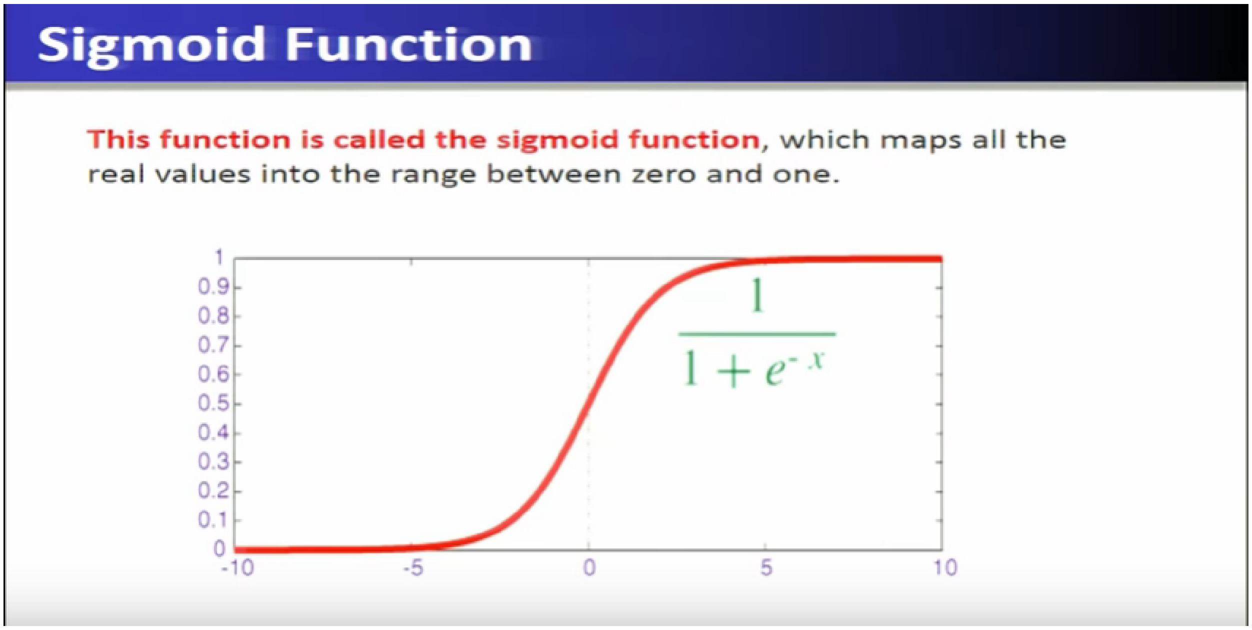
$$0 < P(y = 1|\boldsymbol{x}) < 1$$

We model the probability of belonging of an object to the first class. And here inside the exp, there is the sum of hm(x), and each hm(x) is a decision tree.

We can easily check that such expression for probability will be always between zero and one, so it is normal regular probability.

So, gradient boosted trees for classification. Here, we have to see how we are going to build, such a function FX. So, we use the probabilistic method by using the following expression, so the probability where y is equal to 1 given X is given by this equation that is 1 upon, 1 plus exponential of minus to the rich power of summation of M is equal to 1 to M H X, where H X is the decision tree. So, therefore this the value of probability, function lies between 0 to 1. Therefore, we model the probability of belonging of an, object to the first class and here, inside the exponential there is a sum of HM' S and each HM is a decision tree. So, we can easily check that each expression or the probability will always be between 0 and 1. So, it is the normal regular probability.

Refer Slide Time :(12:12)

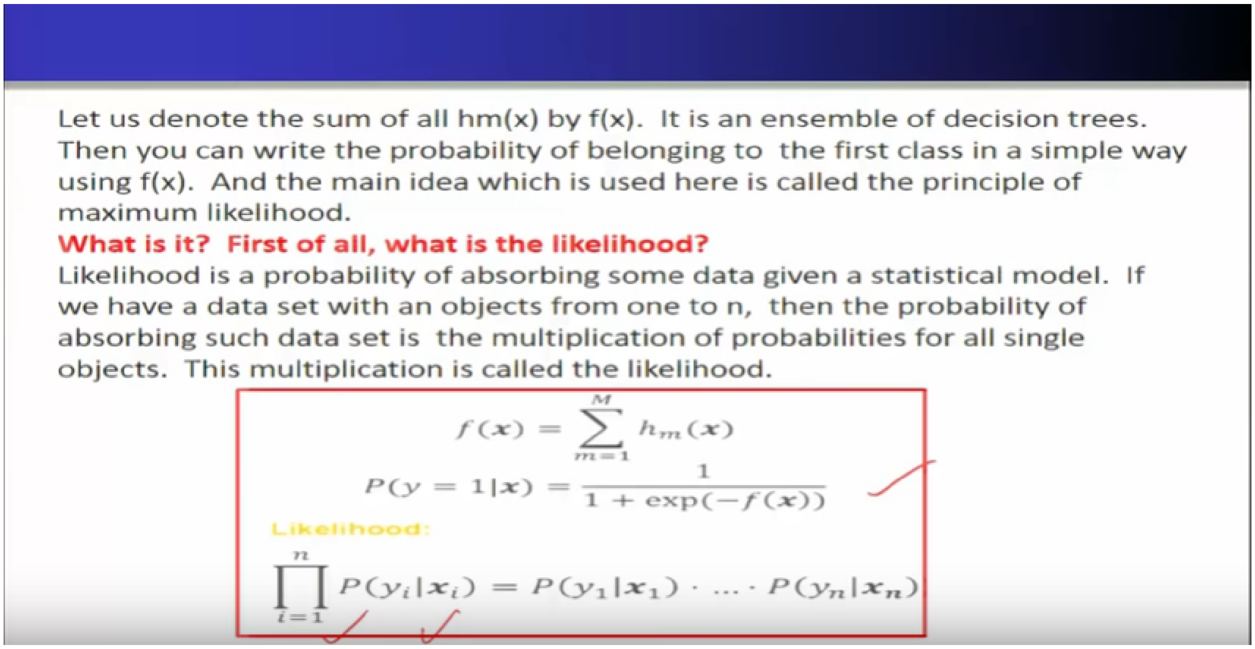


Sigmoid Function

This function is called the sigmoid function, which maps all the real values into the range between zero and one.

This particular function is called the, ‘Sigmoid Function’, which Maps all the real values, into the range between 0 & 1. So, this particular sigmoid function is 1 upon, 1 1 by 1 plus e raise power minus X.

Refer Slide Time :(12:29)



Let us denote the sum of all hm(x) by f(x). It is an ensemble of decision trees. Then you can write the probability of belonging to the first class in a simple way using f(x). And the main idea which is used here is called the principle of maximum likelihood.

What is it? First of all, what is the likelihood?

Likelihood is a probability of absorbing some data given a statistical model. If we have a data set with an objects from one to n, then the probability of absorbing such data set is the multiplication of probabilities for all single objects. This multiplication is called the likelihood.

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} h_m(\boldsymbol{x})$$

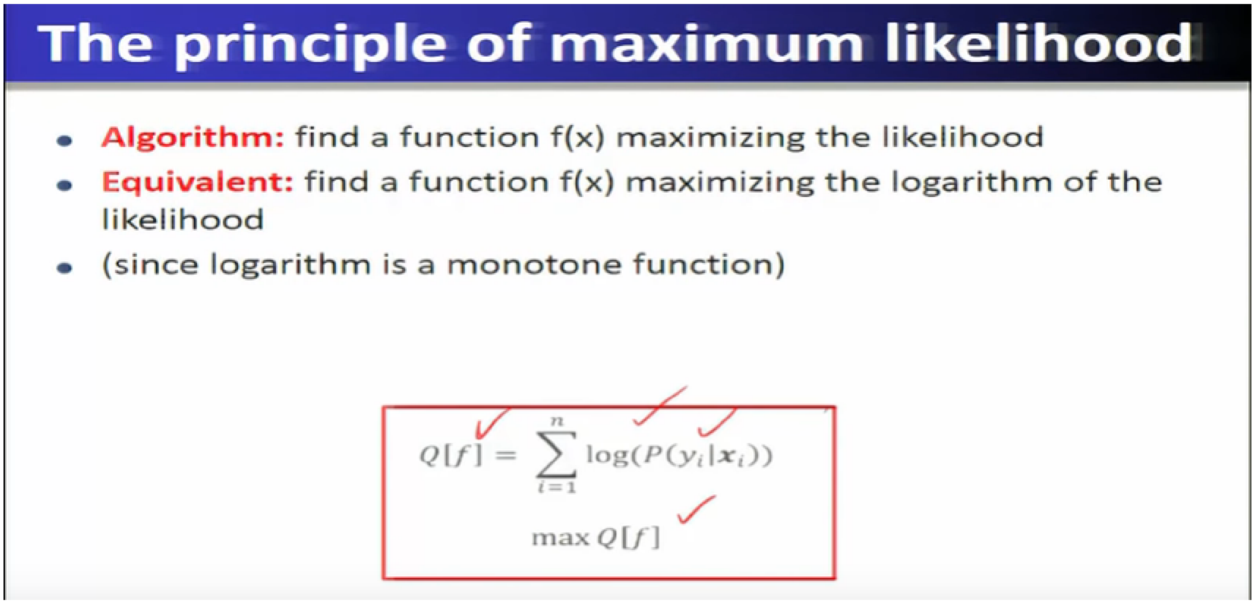
$$P(y = 1|\boldsymbol{x}) = \frac{1}{1 + \exp(-f(\boldsymbol{x}))}$$

Likelihood:

$$\prod_{i=1}^{n} P(y_i|\boldsymbol{x}_i) = P(y_1|\boldsymbol{x}_1) \cdot \ldots \cdot P(y_n|\boldsymbol{x}_n)$$

So here, also the same equation is appearing 1 by 1 plus exponential, exponential to the power of minus FX. This is called sigmoid function. So, let us let us denote the sum of hm(x) by FX. It is an ensemble of decision trees, then you can write the probability of the belonging to a 1st class in a simple, way using FX and the main idea which is used here, is called the, ‘Principle of Maximum Likelihood’. So, what is this principle of like maximum likelihood? So, likelihood is a probability of absorbing some data, given the statistical model, if we have the data set, with an object from 1 to n, then the probability of absorbing such data set is the multiplication of probabilities of all single objects, the multiplication is called, ‘Likelihood’. And here that can be expressed, likelihood as the multiplication of the probabilities for all single objects that is nothing but, probability of y 1 given X 1 dot, probability of Y 2 given X 2, X 2 and so on, 2 probability 1. So therefore, it comes out to be the multiplication of all the probabilities.

Refer Slide Time :(13:53)



The principle of maximum likelihood

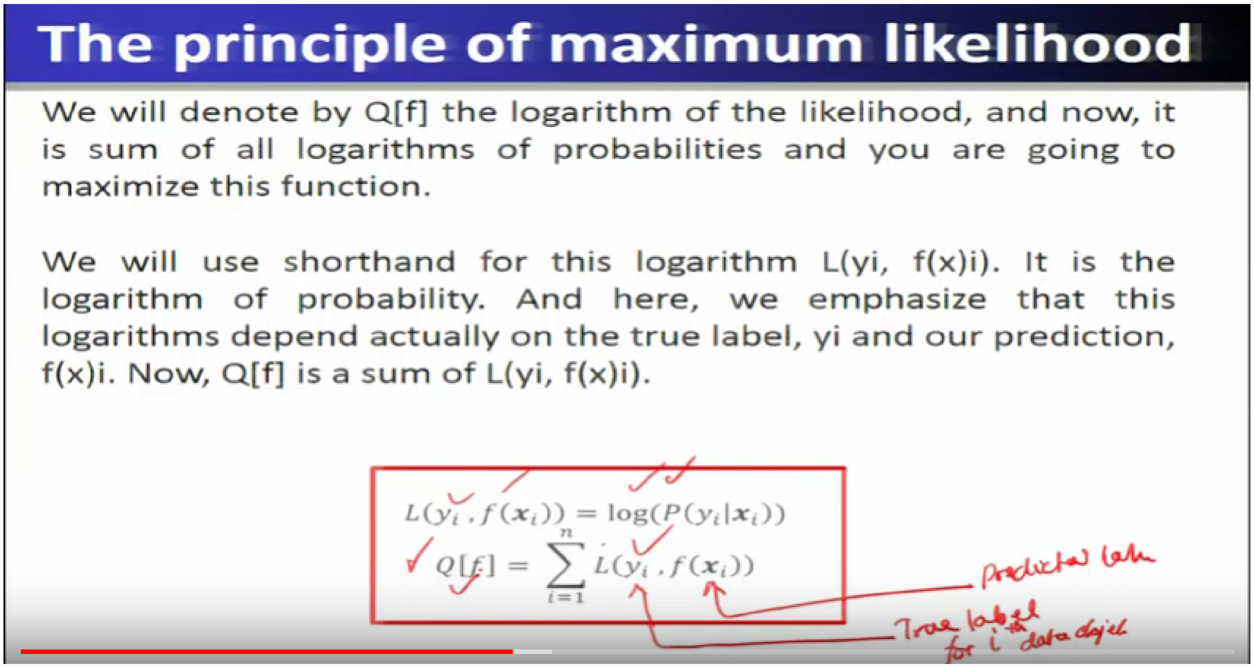
- **Algorithm:** find a function f(x) maximizing the likelihood
- **Equivalent:** find a function f(x) maximizing the logarithm of the likelihood
- (since logarithm is a monotone function)

$$Q[f] = \sum_{i=1}^{n} \log(P(y_i|\boldsymbol{x}_i))$$

$$\max Q[f]$$

And now, likelihood function we have to calculate. So, the principle of maximum likelihood, can be given by this algorithm, to find a function FX which maximizing the likelihood, which is equivalent to find FX maximizing the logarithmic of the likelihood. Since, the logarithmic is the monotone function. So that can be represented here in this case that, the logarithmic of the probability of all Y ones given X 1 and this particular summation, here is called ‘QF’. And we have to find out the maximum of qf that is, which will maximize the likelihood. So, we have to find out that FX, which will maximize the likelihood.

Refer Slide Time :(14:45)



The principle of maximum likelihood

We will denote by Q[f] the logarithm of the likelihood, and now, it is sum of all logarithms of probabilities and you are going to maximize this function.

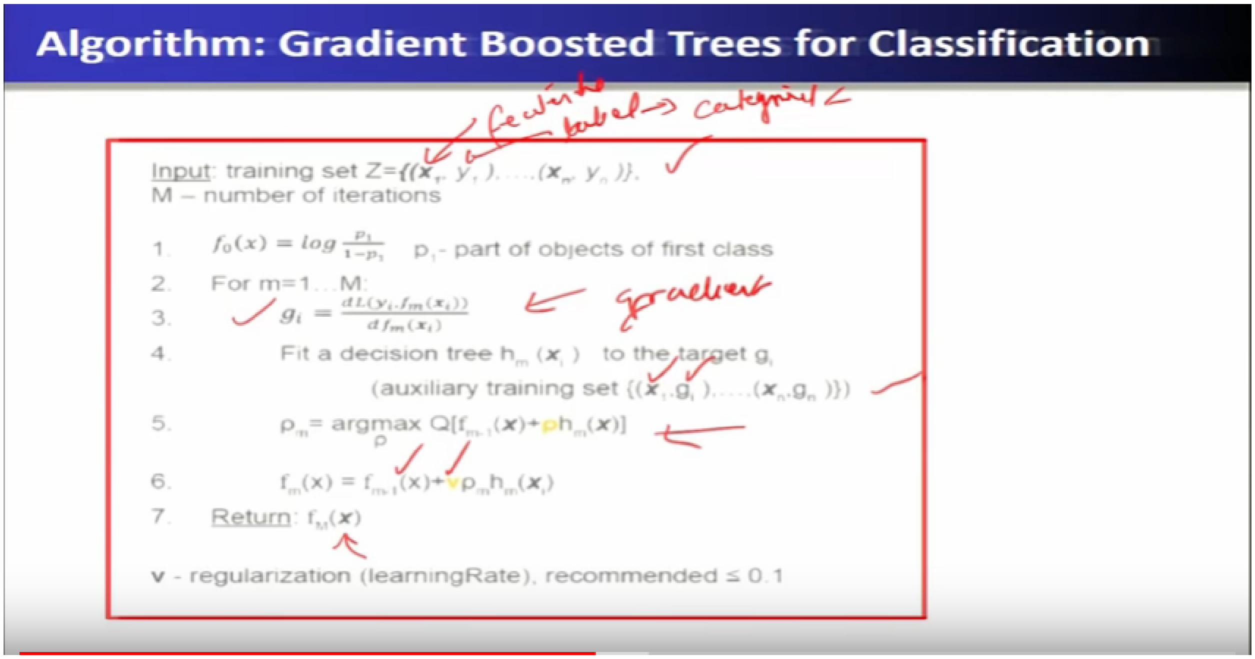
We will use shorthand for this logarithm L(yi, f(x)i). It is the logarithm of probability. And here, we emphasize that this logarithms depend actually on the true label, yi and our prediction, f(x)i. Now, Q[f] is a sum of L(yi, f(x)i).

$$L(y_i, f(\boldsymbol{x}_i)) = \log(P(y_i|\boldsymbol{x}_i))$$

$$Q[f] = \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i))$$

Hence, we have to fit our distribution in this data set, by way of, principle of maximum likelihood .So, we will denote it by Q of F, the logarithmic of the likelihood and it's now ,it is the sum of all the logarithm logarithms of the probabilities and you are going to maximize this particular function. Now, you will use the shorthand, for this logarithmic that is capital L by Y I and FX of I. So, it is the logarithmic of the probability. And here, we emphasize that this logarithms, depends actually on the true label that is Y I and our predicted values that is FX of I. And now, Q of F is the sum of, the logarithm of Y I and F X of I. So, logarithmic, a logarithm of Y I and FX of Y is nothing but, given as likely, given as the log of the probability of Y given X I and which is nothing but, Q F is nothing but the summation of all I is from 1 to N and the logarithmic of Y I and FX of I. So, Y I is the, the true label, for the idea the data object in the set and FX of I is, the predicted label and this logarithmic of, of this is represented by this likelihood and the summation of this is represented by this, likelihood which has to be maximized for that key way.

Refer Slide Time :(16:42)



Algorithm: Gradient Boosted Trees for Classification

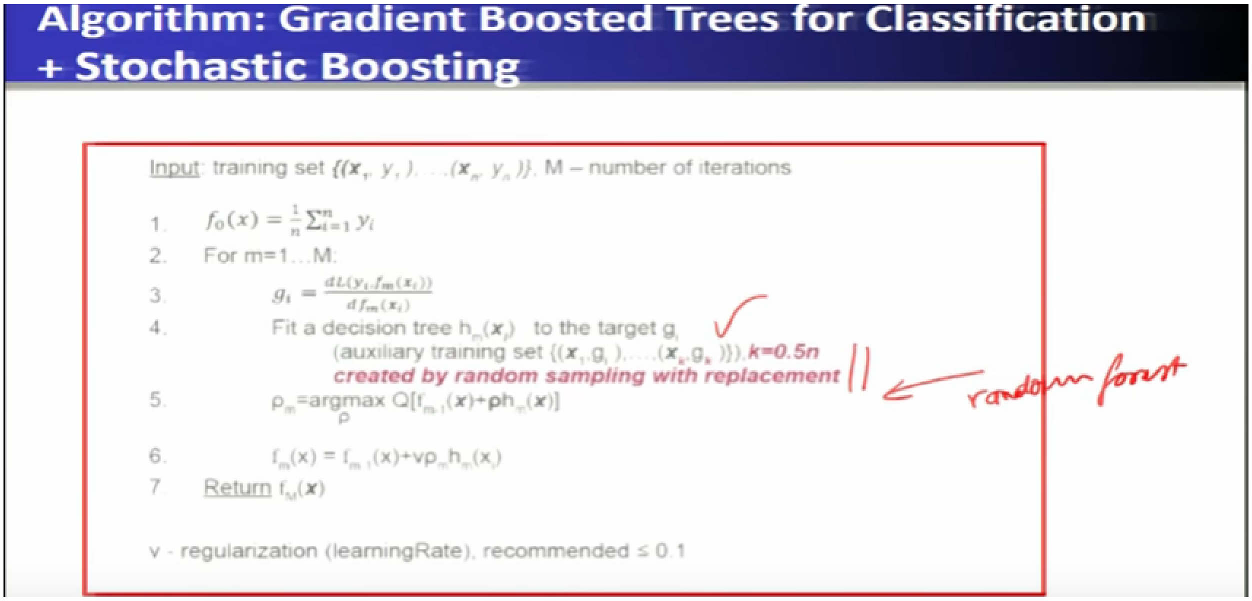
Input: training set $Z=\{(x_1, y_1),\ldots,(x_n, y_n)\}$.
M – number of iterations

```
1. $f_0(x) = \log\frac{p_1}{1-p_1}$    $p_1$ - part of objects of first class
2. For m=1...M:
3.     $g_i = \frac{dL(y_i, f_m(x_i))}{df_m(x_i)}$
4.     Fit a decision tree $h_m(x_i)$ to the target $g_i$
           (auxiliary training set $\{(x_1, g_1),\ldots,(x_n, g_n)\}$)
5.     $\rho_m = \underset{\rho}{\arg\max}\, Q[f_{m-1}(x) + \rho h_m(x)]$
6.     $f_m(x) = f_{m-1}(x) + v\rho_m h_m(x_i)$
7. Return: $f_M(x)$
```

v - regularization (learningRate), recommended ≤ 0.1

Let us see the gradient boosted trees for classification, by putting it everything whatever we have discussed. Now, the algorithm for gradient boosted trees for classification. Here, the input set Z is given as, X I, Y I where in X I you know that, it's a features in the data set and why is the labels, which are assigned as, the categorical type that is labels given. And M is the number of iterations and the for, the first class, the part of the objects of the first class is P of 1 and F 0 of X is equal to log of P 1 minus P 1 by 1 minus P 1 and for, iterating between from 1 to M, so we will find out the gradient that is GI is equal to differentiation of, this likelihood function L of Y I and FM of X I. So, this will calculate divided by differentiation of F M of X I. So, this is called the, 'Gradient'. So, gradients are calculated and then, it will fit a decision tree hm of X, to the target, to the target GI. So, auxilary training data set, which will be used here, is that, X I and x1 and then it will be replaced by, label will be replaced by the gradients and this will call an, 'Auxilary Data Set'. So, given the other rate data set, it will fit the decision tree that is called, 'HM of X I'. And wherein the role value will be arc max of Rho that is for Q of F M minus a1 X plus Rho times, H M of X. So, Rho will be calculated and F M of X is equal to FM minus 1 of X plus, V is the regularization coefficient Rho M and H M of X I. So, this process will in turn, will give the X symbol of different values. And so, it will do this kind of classification in this particular manner. So, let us see the stochastic boosting, so gradient boosting trees for classification, if we use this stochastic boosting.

Refer Slide Time :(19:49)



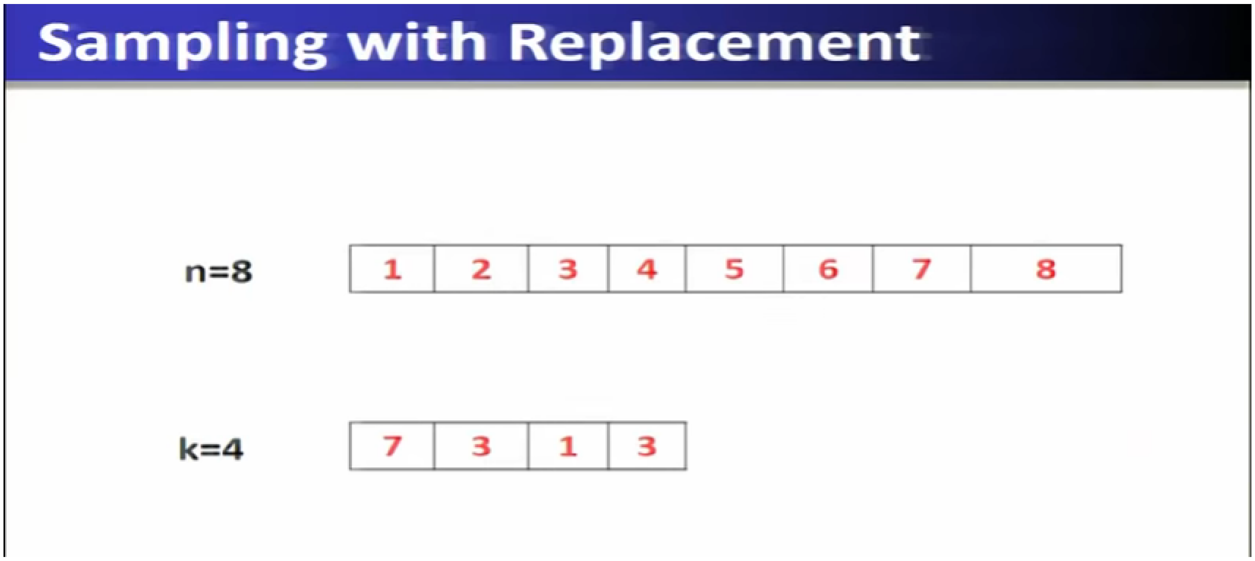
Algorithm: Gradient Boosted Trees for Classification + Stochastic Boosting

```
Input: training set {(x_1, y_1),...,(x_n, y_n)}, M – number of iterations
1.  $f_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$
2.  For m=1...M:
3.      $g_i = \frac{dL(y_i, f_m(x_i))}{df_m(x_i)}$
4.      Fit a decision tree $h_m(x_i)$ to the target $g_i$
            (auxiliary training set {(x_1,g_1),...,(x_k,g_k)}).k=0.5n
            created by random sampling with replacement
5.      $\rho_m = \underset{\rho}{\operatorname{argmax}}\, Q[f_{m-1}(x)+\rho h_m(x)]$
6.      $f_m(x) = f_{m-1}(x) + v\rho_m h_m(x_i)$
7.  Return $f_M(x)$

v - regularization (learningRate), recommended ≤ 0.1
```

random forest

Then it will be represented here in this case, we are this observe a set, which is used as the training set will be now, k is equal to 0.5 n will be created by the random sampling with the replacement, so this particular part here, we are going to use the, the concept of the random forest, for the bootstrap generation of the data side. So, this is the, the gradient boosting, gradient boosted trees + stochastic boosting is shown over here.

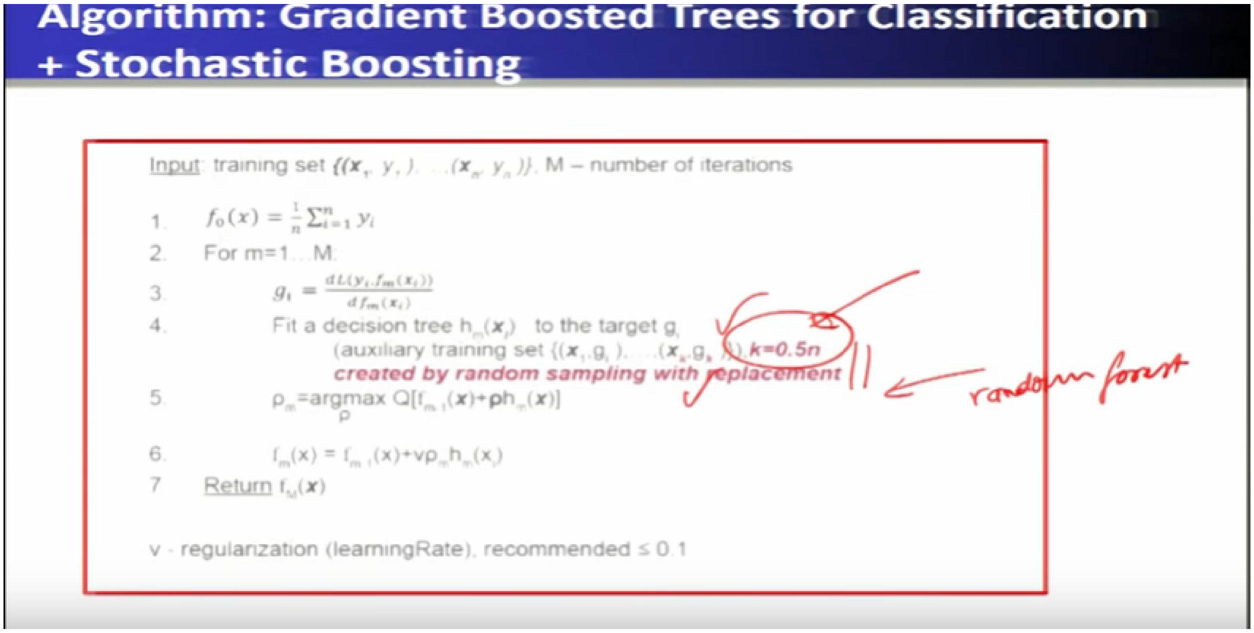
Refer Slide Time :(20:32)



Sampling with Replacement

n=8	1	2	3	4	5	6	7	8
k=4	7	3	1	3				

And this way, sampling with the replacement is there.
Refer Slide Time :(20:37)



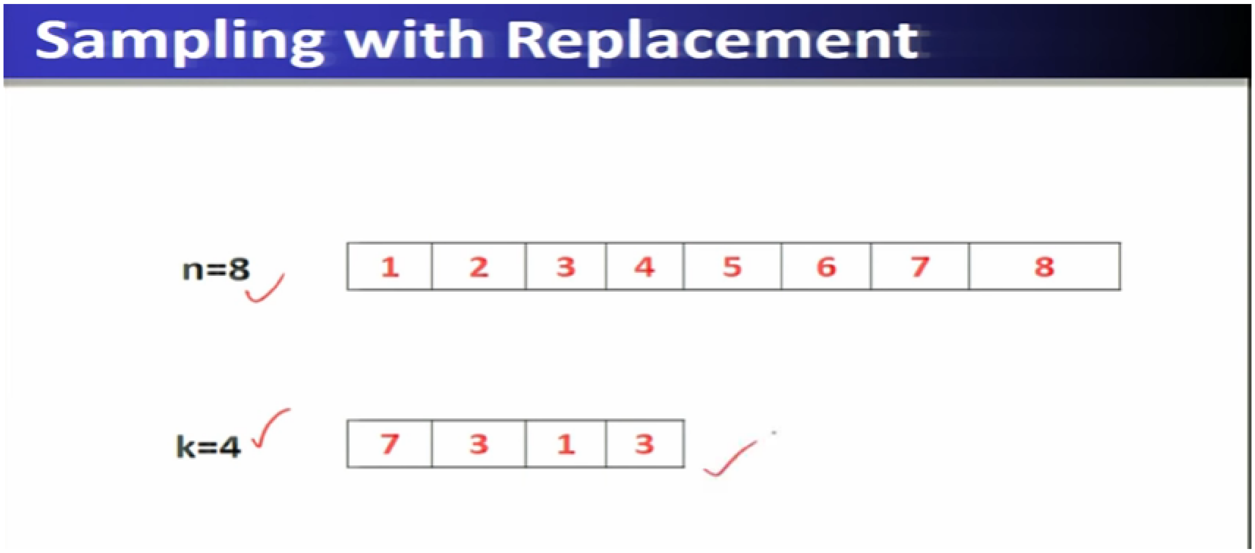
Algorithm: Gradient Boosted Trees for Classification + Stochastic Boosting

```
Input: training set {(x_1, y_1), ...(x_n, y_n)}, M – number of iterations

1.  $f_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$
2.  For m=1...M:
3.      $g_i = \frac{dL(y_i, f_m(x_i))}{d f_m(x_i)}$
4.      Fit a decision tree $h_m(x_i)$ to the target $g_i$
            (auxiliary training set $\{(x_1, g_1), ..., (x_k, g_k)\}$), $k=0.5n$
            created by random sampling with replacement
5.      $\rho_m = \underset{\rho}{\operatorname{argmax}}\, Q[f_{m-1}(x) + \rho h_m(x)]$
6.      $f_m(x) = f_{m-1}(x) + v\rho_m h_m(x_i)$
7   Return $f_M(x)$

v - regularization (learningRate), recommended ≤ 0.1
```

random forest

So here, we have to see that the size of the sample will be reduced by K is equal to 0.5 n that is here, the author a training set, will be reduced by the size half and it will be created, by random sampling with the replacement. And this is called the, the, the, 'Concept of Bagging', which will be used over here in the stochastic boosting.

Refer Slide Time :(21:07)



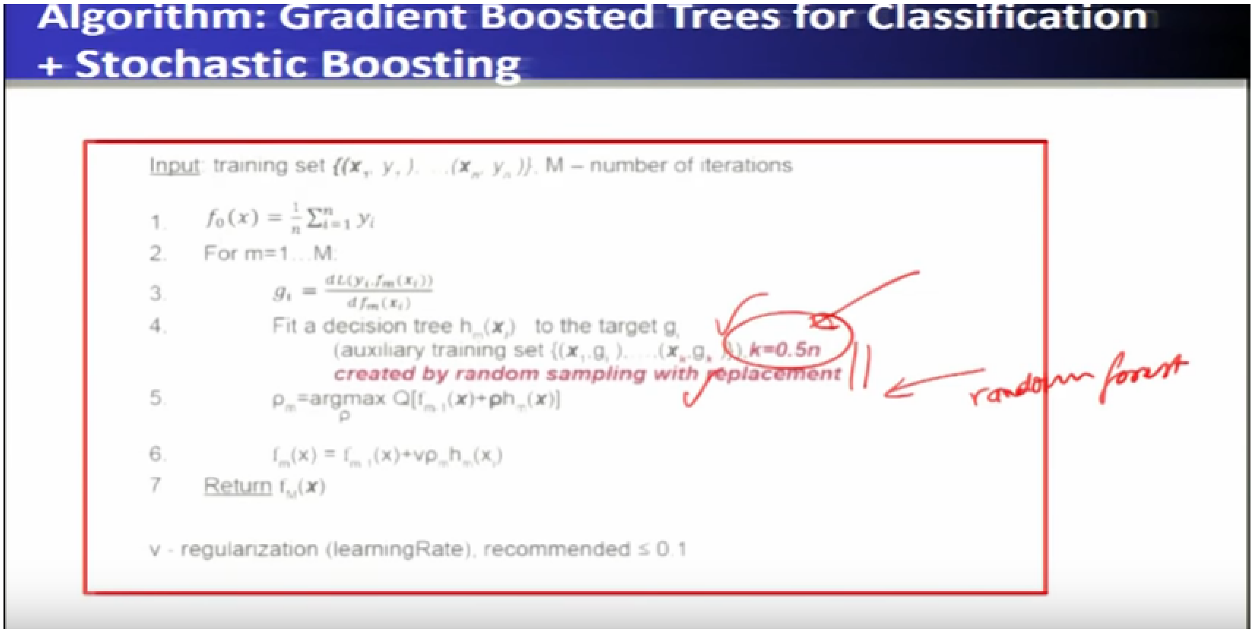
Sampling with Replacement

n=8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

k=4 | 7 | 3 | 1 | 3

So, sampling with the replacement is used but here, the value of K will become half of, the size of the, the, the total features. So here, it will be half of that particular features, will be taken up into the considerations.

Refer Slide Time :(21:26)



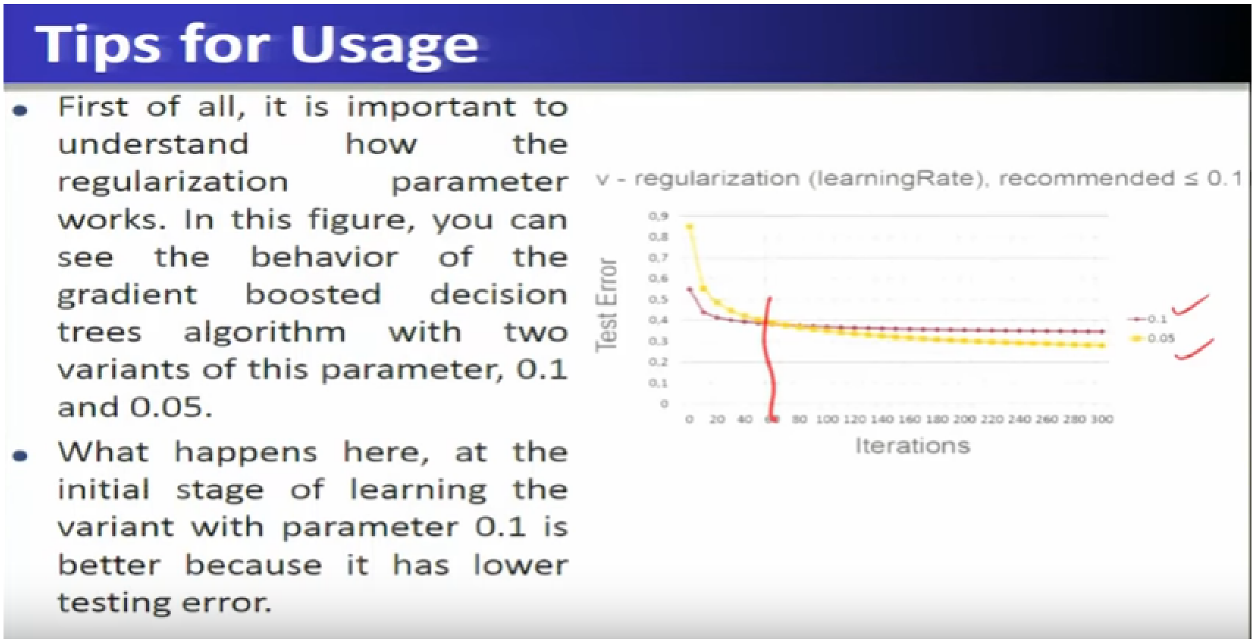
Algorithm: Gradient Boosted Trees for Classification + Stochastic Boosting

```
Input: training set $\{(x_1, y_1), \ldots (x_n, y_n)\}$, M – number of iterations

1.  $f_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$
2.  For m=1…M:
3.      $g_i = \frac{dL(y_i, f_m(x_i))}{df_m(x_i)}$
4.      Fit a decision tree $h_m(x_i)$ to the target $g_i$
            (auxiliary training set $\{(x_1, g_1), \ldots (x_k, g_k)\}$) k=0.5n
            created by random sampling with replacement
5.      $\rho_m = \underset{\rho}{\arg\max}\, Q[f_{m-1}(x) + \rho h_m(x)]$
6.      $f_m(x) = f_{m-1}(x) + v\rho_m h_m(x_i)$
7.  Return $f_M(x)$

v - regularization (learningRate), recommended ≤ 0.1
```

random forest

Hence, as we reduce the size, this becomes more efficient this particular case.

Refer Slide Time :(21:33)



Tips for Usage

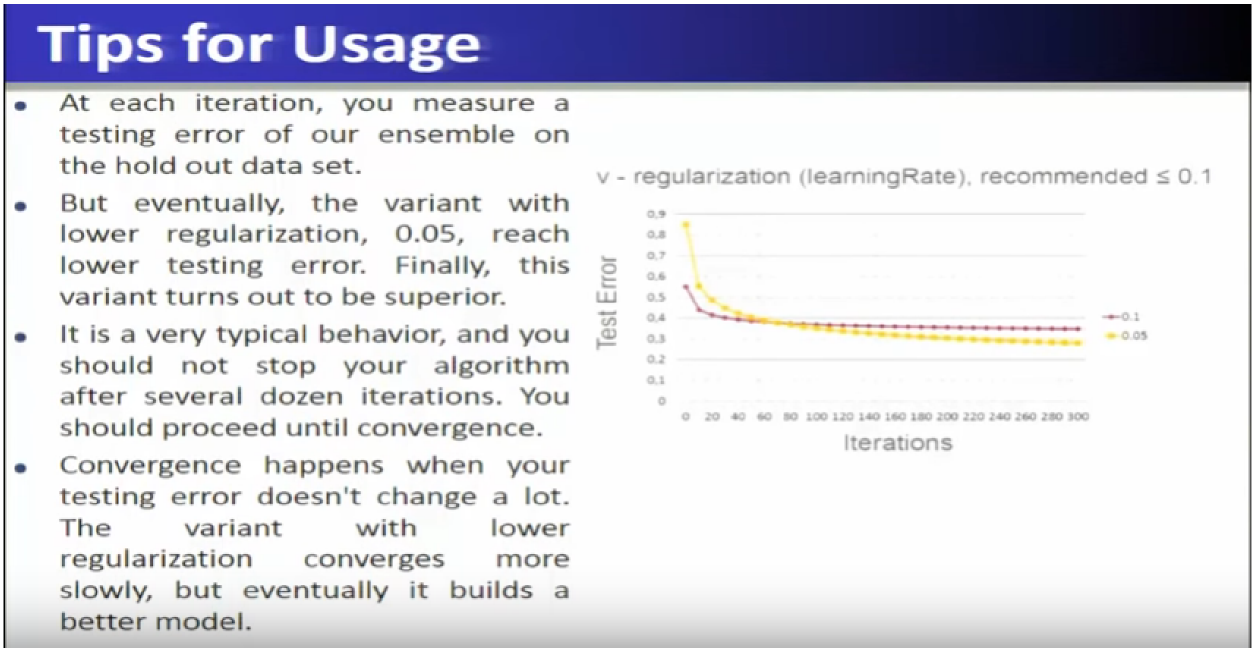
- First of all, it is important to understand how the regularization parameter works. In this figure, you can see the behavior of the gradient boosted decision trees algorithm with two variants of this parameter, 0.1 and 0.05.
- What happens here, at the initial stage of learning the variant with parameter 0.1 is better because it has lower testing error.

v - regularization (learningRate), recommended ≤ 0.1

So, let us see the tips for the usage, first of all it is important to understand how the regularization parameter works. In this figure, we can see that the behavior of the gradient boosted, decision trees with the, with, with the two variants we can see with the parameter, point 1 & 0 & point 0 5 that is point 0 5.

So, we can see that, this one point 1 is not that accurate and point 0 5 is more accurate. So, what happens, here is that initially stage of the learning, at the initial stage the learning, the, the variant that is 0.1 is better but because, it has the lower testing error.

Refer Slide Time :(22:21)

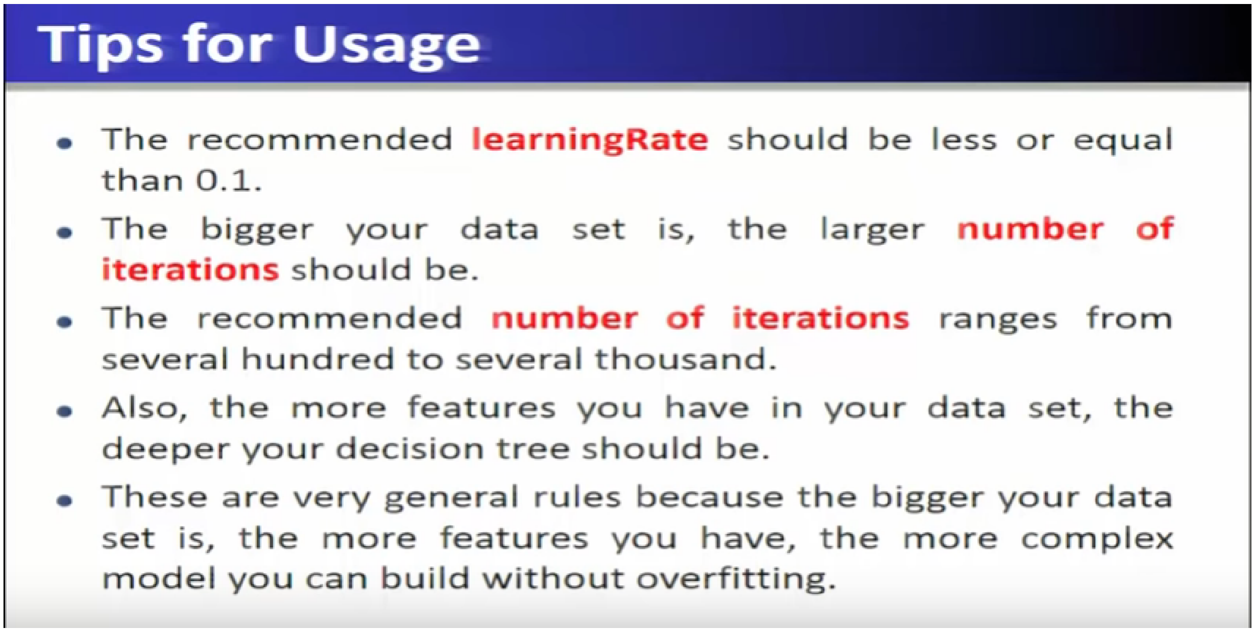


Tips for Usage

- At each iteration, you measure a testing error of our ensemble on the hold out data set.
- But eventually, the variant with lower regularization, 0.05, reach lower testing error. Finally, this variant turns out to be superior.
- It is a very typical behavior, and you should not stop your algorithm after several dozen iterations. You should proceed until convergence.
- Convergence happens when your testing error doesn't change a lot. The variant with lower regularization converges more slowly, but eventually it builds a better model.

But, later on so at each iteration, you measure the testing error, up our example of on the holed out data set. But, eventually the variant with the lower regularization that is point zero five, reach the lower testing error, finally this variant turns out to be the superior. So, it is very typical behavior and you should not stop, your algorithm after several dozens of iterations, so you should proceed over until it converges. So, convergence happens when you're testing error does not change a lot. The variant with the lower regularization convert more slowly, but eventually it builds a better model.

Refer Slide Time :(23:04)

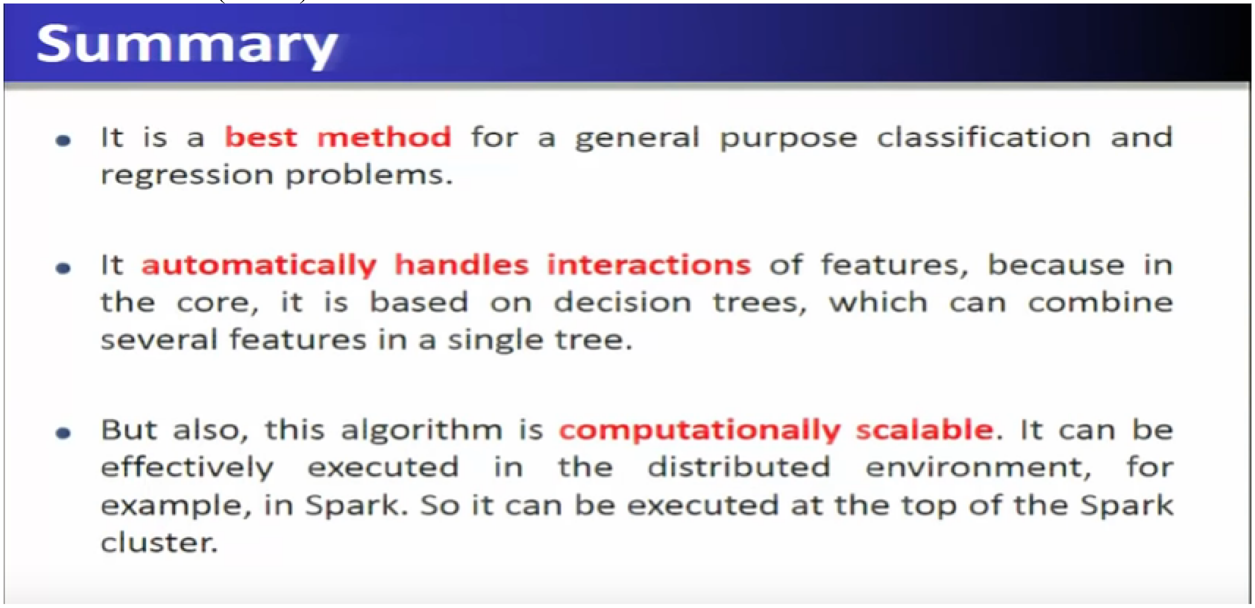


Tips for Usage

- The recommended **learningRate** should be less or equal than 0.1.
- The bigger your data set is, the larger **number of iterations** should be.
- The recommended **number of iterations** ranges from several hundred to several thousand.
- Also, the more features you have in your data set, the deeper your decision tree should be.
- These are very general rules because the bigger your data set is, the more features you have, the more complex model you can build without overfitting.

So, the recommended learning rate, should be less than or equal to 0.1 that we have already seen. And the bigger your data set is, the larger it will be the number of iterations it, should have. So, the recommended number of iteration ranges from several hundred to the several thousand's. Also, the more features you have in your data set, the deeper will be your decision tree. And there are many general rules, because the bigger your data set is the more features you have, the more complex model you can build without over fitting, in this particular scenario.

Refer Slide Time :(23:39)

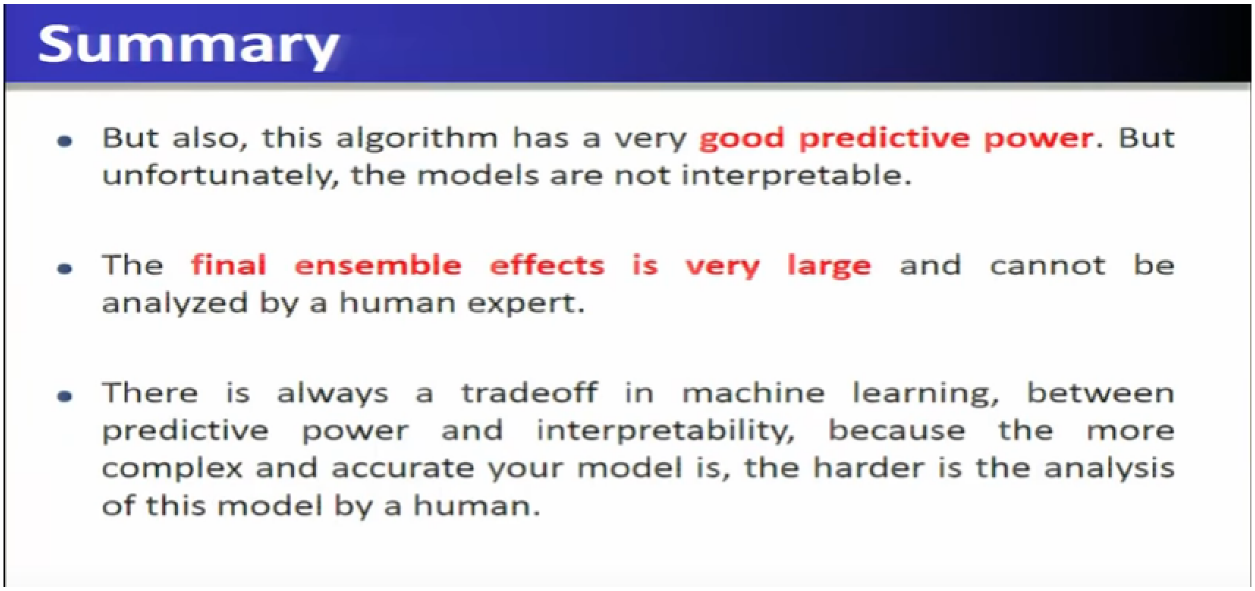


Summary

- It is a **best method** for a general purpose classification and regression problems.
- It **automatically handles interactions** of features, because in the core, it is based on decision trees, which can combine several features in a single tree.
- But also, this algorithm is **computationally scalable**. It can be effectively executed in the distributed environment, for example, in Spark. So it can be executed at the top of the Spark cluster.

So, somebody it is the best method, for general-purpose classification and regression problems. So, it automatically handles the interaction of the features, because in the core, it is based on the decision trees, which can combine several features in a single tree. But also, this algorithm is computationally scalable. It can be effectively executed in the distributed environment, for example, in the SPARK. So, it can be executed on top of the spark cluster.

Refer Slide Time :(24:09)



Summary

- But also, this algorithm has a very **good predictive power**. But unfortunately, the models are not interpretable.
- The **final ensemble effects is very large** and cannot be analyzed by a human expert.
- There is always a tradeoff in machine learning, between predictive power and interpretability, because the more complex and accurate your model is, the harder is the analysis of this model by a human.

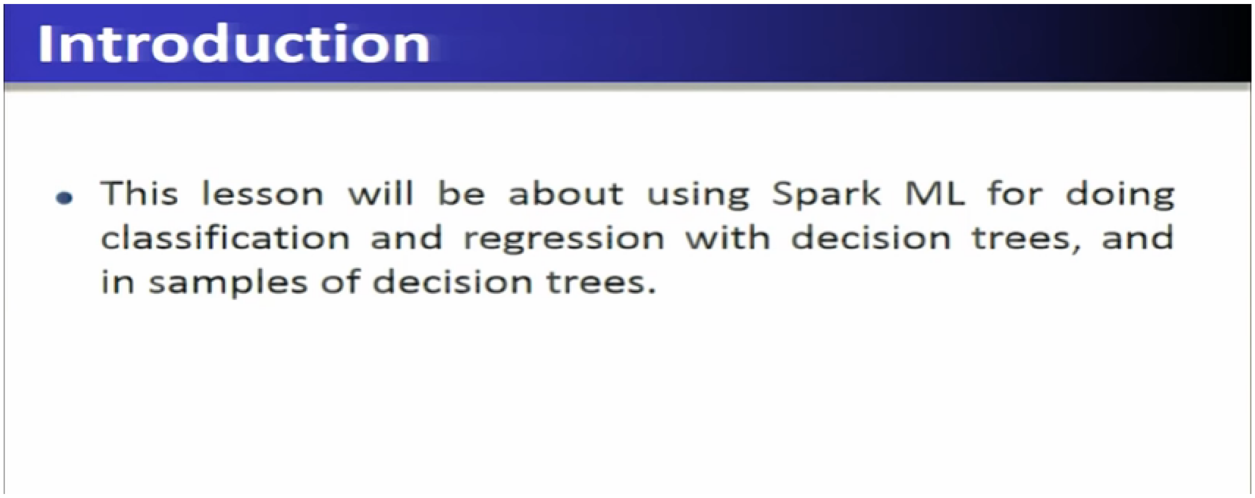
But also, this algorithm has very good predictive power. But, unfortunately the model are not interpretable. And that problem we have also seen, in the random forest but here, the predictive power is better than that in the forest. So, the final and simple effect is very large and cannot be analyzed by the human experts. So, obviously it is not having the good interpretation of this due to the complex, nature of this particular model. So, there are always a trade-off in the machine learning, between the predictive power and interpretability, because the more complex and accurate your model is the harder is the analysis of this model, to be interpreted by the humans.

Refer Slide Time :(24:53)

Spark ML, Decision Trees and Ensembles

Spark ML, based decision trees and examples, we have to see the programming aspect of it, how using is part ml we will now use, the decision tree and decision tree and ensemble.

Refer Slide Time :(25:10)

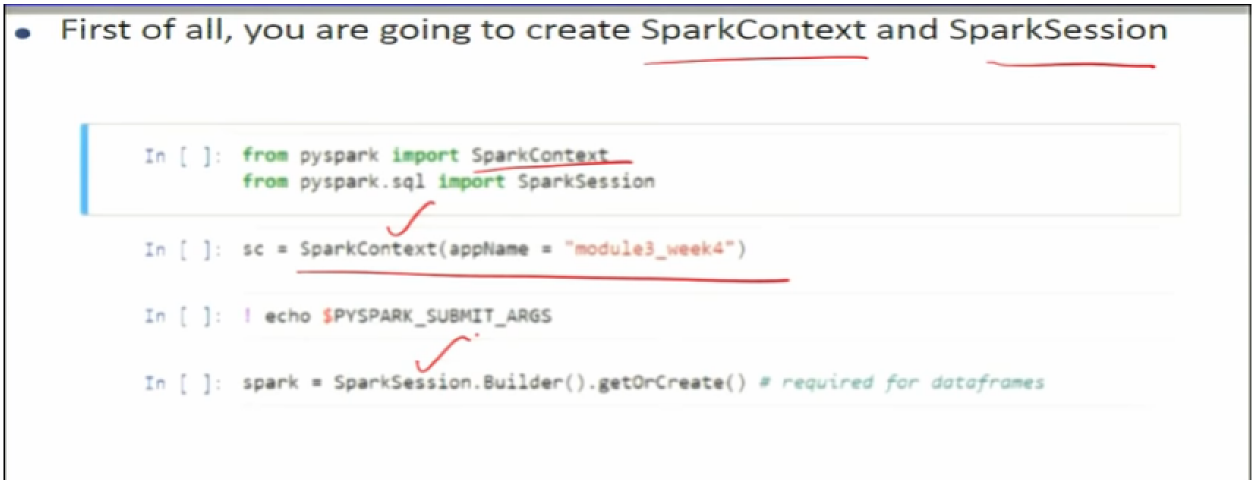


Introduction

- This lesson will be about using Spark ML for doing classification and regression with decision trees, and in samples of decision trees.

So here, we will talk about a spark ML for doing classification regression, with the ensemble trees.

Refer Slide Time :(25:19)

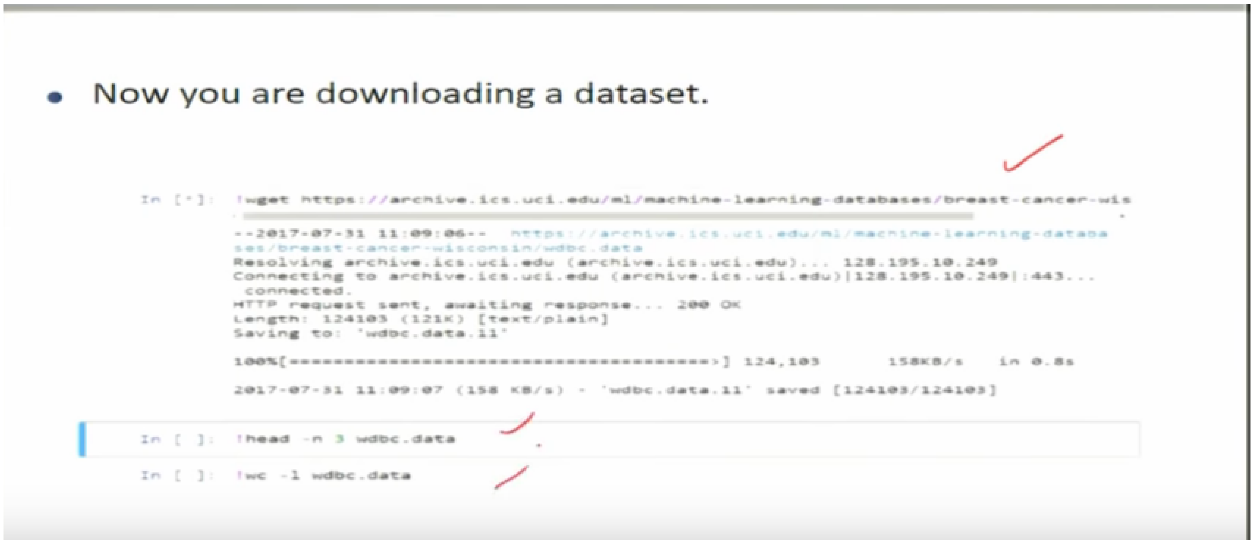


- First of all, you are going to create SparkContext and SparkSession

```
In [ ]: from pyspark import SparkContext
        from pyspark.sql import SparkSession

In [ ]: sc = SparkContext(appName = "module3_week4")

In [ ]: ! echo $PYSPARK_SUBMIT_ARGS

In [ ]: spark = SparkSession.Builder().getOrCreate() # required for dataframes
```

And first we will see that, we have to, we have to first see the decision trees how, the decision tree will be implemented over the spark image and then we can extend it for, the random forest and gradient boosted trees. So, the first step here is to create the spark context and in the spark session. Which is shown here, in this particular steps that we have to, create the spark context and we have to also, create the spark session.

Refer Slide Time :(25:59)

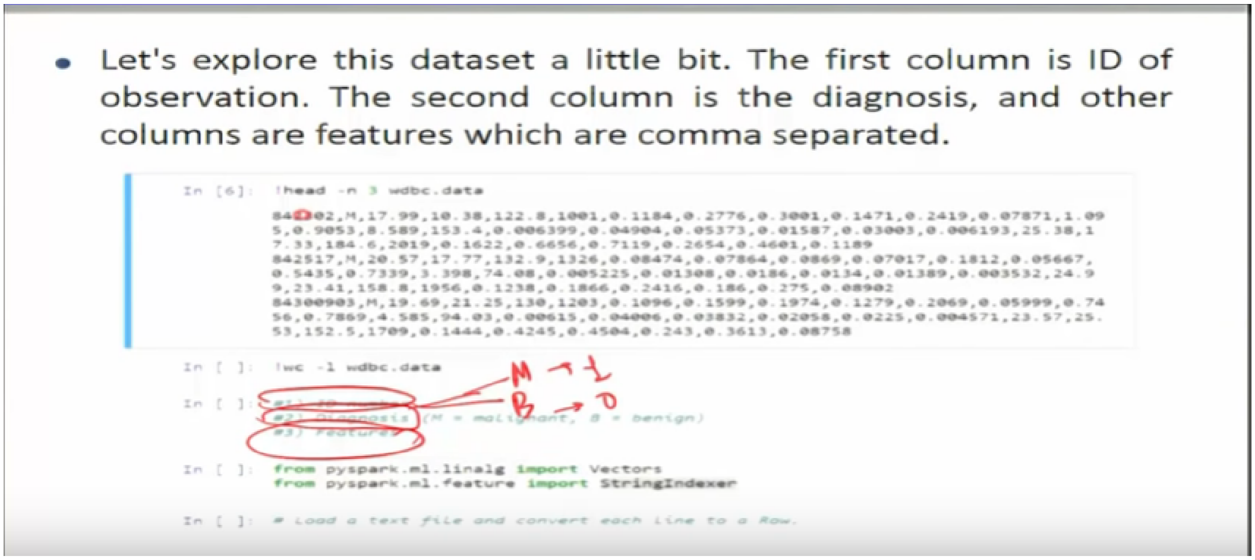


- Now you are downloading a dataset.

```
In [*]:  !wget https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wis

         --2017-07-31 11:09:06--  https://archive.ics.uci.edu/ml/machine-learning-databa
         ses/breast-cancer-wisconsin/wdbc.data
         Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.249
         Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.249|:443...
          connected.
         HTTP request sent, awaiting response... 200 OK
         Length: 124103 (121K) [text/plain]
         Saving to: 'wdbc.data.11'

         100%[======================================>] 124,103      158KB/s    in 0.8s

         2017-07-31 11:09:07 (158 KB/s) - 'wdbc.data.11' saved [124103/124103]

In [ ]:  !head -n 3 wdbc.data

In [ ]:  !wc -l wdbc.data
```

And this is spark session is required for building the data frame. And then, we can download the data set and now, we can see the data set, after downloading.

Refer Slide Time :(26:10)

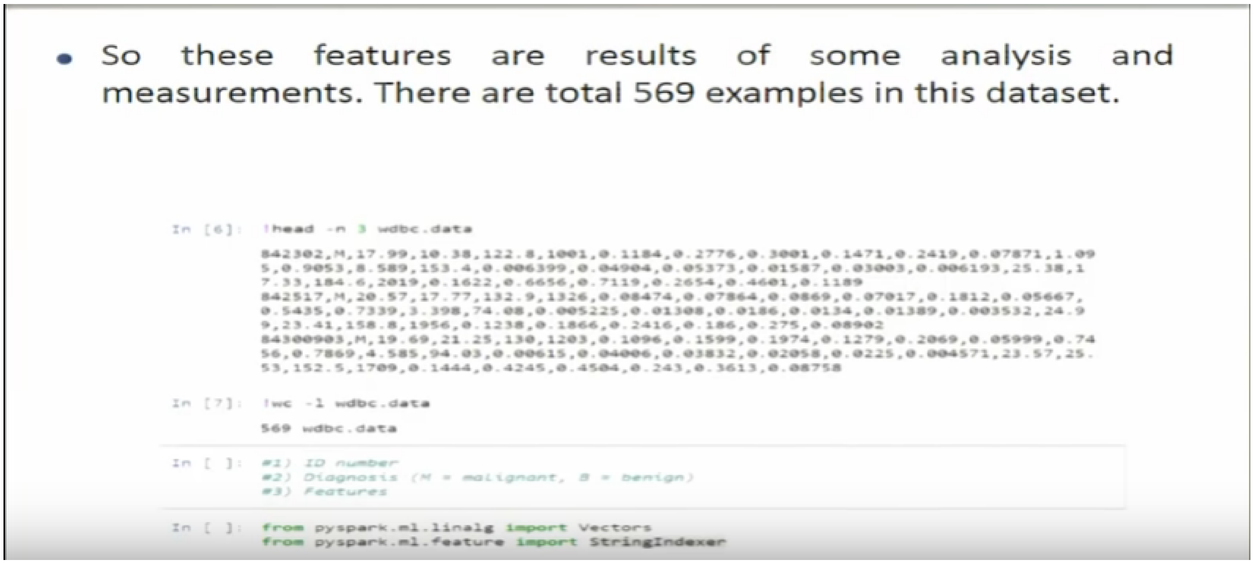


- Let's explore this dataset a little bit. The first column is ID of observation. The second column is the diagnosis, and other columns are features which are comma separated.

```
In [6]:  !head -n 3 wdbc.data

         842302,M,17.99,10.38,122.8,1001,0.1184,0.2776,0.3001,0.1471,0.2419,0.07871,1.09
         5,0.9053,8.589,153.4,0.006399,0.04904,0.05373,0.01587,0.03003,0.006193,25.38,1
         7.33,184.6,2019,0.1622,0.6656,0.7119,0.2654,0.4601,0.1189
         842517,M,20.57,17.77,132.9,1326,0.08474,0.07864,0.0869,0.07017,0.1812,0.05667,
         0.5435,0.7339,3.398,74.08,0.005225,0.01308,0.0186,0.0134,0.01389,0.003532,24.9
         9,23.41,158.8,1956,0.1238,0.1866,0.2416,0.186,0.275,0.08902
         84300903,M,19.69,21.25,130,1203,0.1096,0.1599,0.1974,0.1279,0.2069,0.05999,0.74
         56,0.7869,4.585,94.03,0.00615,0.04006,0.03832,0.02058,0.0225,0.004571,23.57,25.
         53,152.5,1709,0.1444,0.4245,0.4504,0.243,0.3613,0.08758

In [ ]:  !wc -l wdbc.data

In [ ]:  #1) ID number
         #2) Diagnosis (M = malignant, B = benign)
         #3) Features

In [ ]:  from pyspark.ml.linalg import Vectors
         from pyspark.ml.feature import StringIndexer

In [ ]:  # Load a text file and convert each line to a Row.
```

The data set into data frames and then we can see the features that, it has the data has an ID number and the label that is categorical label, which can be denoted by 0 & 1 or you can see that it is, it is M and B, finally which will be converted into 1 & 0 .So, this particular data set has the, the features and it has the labels and it has an ID. So, there are three different important parameter sighs. So, let us take this particular case, where all these important parts are, there in the data set. So, now we can further look into the data set.

Refer Slide Time :(27:05)

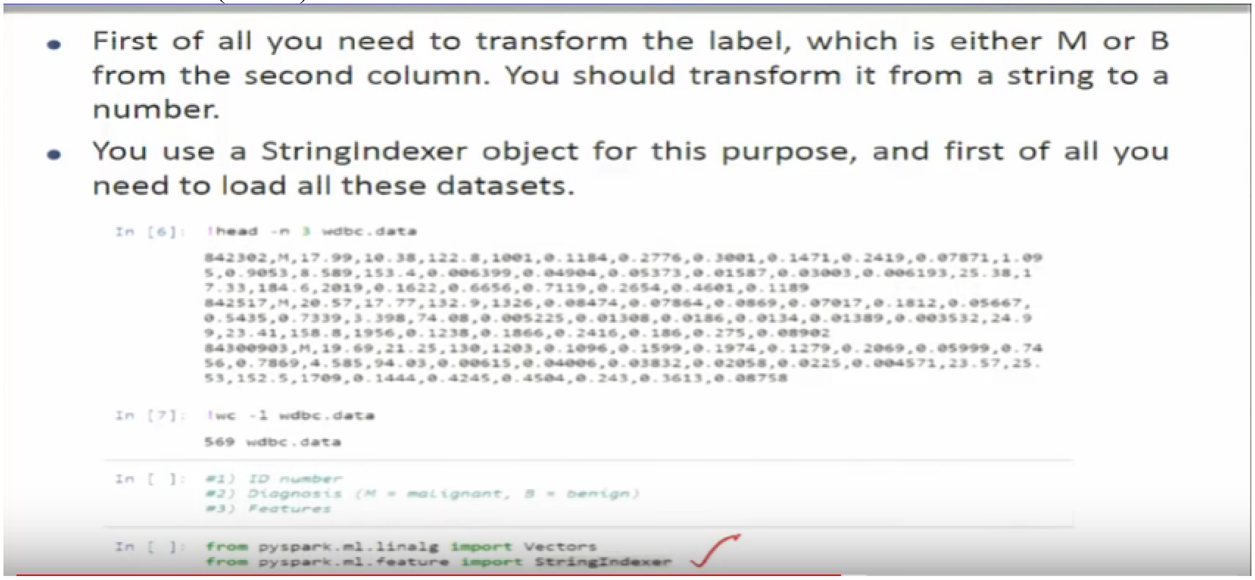


- So these features are results of some analysis and measurements. There are total 569 examples in this dataset.

```
In [6]:  !head -n 3 wdbc.data

         842302,M,17.99,10.38,122.8,1001,0.1184,0.2776,0.3001,0.1471,0.2419,0.07871,1.09
         5,0.9053,8.589,153.4,0.006399,0.04904,0.05373,0.01587,0.03003,0.006193,25.38,1
         7.33,184.6,2019,0.1622,0.6656,0.7119,0.2654,0.4601,0.1189
         842517,M,20.57,17.77,132.9,1326,0.08474,0.07864,0.0869,0.07017,0.1812,0.05667,
         0.5435,0.7339,3.398,74.08,0.005225,0.01308,0.0186,0.0134,0.01389,0.003532,24.9
         9,23.41,158.8,1956,0.1238,0.1866,0.2416,0.186,0.275,0.08902
         84300903,M,19.69,21.25,130,1203,0.1096,0.1599,0.1974,0.1279,0.2069,0.05999,0.74
         56,0.7869,4.585,94.03,0.00615,0.04006,0.03832,0.02058,0.0225,0.004571,23.57,25.
         53,152.5,1709,0.1444,0.4245,0.4504,0.243,0.3613,0.08758

In [7]:  !wc -l wdbc.data

         569 wdbc.data

In [ ]:  #1) ID number
         #2) Diagnosis (M = malignant, B = benign)
         #3) Features

In [ ]:  from pyspark.ml.linalg import Vectors
         from pyspark.ml.feature import StringIndexer
```

You can see that, these data set. How, so these data sets, so these features are the result of the analysis and it has around 569 different examples, in the data set.

Refer Slide Time :(27:24)

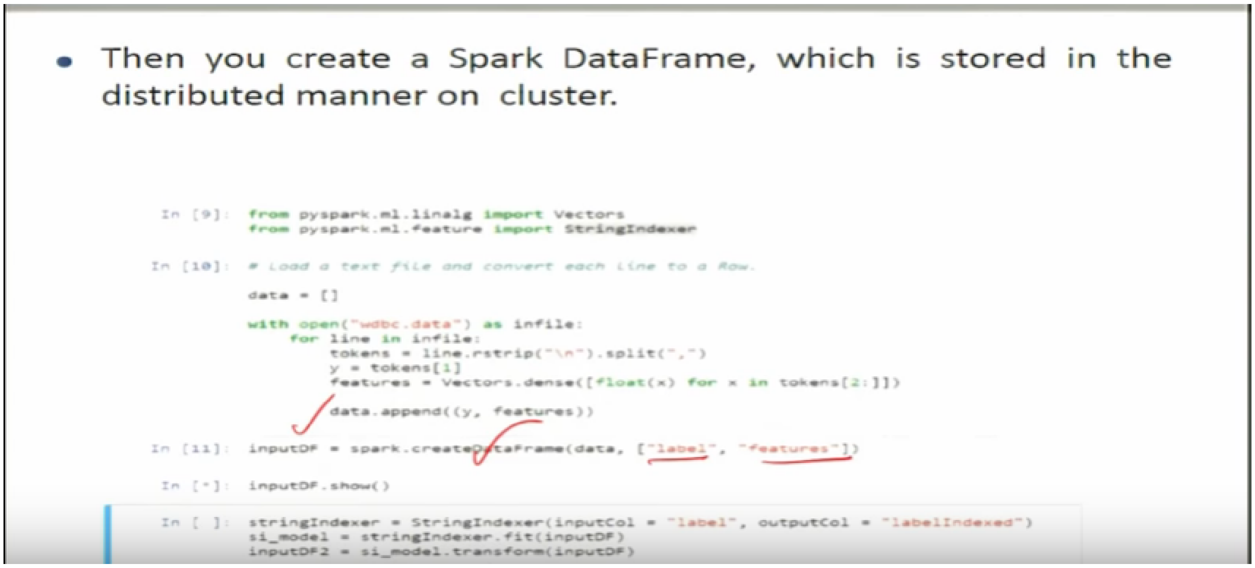


- First of all you need to transform the label, which is either M or B from the second column. You should transform it from a string to a number.
- You use a StringIndexer object for this purpose, and first of all you need to load all these datasets.

```
In [6]:  !head -n 3 wdbc.data

         842302,M,17.99,10.38,122.8,1001,0.1184,0.2776,0.3001,0.1471,0.2419,0.07871,1.09
         5,0.9053,8.589,153.4,0.006399,0.04904,0.05373,0.01587,0.03003,0.006193,25.38,1
         7.33,184.6,2019,0.1622,0.6656,0.7119,0.2654,0.4601,0.1189
         842517,M,20.57,17.77,132.9,1326,0.08474,0.07864,0.0869,0.07017,0.1812,0.05667,
         0.5435,0.7339,3.398,74.08,0.005225,0.01308,0.0186,0.0134,0.01389,0.003532,24.9
         9,23.41,158.8,1956,0.1238,0.1866,0.2416,0.186,0.275,0.08902
         84300903,M,19.69,21.25,130,1203,0.1096,0.1599,0.1974,0.1279,0.2069,0.05999,0.74
         56,0.7869,4.585,94.03,0.00615,0.04006,0.03832,0.02058,0.0225,0.004571,23.57,25.
         53,152.5,1709,0.1444,0.4245,0.4504,0.243,0.3613,0.08758

In [7]:  !wc -l wdbc.data

         569 wdbc.data

In [ ]:  #1) ID number
         #2) Diagnosis (M = malignant, B = benign)
         #3) Features

In [ ]:  from pyspark.ml.linalg import Vectors
         from pyspark.ml.feature import StringIndexer
```

And what we, we all need to transform this label, which is given in M or B to the, to the numeric value and using string index or object, we can do this and that is the string index our object will create this, into the label values.

Refer Slide Time :(27:48)

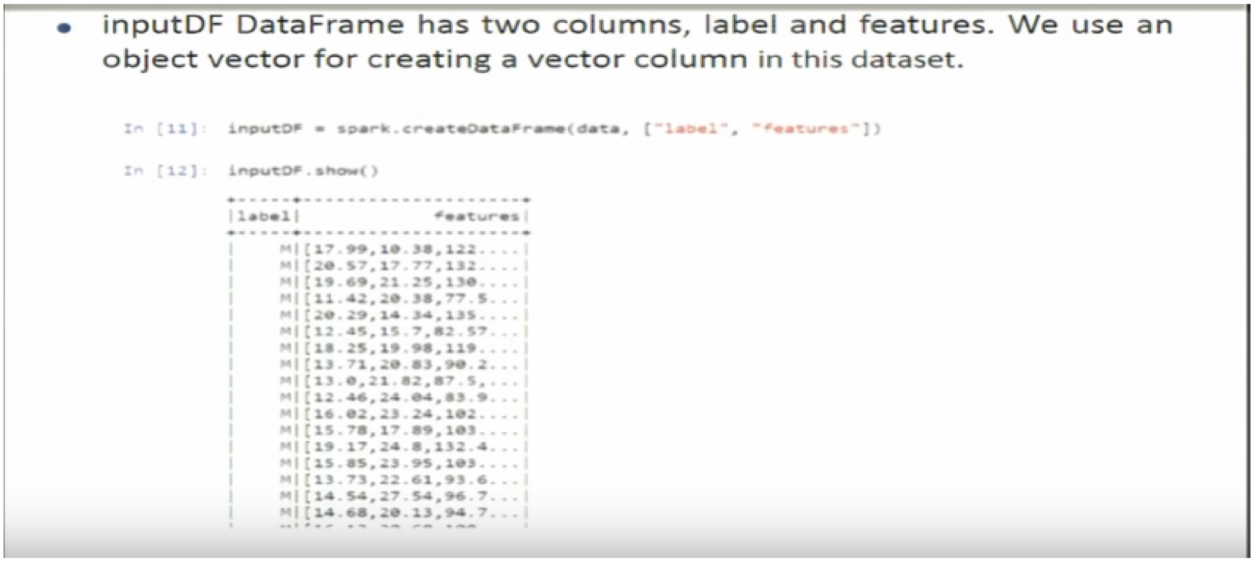


- Then you create a Spark DataFrame, which is stored in the distributed manner on cluster.

```
In [9]:  from pyspark.ml.linalg import Vectors
          from pyspark.ml.feature import StringIndexer

In [10]: # Load a text file and convert each line to a Row.

          data = []

          with open("wdbc.data") as infile:
              for line in infile:
                  tokens = line.rstrip("\n").split(",")
                  y = tokens[1]
                  features = Vectors.dense([float(x) for x in tokens[2:]])

                  data.append((y, features))

In [11]: inputDF = spark.createDataFrame(data, ["label", "features"])

In [*]:  inputDF.show()

In [ ]:  stringIndexer = StringIndexer(inputCol = "label", outputCol = "labelIndexed")
          si_model = stringIndexer.fit(inputDF)
          inputDF2 = si_model.transform(inputDF)
```

Into the numeric values and then we can create the data frame, which is stored in the distributed manner on the on the cluster. So, this particular data frame will be, created with the label and the feature, so these two part will be used up in the data frame, which will be input to our system. So, string indexer and then it will transform,

Refer Slide Time :(28:11)

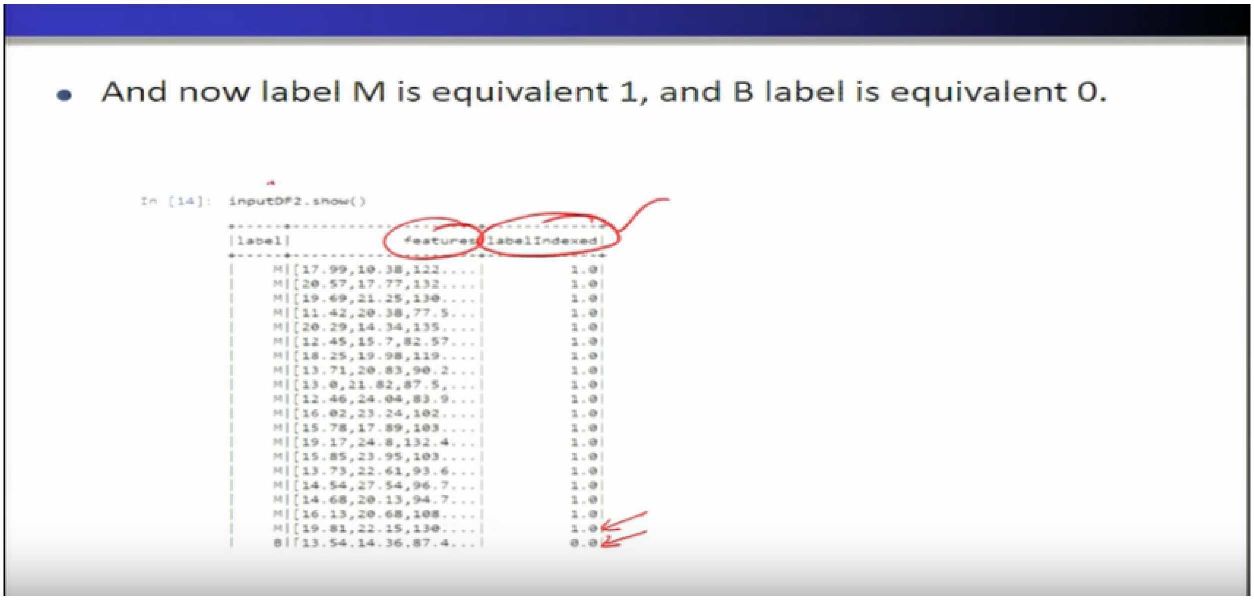


- inputDF DataFrame has two columns, label and features. We use an object vector for creating a vector column in this dataset.

```
In [11]: inputDF = spark.createDataFrame(data, ["label", "features"])

In [12]: inputDF.show()

         +-----+--------------------+
         |label|            features|
         +-----+--------------------+
         |    M|[17.99,10.38,122....|
         |    M|[20.57,17.77,132....|
         |    M|[19.69,21.25,130....|
         |    M|[11.42,20.38,77.5...|
         |    M|[20.29,14.34,135....|
         |    M|[12.45,15.7,82.57...|
         |    M|[18.25,19.98,119....|
         |    M|[13.71,20.83,90.2...|
         |    M|[13.0,21.82,87.5,...|
         |    M|[12.46,24.04,83.9...|
         |    M|[16.02,23.24,102....|
         |    M|[15.78,17.89,103....|
         |    M|[19.17,24.8,132.4...|
         |    M|[15.85,23.95,103....|
         |    M|[13.73,22.61,93.6...|
         |    M|[14.54,27.54,96.7...|
         |    M|[14.68,20.13,94.7...|
```

so that, all the values will become here. So, all these labels will be converted into the values using string indexer.

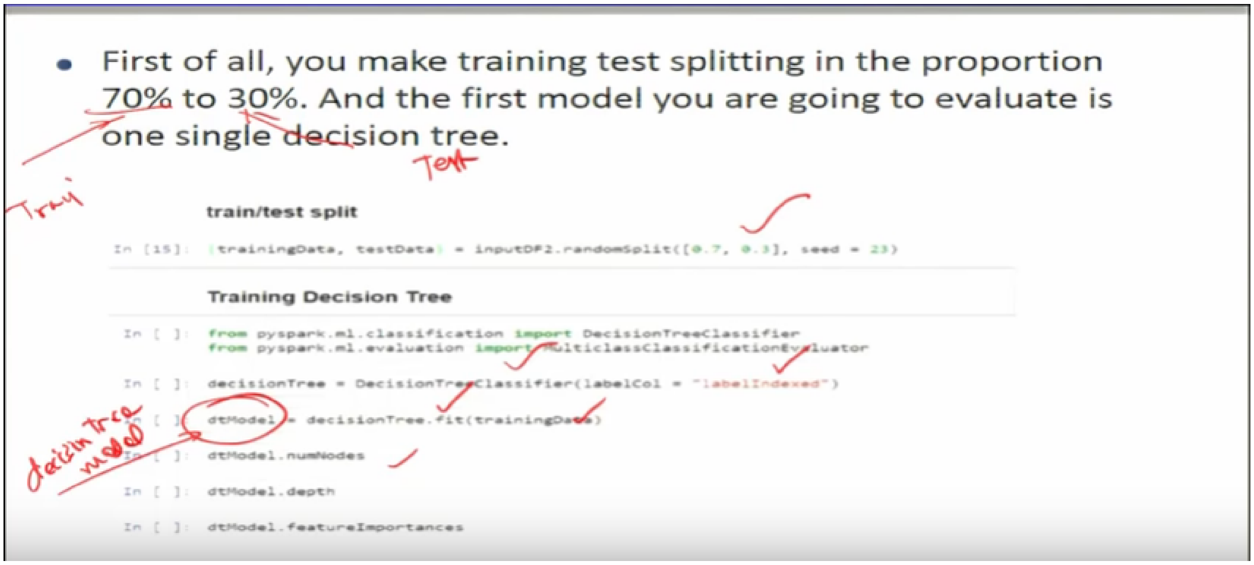
Refer Slide Time :(28:18)



- And now label M is equivalent 1, and B label is equivalent 0.

```
In [14]:  inputDF2.show()

          +-----+--------------------+------------+
          |label|            features|labelIndexed|
          +-----+--------------------+------------+
          |    M|[17.99,10.38,122....|         1.0|
          |    M|[20.57,17.77,132....|         1.0|
          |    M|[19.69,21.25,130....|         1.0|
          |    M|[11.42,20.38,77.5...|         1.0|
          |    M|[20.29,14.34,135....|         1.0|
          |    M|[12.45,15.7,82.57...|         1.0|
          |    M|[18.25,19.98,119....|         1.0|
          |    M|[13.71,20.83,90.2...|         1.0|
          |    M|[13.0,21.82,87.5,...|         1.0|
          |    M|[12.46,24.04,83.9...|         1.0|
          |    M|[16.02,23.24,102....|         1.0|
          |    M|[15.78,17.89,103....|         1.0|
          |    M|[19.17,24.8,132.4...|         1.0|
          |    M|[15.85,23.95,103....|         1.0|
          |    M|[13.73,22.61,93.6...|         1.0|
          |    M|[14.54,27.54,96.7...|         1.0|
          |    M|[14.68,20.13,94.7...|         1.0|
          |    M|[16.13,20.68,108....|         1.0|
          |    M|[19.81,22.15,130....|         1.0|
          |    B|[13.54,14.36,87.4...|         0.0|
```

So now, after index, label index and all these label values are converted, from letter to the numeric values. So, here can have this numeric value either 1 or it is 0, as the label values. So, we have now two important things, one is the feature and the other is label indexed, in the data frame which is shown over here, this particular data frame will be now used for further analysis.
Refer Slide Time :(28:55)



- First of all, you make training test splitting in the proportion 70% to 30%. And the first model you are going to evaluate is one single decision tree.

train/test split

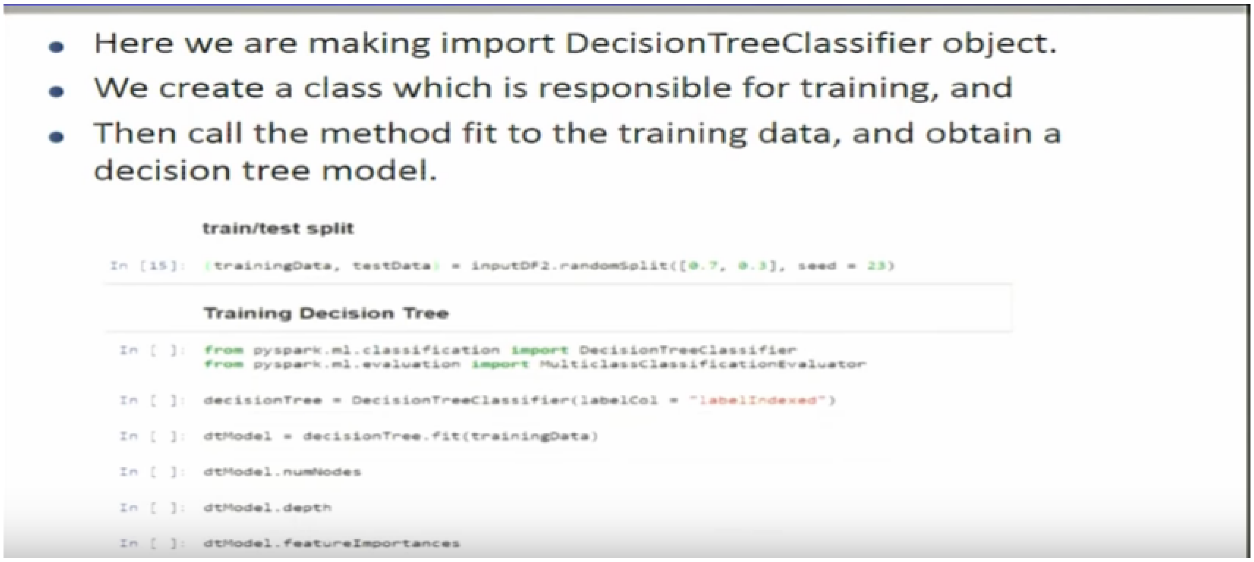
```
In [15]:  (trainingData, testData) = inputDF2.randomSplit([0.7, 0.3], seed = 23)
```

Training Decision Tree

```
In [ ]:  from pyspark.ml.classification import DecisionTreeClassifier
         from pyspark.ml.evaluation import MulticlassClassificationEvaluator

In [ ]:  decisionTree = DecisionTreeClassifier(labelCol = "labelIndexed")

In [ ]:  dtModel = decisionTree.fit(trainingData)

In [ ]:  dtModel.numNodes

In [ ]:  dtModel.depth

In [ ]:  dtModel.featureImportances
```

Now, then we will make the training test is splitting, into the proportion 70 to 30 that when 70% is the training portion and 30% will be the test portion of the data set. So, first we model with that 70% data set to train the model on a single decision tree and then we will use the 30% of the data set for testing purpose. So, this is to be done in this particular manner using random split, of 70 by 30 and then we will train the decision tree model, using decision tree classifier and we'll apply the label in text, in this particular case and then we will fit, the decision tree on the training data set. So, this will become this will build a model called, ‘Decision Tree Model ‘. It will build and this will be represented as DT model, as the variable. Now, you can see this particular model has how many nodes? What is the depth? And what are the important features? And so on.

Refer Slide Time :(30:14)



- Here we are making import DecisionTreeClassifier object.
- We create a class which is responsible for training, and
- Then call the method fit to the training data, and obtain a decision tree model.

train/test split

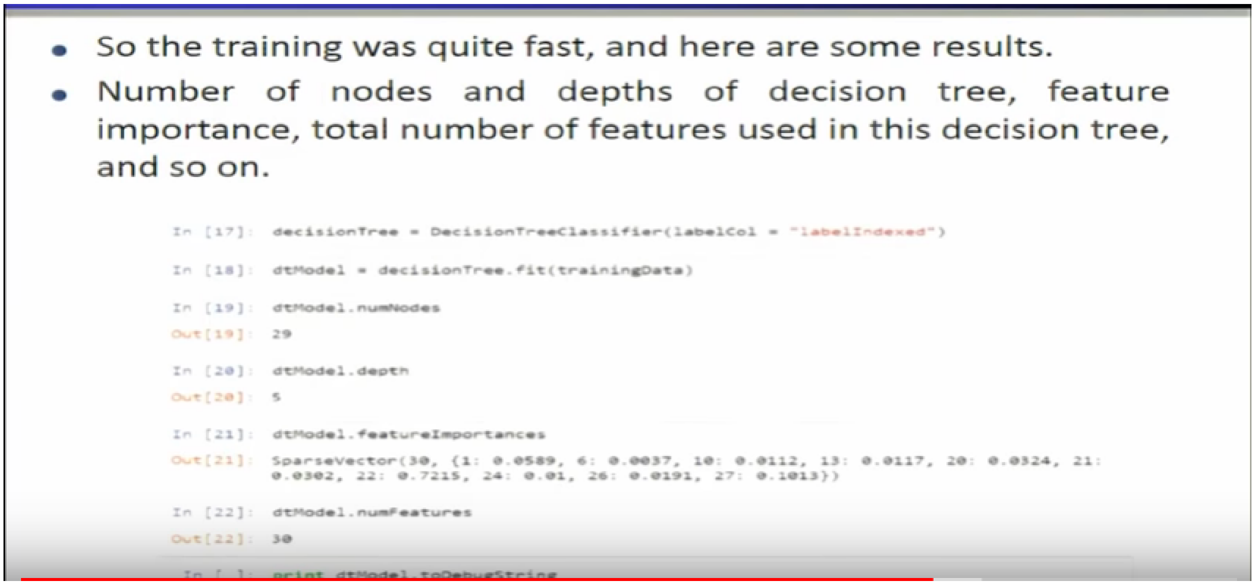
```
In [15]: (trainingData, testData) = inputDF2.randomSplit([0.7, 0.3], seed = 23)
```

Training Decision Tree

```
In [ ]: from pyspark.ml.classification import DecisionTreeClassifier
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator

In [ ]: decisionTree = DecisionTreeClassifier(labelCol = "labelIndexed")

In [ ]: dtModel = decisionTree.fit(trainingData)

In [ ]: dtModel.numNodes

In [ ]: dtModel.depth

In [ ]: dtModel.featureImportances
```

And so after the, so these things now we are making, the import decision tree classifier we can create the class that is responsible for the training and then we call the, ‘Method fit’, to the training data and obtain the decision tree model that we have shown.

Refer Slide Time :(30:36)

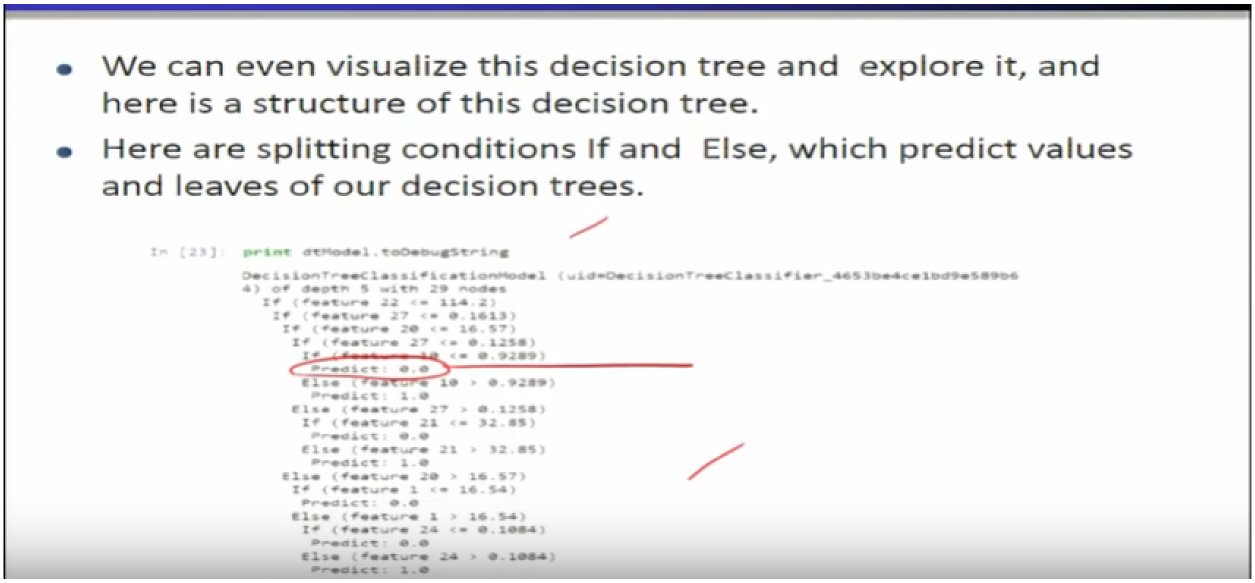


- So the training was quite fast, and here are some results.
- Number of nodes and depths of decision tree, feature importance, total number of features used in this decision tree, and so on.

```
In [17]: decisionTree = DecisionTreeClassifier(labelCol = "labelIndexed")

In [18]: dtModel = decisionTree.fit(trainingData)

In [19]: dtModel.numNodes
Out[19]: 29

In [20]: dtModel.depth
Out[20]: 5

In [21]: dtModel.featureImportances
Out[21]: SparseVector(30, {1: 0.0589, 6: 0.0037, 10: 0.0112, 13: 0.0117, 20: 0.0324, 21:
         0.0302, 22: 0.7215, 24: 0.01, 26: 0.0191, 27: 0.1013})

In [22]: dtModel.numFeatures
Out[22]: 30

In [ ]: print dtModel.toDebugString
```

Now, so the training was done in a very fast manner and now the we have to see the results, so the number of nodes and depth out the decision tree and a feature importance and number of features used in the decision tree and so on. These different things we can inspect.

Refer Slide Time :(31:01)

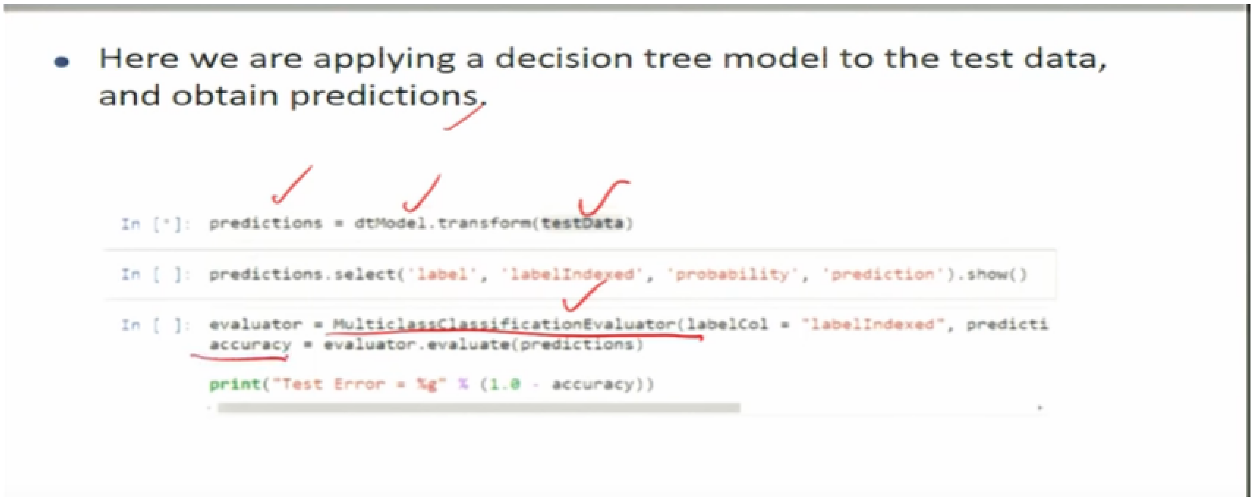


- We can even visualize this decision tree and explore it, and here is a structure of this decision tree.
- Here are splitting conditions If and Else, which predict values and leaves of our decision trees.

```
In [23]:  print dtModel.toDebugString
          DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4653be4ce1bd9e589b6
          4) of depth 5 with 29 nodes
            If (feature 22 <= 114.2)
             If (feature 27 <= 0.1613)
              If (feature 20 <= 16.57)
               If (feature 27 <= 0.1258)
                If (feature 10 <= 0.9289)
                 Predict: 0.0
                Else (feature 10 > 0.9289)
                 Predict: 1.0
               Else (feature 27 > 0.1258)
                If (feature 21 <= 32.85)
                 Predict: 0.0
                Else (feature 21 > 32.85)
                 Predict: 1.0
              Else (feature 20 > 16.57)
               If (feature 1 <= 16.54)
                Predict: 0.0
               Else (feature 1 > 16.54)
                If (feature 24 <= 0.1084)
                 Predict: 0.0
                Else (feature 24 > 0.1084)
                 Predict: 1.0
```

And now, we are going to see the decision tree and we can take this one, so we can print this decision, decision tree model and you can see that, it is nothing but an if-then-else and then that means all the internal nodes of a DC entry and finally the leap will be having the predictions. So, once the decision tree is built,

Refer Slide Time :(31:23)

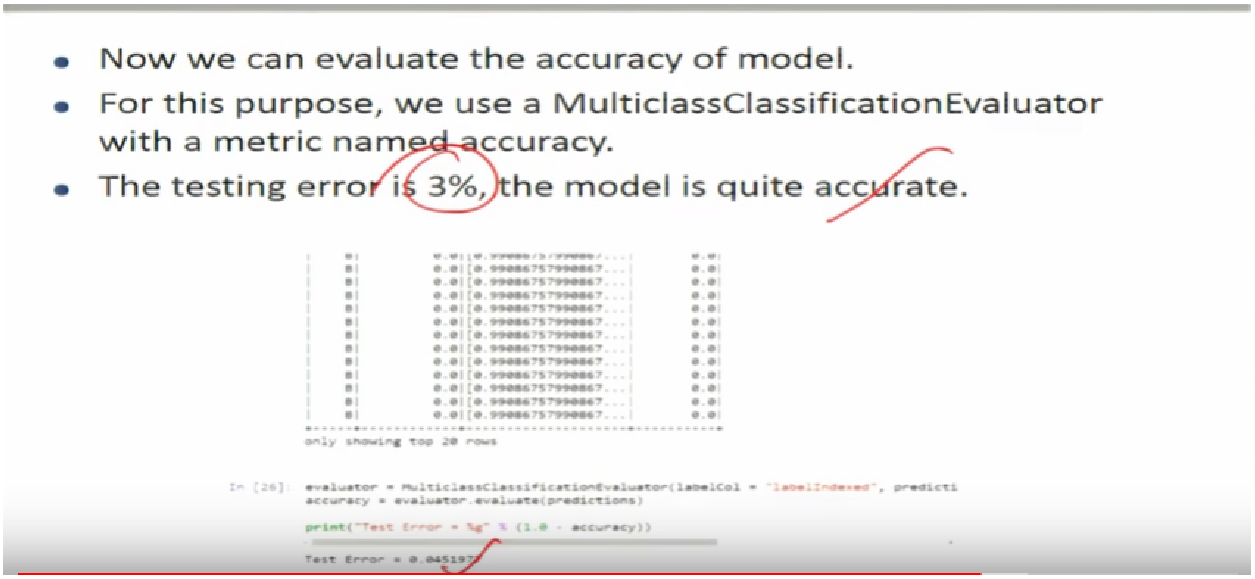


- Here we are applying a decision tree model to the test data, and obtain predictions.

```
In [*]:  predictions = dtModel.transform(testData)

In [ ]:  predictions.select('label', 'labelIndexed', 'probability', 'prediction').show()

In [ ]:  evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti
         accuracy = evaluator.evaluate(predictions)

         print("Test Error = %g" % (1.0 - accuracy))
```

now we can use this decision tree, to for the test data, to obtain the predictions. So, now we can use the test data and use the same decision tree model and now, we will perform the, the predictions. So, so the predictions are now, there and then we will evaluate this particular, the predictions using multi class, multi class classifier evaluator, we will evaluate these predictions, which are done by this one decision tree for its accuracy. So, what we will find here is that?

Refer Slide Time :(32:08)

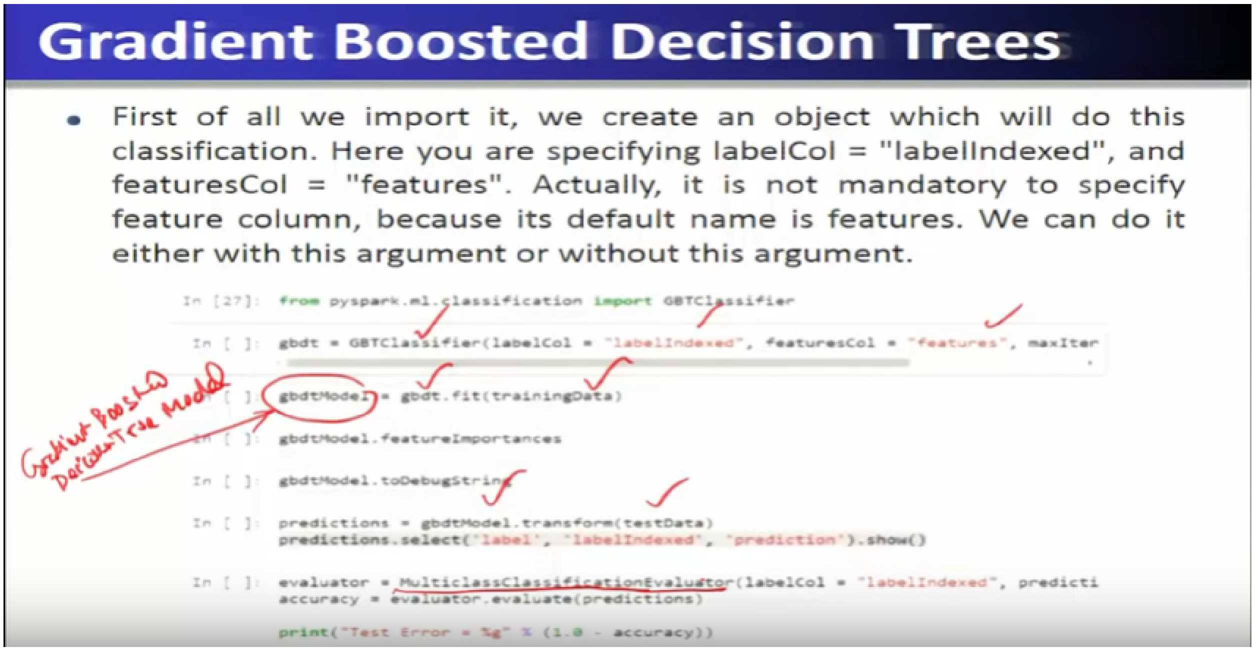


- Now we can evaluate the accuracy of model.
- For this purpose, we use a MulticlassClassificationEvaluator with a metric named accuracy.
- The testing error is 3%, the model is quite accurate.

```
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
|       0|          0.0|[0.99086757990867...|       0.0|
+--------+-------------+--------------------+----------+
only showing top 20 rows

In [26]: evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti
         accuracy = evaluator.evaluate(predictions)

         print("Test Error = %g" % (1.0 - accuracy))

         Test Error = 0.045197
```

That accuracy is very high and the error is very less. And so, the testing error is only thirty percent, so the model is quite accurate. So, with only three percent error, the model is quite accurate.

Refer Slide Time :(32:28)



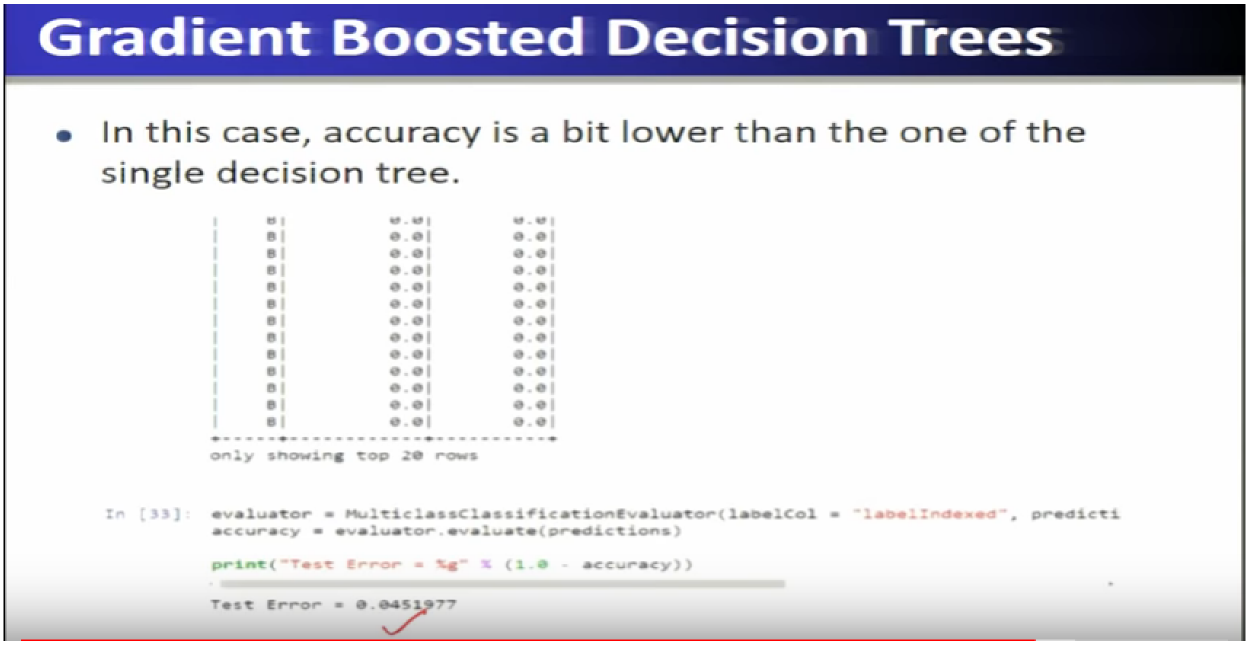
Gradient Boosted Decision Trees

- First of all we import it, we create an object which will do this classification. Here you are specifying labelCol = "labelIndexed", and featuresCol = "features". Actually, it is not mandatory to specify feature column, because its default name is features. We can do it either with this argument or without this argument.

```
In [27]: from pyspark.ml.classification import GBTClassifier

In [ ]:  gbdt = GBTClassifier(labelCol = "labelIndexed", featuresCol = "features", maxIter

In [ ]:  gbdtModel = gbdt.fit(trainingData)

In [ ]:  gbdtModel.featureImportances

In [ ]:  gbdtModel.toDebugString

In [ ]:  predictions = gbdtModel.transform(testData)
         predictions.select('label', 'labelIndexed', 'prediction').show()

In [ ]:  evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti
         accuracy = evaluator.evaluate(predictions)

         print("Test Error = %g" % (1.0 - accuracy))
```

And now, we are going to see the another method, which is called a, ‘Gradient Boosted Decision Trees’. So, first of all we import, the data and then create the object which will do the classification. And here, we are specifying the label column is a labelindexed that we have seen, in the previous example also and the features column as the features. Actually, it is not mandatory to specify the feature column, we can do it either using the assignment or without this argument, now the thing is, so after having done this, now we will fit this particular model and apply this particular gradient boosted, classifier using labelindexed the features that we have seen, now we will fit this particular model, onto the training data set and we will get the model, gradient boosted a decision tree model. So, once we obtain the model, then we will see, its where then we will using this particular model, we will now apply the test data on it this particular model and now, we using a multi-class classification evaluator, we will evaluate its predictions.

Refer Slide Time :(34:09)



Gradient Boosted Decision Trees

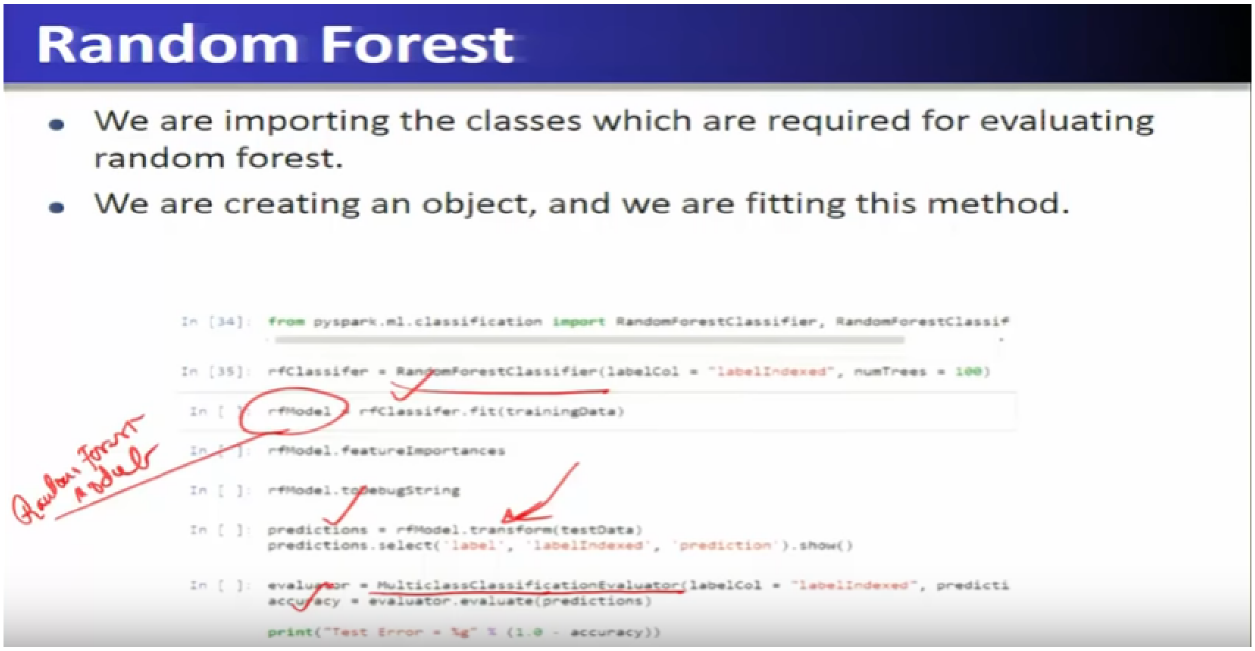
- In this case, accuracy is a bit lower than the one of the single decision tree.

```
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
|     8|          0.0|        0.0|
+------+-------------+-----------+
only showing top 20 rows
```

```
In [33]: evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti
         accuracy = evaluator.evaluate(predictions)

         print("Test Error = %g" % (1.0 - accuracy))

         Test Error = 0.0451977
```

So, let us see, so this we are going to do 100 iterations and default step is point 0.1. So, after that, so we will see that here the accuracy, here the error is here the, the test error is quite. So, basically this has improved or the DC entry model.

Refer Slide Time :(34:33)



Random Forest

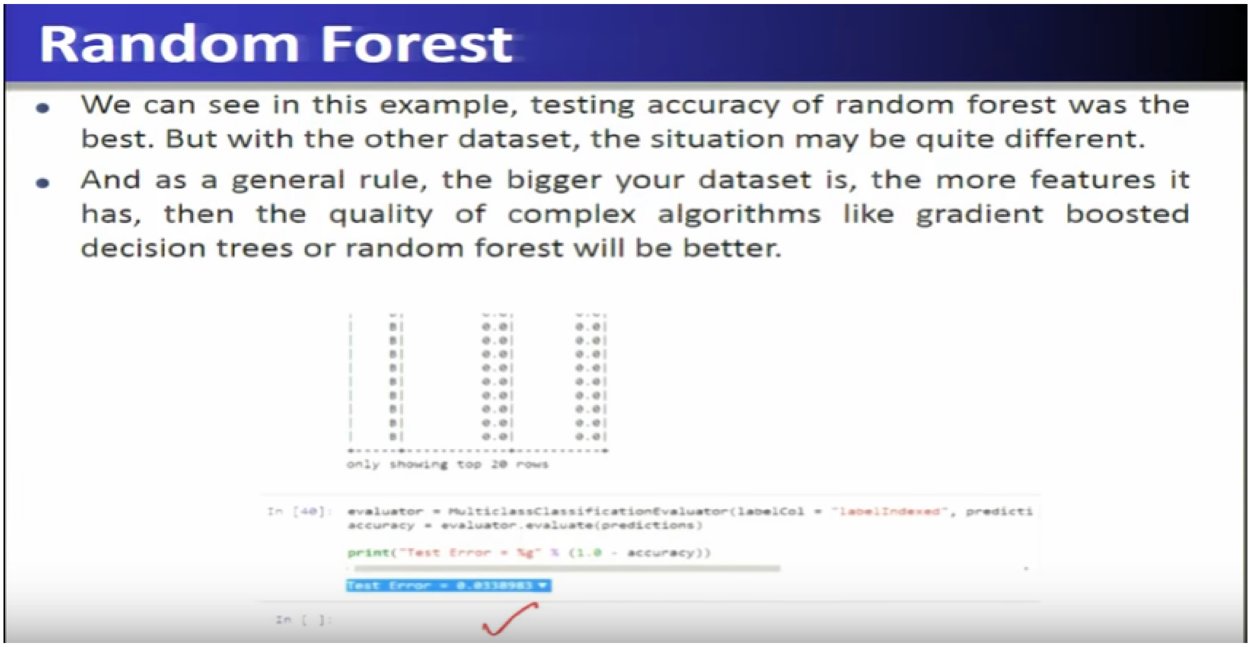
- We are importing the classes which are required for evaluating random forest.
- We are creating an object, and we are fitting this method.

```
In [34]: from pyspark.ml.classification import RandomForestClassifier, RandomForestClassif

In [35]: rfClassifer = RandomForestClassifier(labelCol = "labelIndexed", numTrees = 100)

In [ ]: rfModel = rfClassifer.fit(trainingData)

In [ ]: rfModel.featureImportances

In [ ]: rfModel.toDebugString

In [ ]: predictions = rfModel.transform(testData)
        predictions.select('label', 'labelIndexed', 'prediction').show()

In [ ]: evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti
        accuracy = evaluator.evaluate(predictions)

        print("Test Error = %g" % (1.0 - accuracy))
```

Random Forest model

Now, we are going to see the random forest. And random forest again, the same data set we will take and but here, the classifier we are going to change as, random forest classifier and we will fit, the training data on random forest classifier and we will get a model, which is called, 'Random Forest', 'Forest Model'. Now, using this particular model we will, now we will transform the test data, we will apply the test it on this particular model and get the predictions. So, now after getting the predictions we are not evaluating these particular predictions, using multi-class classification evaluator and this will now, evaluate its prediction accuracies.

Refer Slide Time :(35:28)



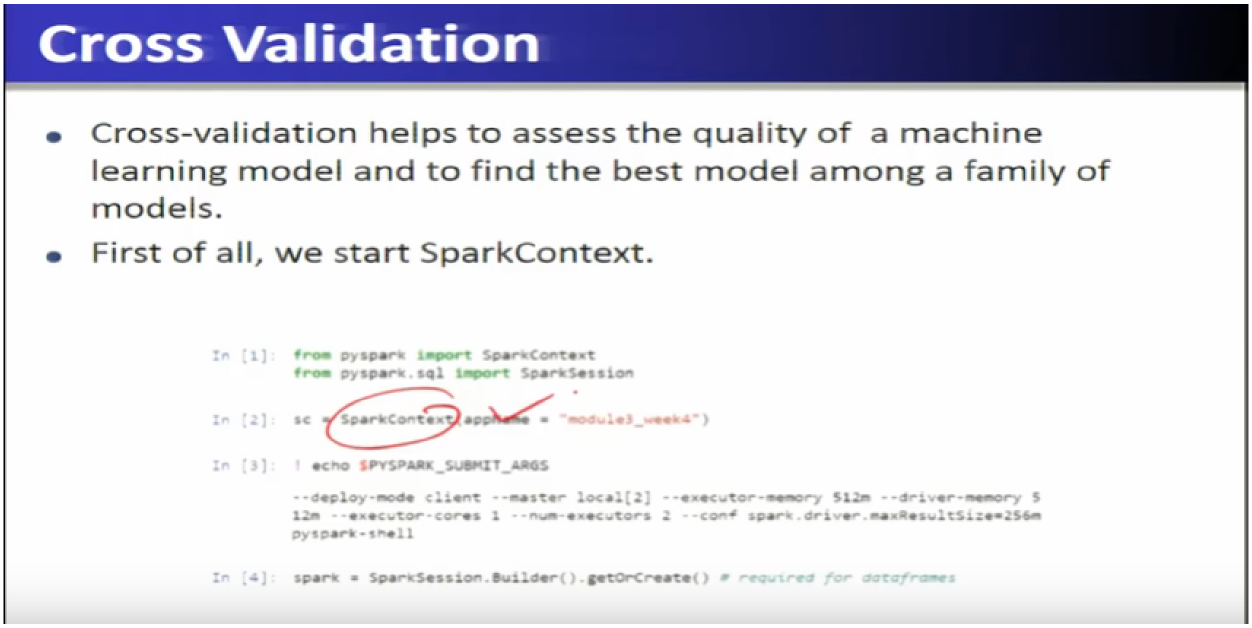
Random Forest

- We can see in this example, testing accuracy of random forest was the best. But with the other dataset, the situation may be quite different.
- And as a general rule, the bigger your dataset is, the more features it has, then the quality of complex algorithms like gradient boosted decision trees or random forest will be better.

```
|    0|         0.0|        0.0|
|    0|         0.0|        0.0|
|    0|         0.0|        0.0|
|    0|         0.0|        0.0|
|    0|         0.0|        0.0|
|    0|         0.0|        0.0|
|    0|         0.0|        0.0|
|    0|         0.0|        0.0|
|    0|         0.0|        0.0|
+-----+------------+-----------+
only showing top 20 rows

In [40]: evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti
         accuracy = evaluator.evaluate(predictions)

         print("Test Error = %g" % (1.0 - accuracy))

Test Error = 0.0338983

In [ ]:
```

And this we will now, be able to see that, that the test error is very less that is, than the previous decision trees. Hence, this we can see that, in this example, the testing accuracy of the random forest was the best, with the other data the situation may be quite different. In general case, the bigger your data set is more features, it has the quality of complex algorithm like gradient boosted or the forest will be much better.

Refer Slide Time :(36:02)



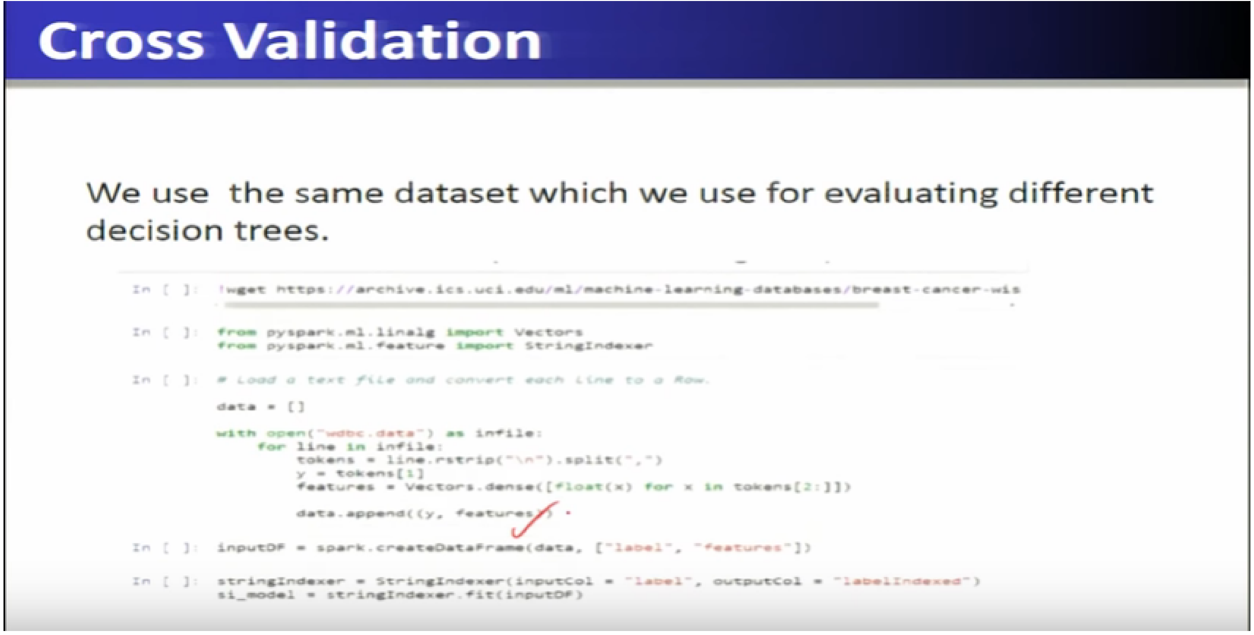
Cross Validation

- Cross-validation helps to assess the quality of a machine learning model and to find the best model among a family of models.
- First of all, we start SparkContext.

```
In [1]:  from pyspark import SparkContext
         from pyspark.sql import SparkSession

In [2]:  sc = SparkContext(appName = "module3_week4")

In [3]:  ! echo $PYSPARK_SUBMIT_ARGS

         --deploy-mode client --master local[2] --executor-memory 512m --driver-memory 5
         12m --executor-cores 1 --num-executors 2 --conf spark.driver.maxResultSize=256m
         pyspark-shell

In [4]:  spark = SparkSession.Builder().getOrCreate() # required for dataframes
```

Now, let us see the cross-validation using spark ml. So, class validation helps to assess, the quality of machine learning model and to find the best model, among the family of the models. First of all we have to create the spark context that is, shown over here and then we will, do the spark session.

Refer Slide Time :(36:27)



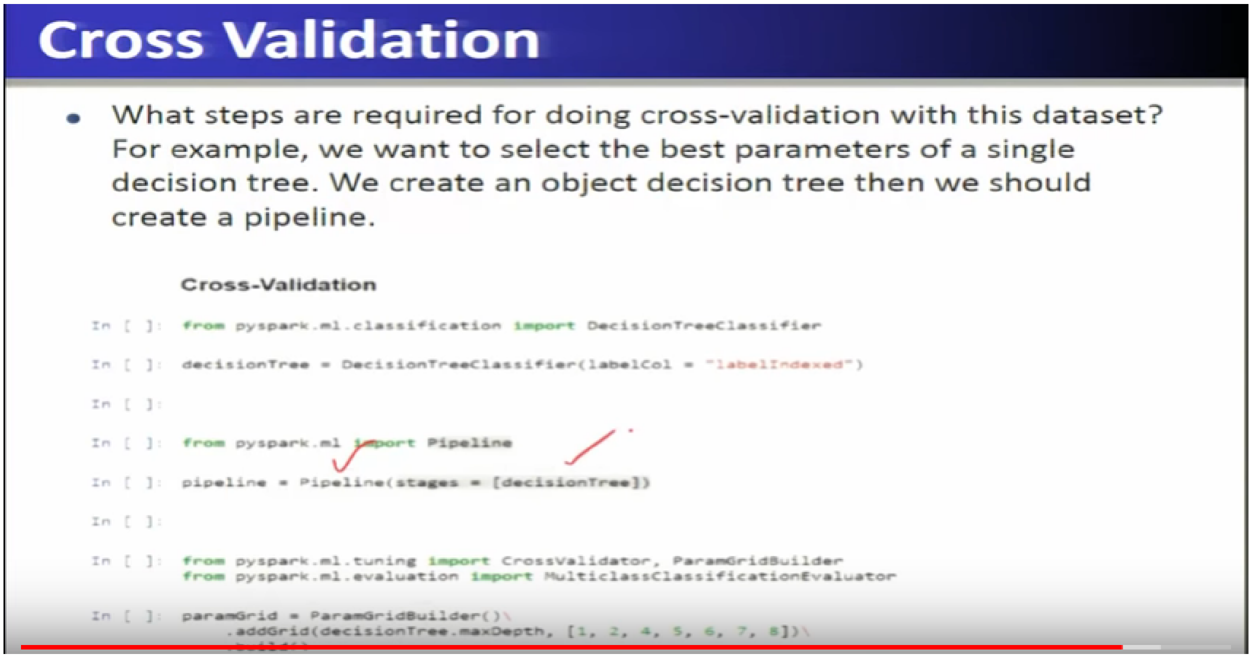
Cross Validation

We use the same dataset which we use for evaluating different decision trees.

```
In [ ]:  !wget https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wis

In [ ]:  from pyspark.ml.linalg import Vectors
         from pyspark.ml.feature import StringIndexer

In [ ]:  # Load a text file and convert each line to a Row.

         data = []

         with open("wdbc.data") as infile:
             for line in infile:
                 tokens = line.rstrip("\n").split(",")
                 y = tokens[1]
                 features = Vectors.dense([float(x) for x in tokens[2:]])

                 data.append((y, features))

In [ ]:  inputDF = spark.createDataFrame(data, ["label", "features"])

In [ ]:  stringIndexer = StringIndexer(inputCol = "label", outputCol = "labelIndexed")
         si_model = stringIndexer.fit(inputDF)
```

Then we'll build the spark session, will build the spark session, which will create the data frames and then we will use the data set, read the data set, load the data file and now, we will create the data frames, DF as the input data frame.

Refer Slide Time :(36:50)



Cross Validation

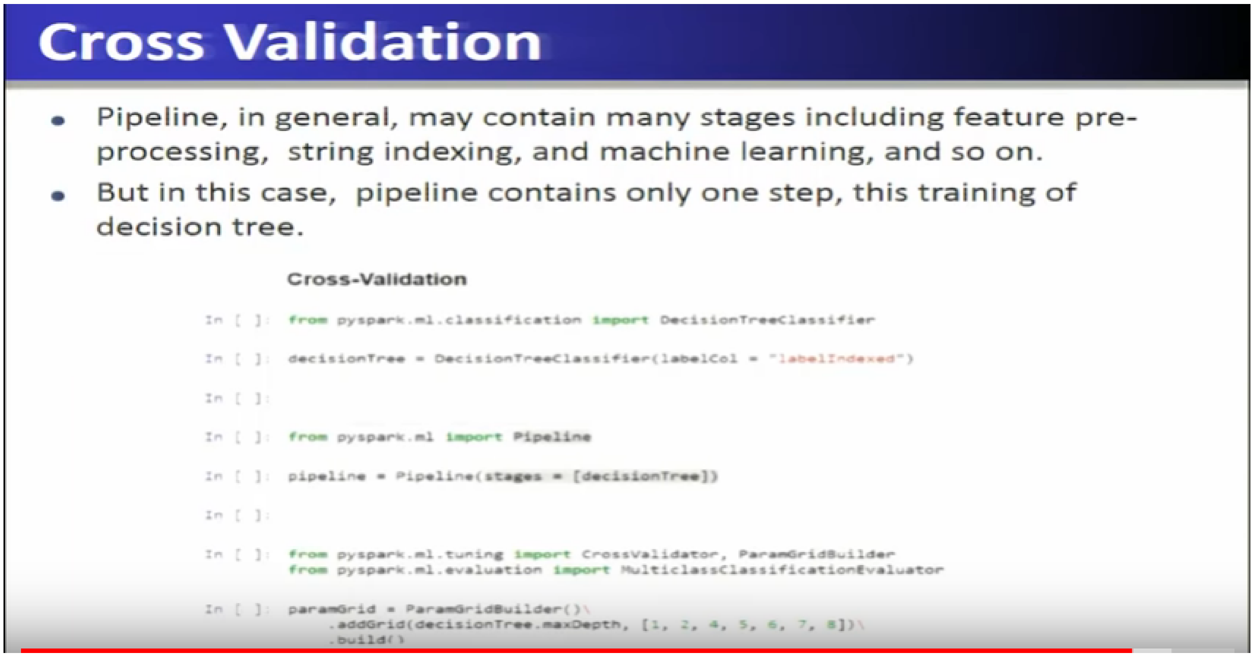
- What steps are required for doing cross-validation with this dataset? For example, we want to select the best parameters of a single decision tree. We create an object decision tree then we should create a pipeline.

Cross-Validation

```
In [ ]: from pyspark.ml.classification import DecisionTreeClassifier

In [ ]: decisionTree = DecisionTreeClassifier(labelCol = "labelIndexed")

In [ ]:

In [ ]: from pyspark.ml import Pipeline

In [ ]: pipeline = Pipeline(stages = [decisionTree])

In [ ]:

In [ ]: from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator

In [ ]: paramGrid = ParamGridBuilder()\
            .addGrid(decisionTree.maxDepth, [1, 2, 4, 5, 6, 7, 8])\
            .build()
```

And now, we will apply we after doing various transformations. Now, we will input data we can see, it is already indexed now we have to do the cross validation. So, what steps are required for doing cross validation with the data set? So, for example, we want to select the best parameters of a single decision tree. And we create the object decision tree, then we should create the pipeline. So here, we are going to create the pipeline and in this pipeline stage, we are only using the decision tree, based pipeline.

Refer Slide Time :(37:25)



Cross Validation

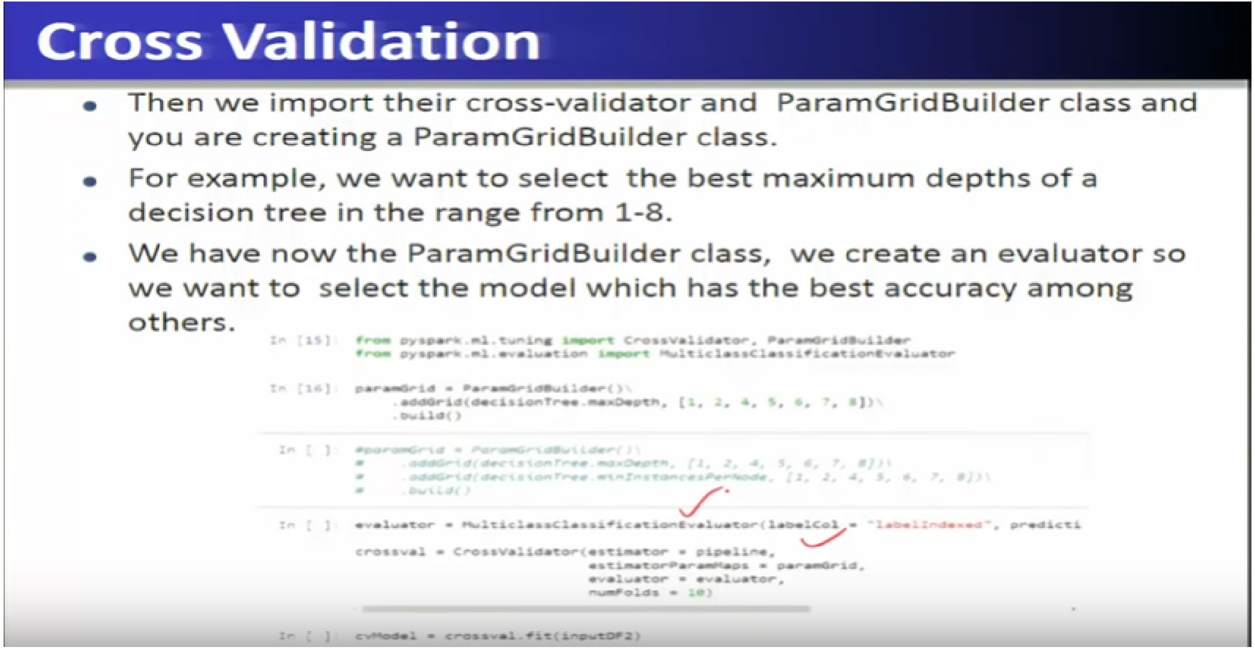
- Pipeline, in general, may contain many stages including feature pre-processing, string indexing, and machine learning, and so on.
- But in this case, pipeline contains only one step, this training of decision tree.

Cross-Validation

```
In [ ]: from pyspark.ml.classification import DecisionTreeClassifier

In [ ]: decisionTree = DecisionTreeClassifier(labelCol = "labelIndexed")

In [ ]:

In [ ]: from pyspark.ml import Pipeline

In [ ]: pipeline = Pipeline(stages = [decisionTree])

In [ ]:

In [ ]: from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator

In [ ]: paramGrid = ParamGridBuilder()\
             .addGrid(decisionTree.maxDepth, [1, 2, 4, 5, 6, 7, 8])\
             .build()
```

So, then we will apply this cross validation, using parameter builder and we have to also, see the cross validation.

Refer Slide Time :(37:39)



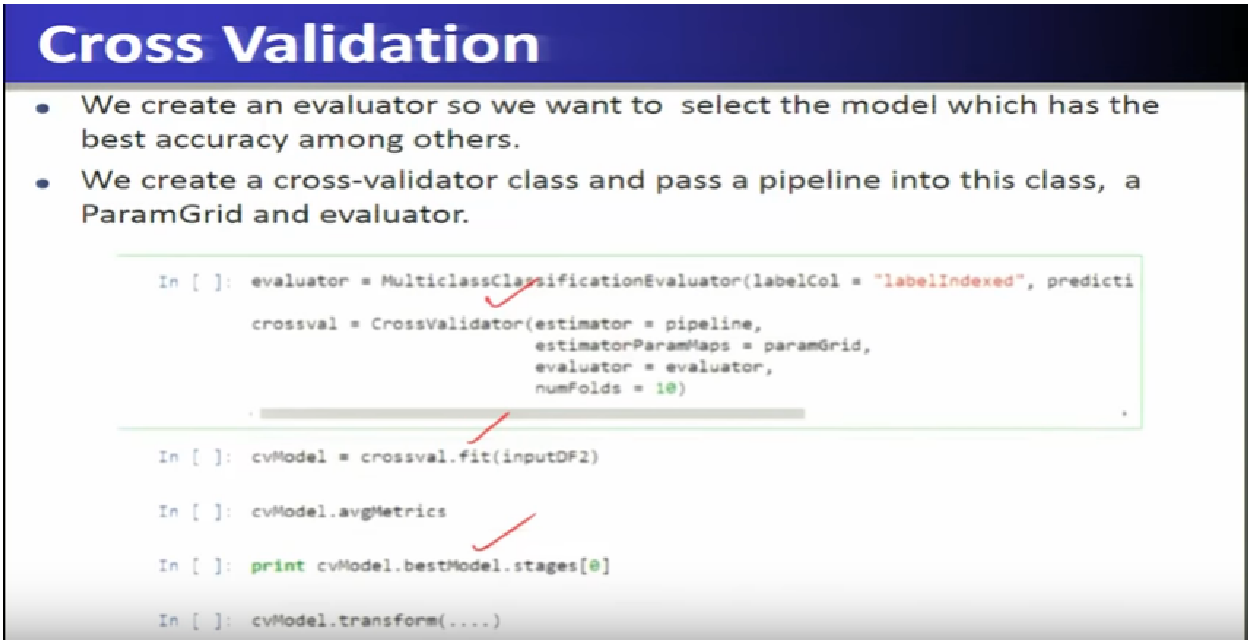
Cross Validation

- Then we import their cross-validator and ParamGridBuilder class and you are creating a ParamGridBuilder class.
- For example, we want to select the best maximum depths of a decision tree in the range from 1-8.
- We have now the ParamGridBuilder class, we create an evaluator so we want to select the model which has the best accuracy among others.

```
In [15]: from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
         from pyspark.ml.evaluation import MulticlassClassificationEvaluator

In [16]: paramGrid = ParamGridBuilder()\
              .addGrid(decisionTree.maxDepth, [1, 2, 4, 5, 6, 7, 8])\
              .build()

In [ ]: #paramGrid = ParamGridBuilder()\
        #     .addGrid(decisionTree.maxDepth, [1, 2, 4, 5, 6, 7, 8])\
        #     .addGrid(decisionTree.minInstancesPerNode, [1, 2, 4, 5, 6, 7, 8])\
        #     .build()

In [ ]: evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti

        crossval = CrossValidator(estimator = pipeline,
                                  estimatorParamMaps = paramGrid,
                                  evaluator = evaluator,
                                  numFolds = 10)

In [ ]: cvModel = crossval.fit(inputDF2)
```

So, then we import their class validator and parameter great, builder class and you are now creating the parameter grid builder class. For example, you want to select the best maximum depth of the decision tree, in the range from 1 to 8. And we don't have the parameter build grid builder class and we create the evaluator, so we want to select, the model that which has the best accuracy among others. And using this, all these parameters whatever we have said, now we are going to evaluate, using multi-class classification evaluator.

Refer Slide Time :(38:22)



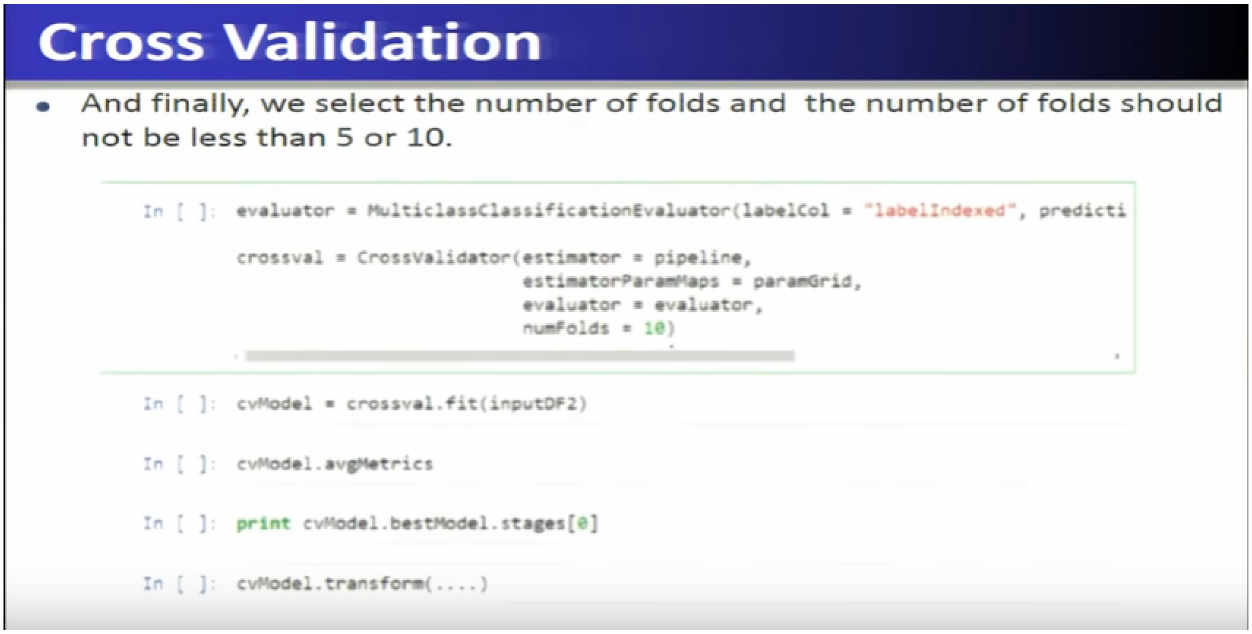
Cross Validation

- We create an evaluator so we want to select the model which has the best accuracy among others.
- We create a cross-validator class and pass a pipeline into this class, a ParamGrid and evaluator.

```
In [ ]: evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti

        crossval = CrossValidator(estimator = pipeline,
                                  estimatorParamMaps = paramGrid,
                                  evaluator = evaluator,
                                  numFolds = 10)

In [ ]: cvModel = crossval.fit(inputDF2)

In [ ]: cvModel.avgMetrics

In [ ]: print cvModel.bestModel.stages[0]

In [ ]: cvModel.transform(....)
```

And now, we will see the evaluator will now, evaluate the cross validator. And cross validator will fit it to the input function and it will select the best model, into the different stages. So, we create the evil water so we want to select the, the model which has the best accuracy among others, so we create a cross validator class and pass a pipeline into, into this class parameter great and the evaluator.

Refer Slide Time :(38:55)



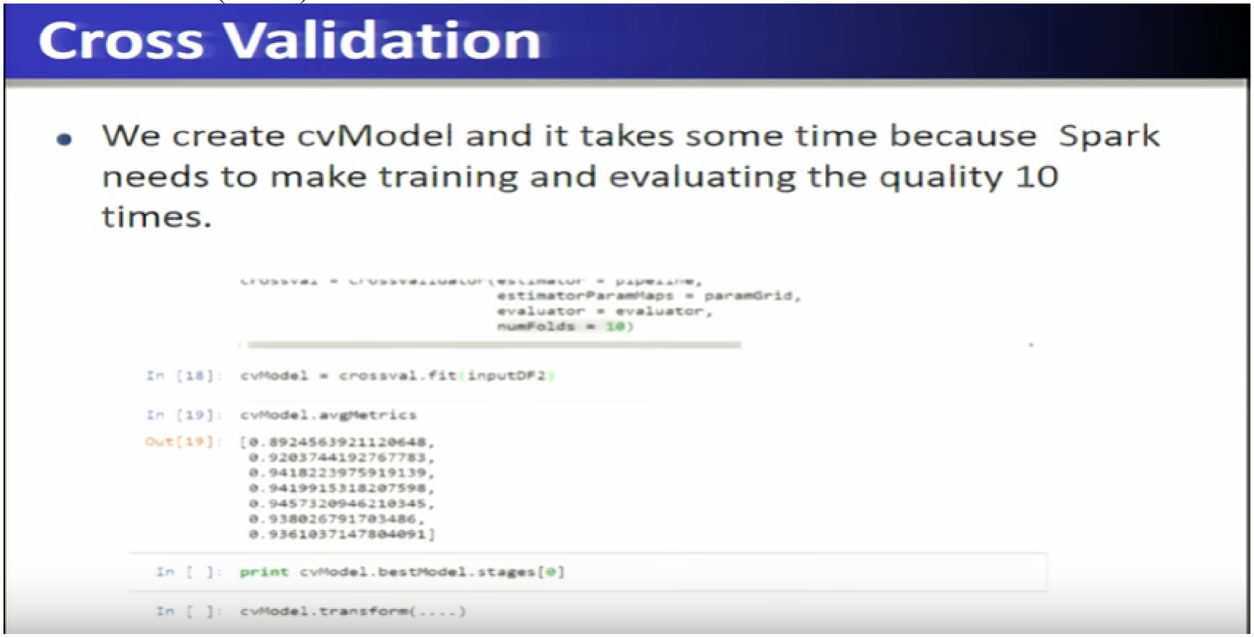
Cross Validation

- And finally, we select the number of folds and the number of folds should not be less than 5 or 10.

```
In [ ]: evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti

        crossval = CrossValidator(estimator = pipeline,
                                  estimatorParamMaps = paramGrid,
                                  evaluator = evaluator,
                                  numFolds = 10)

In [ ]: cvModel = crossval.fit(inputDF2)

In [ ]: cvModel.avgMetrics

In [ ]: print cvModel.bestModel.stages[0]

In [ ]: cvModel.transform(....)
```

And finally, we select the number of folds and the number of holes should be not less than five or ten. And the number of folds we have selected and after that,

Refer Slide Time :(39:07)



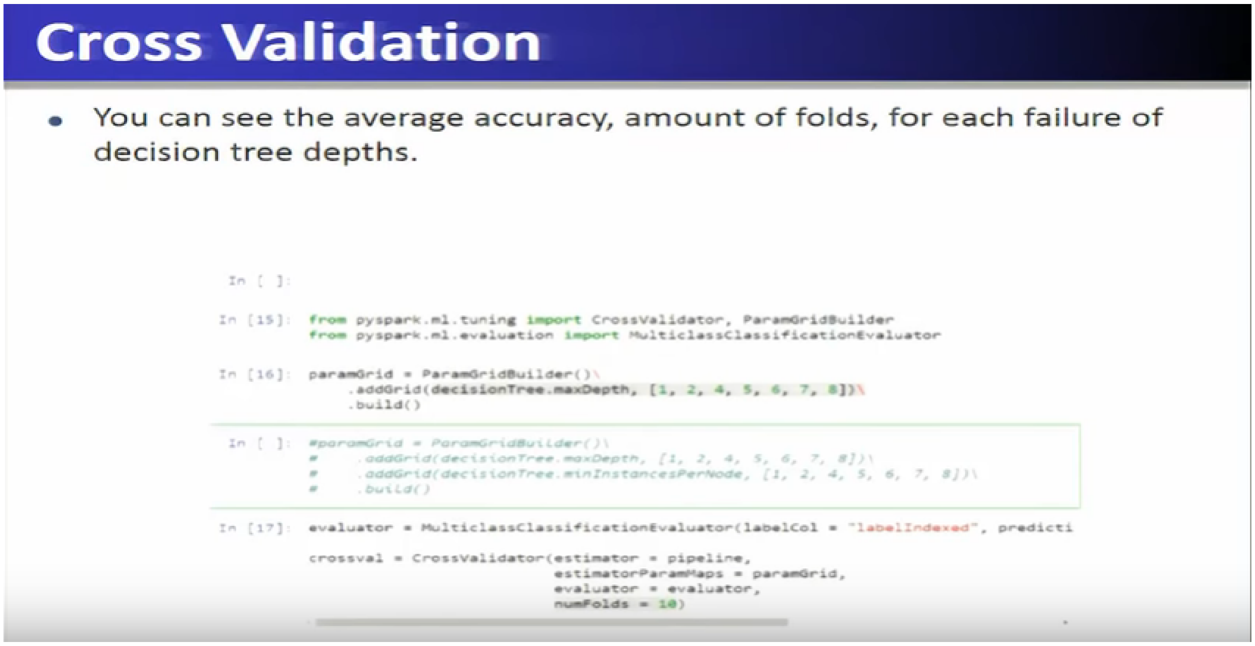
Cross Validation

- We create cvModel and it takes some time because Spark needs to make training and evaluating the quality 10 times.

```
crossval = CrossValidator(estimator = pipeline,
                          estimatorParamMaps = paramGrid,
                          evaluator = evaluator,
                          numFolds = 10)

In [18]: cvModel = crossval.fit(inputDF2)

In [19]: cvModel.avgMetrics

Out[19]: [0.8924563921120648,
          0.9203744192767783,
          0.9418223975919139,
          0.9419915318207598,
          0.9457320946210345,
          0.9380267917034486,
          0.9361037147804091]

In [ ]: print cvModel.bestModel.stages[0]

In [ ]: cvModel.transform(....)
```

we create the CV model and takes some time because, spark need to make the training and evaluation, the qualified the ten times.

Refer Slide Time :(39:21)



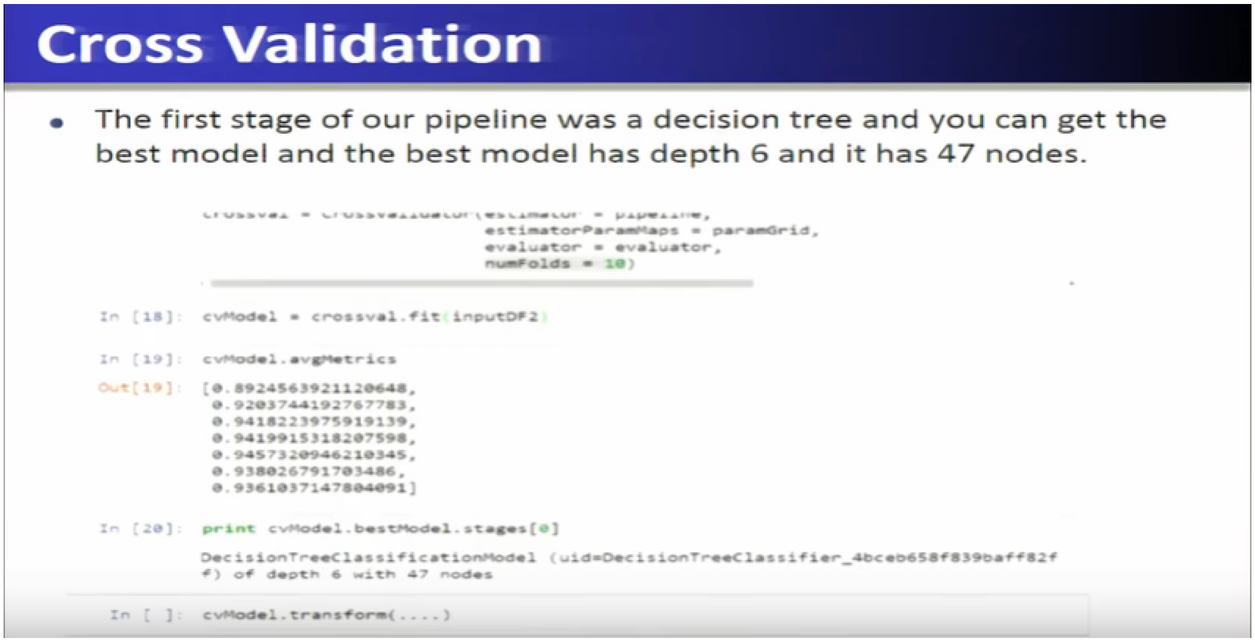
Cross Validation

- You can see the average accuracy, amount of folds, for each failure of decision tree depths.

```
In [ ]:

In [15]: from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
         from pyspark.ml.evaluation import MulticlassClassificationEvaluator

In [16]: paramGrid = ParamGridBuilder()\
             .addGrid(decisionTree.maxDepth, [1, 2, 4, 5, 6, 7, 8])\
             .build()

In [ ]: #paramGrid = ParamGridBuilder()\
         #     .addGrid(decisionTree.maxDepth, [1, 2, 4, 5, 6, 7, 8])\
         #     .addGrid(decisionTree.minInstancesPerNode, [1, 2, 4, 5, 6, 7, 8])\
         #     .build()

In [17]: evaluator = MulticlassClassificationEvaluator(labelCol = "labelIndexed", predicti

         crossval = CrossValidator(estimator = pipeline,
                                   estimatorParamMaps = paramGrid,
                                   evaluator = evaluator,
                                   numFolds = 10)
```

Now, we can see that the average accuracy, amount of fold, for each of the failure of the three depths.

Refer Slide Time :(39:30)



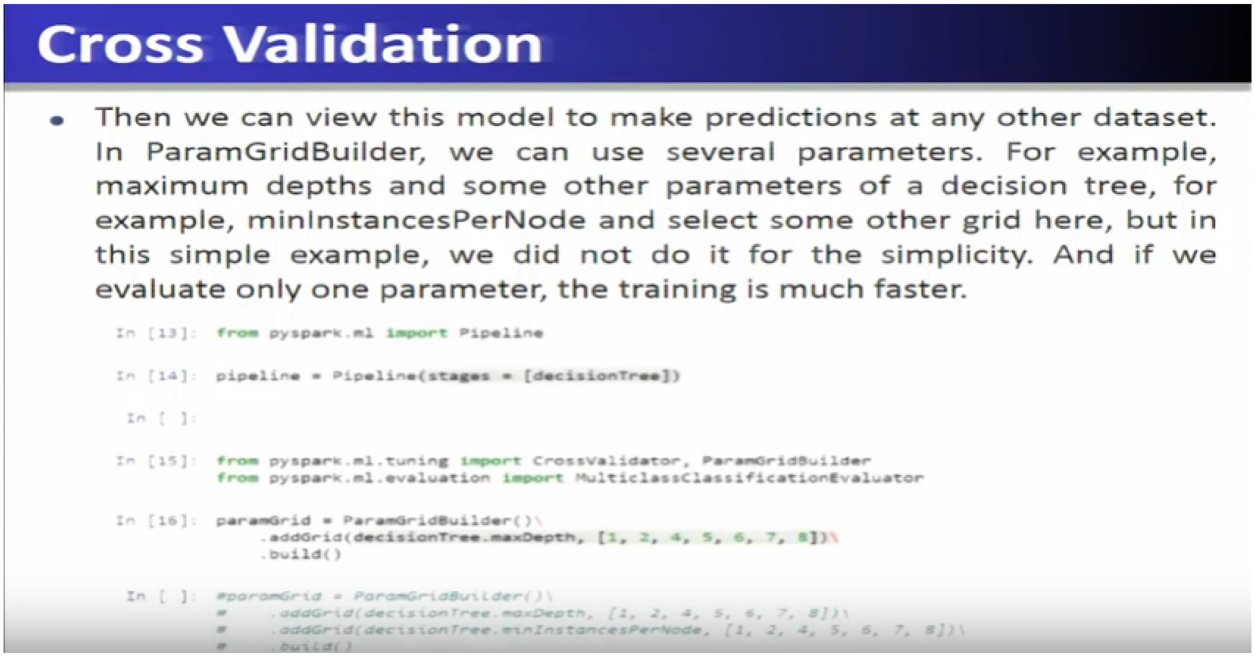
Cross Validation

- The first stage of our pipeline was a decision tree and you can get the best model and the best model has depth 6 and it has 47 nodes.

```
        crossval = CrossValidator(estimator = pipeline,
                                  estimatorParamMaps = paramGrid,
                                  evaluator = evaluator,
                                  numFolds = 10)

In [18]: cvModel = crossval.fit(inputDF2)

In [19]: cvModel.avgMetrics

Out[19]: [0.8924563921120648,
          0.9203744192767783,
          0.9418223975919139,
          0.9419915318207598,
          0.9457320946210345,
          0.938026791703486,
          0.9361037147804091]

In [20]: print cvModel.bestModel.stages[0]

         DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4bceb658f839baff82f
         f) of depth 6 with 47 nodes

 In [ ]: cvModel.transform(....)
```

Now, we can see and the first stage of our pipeline was the decision tree, so you can get the best model and the best model has the depth six and it has 47 nodes, in this case.

Refer Slide Time :(39:42)



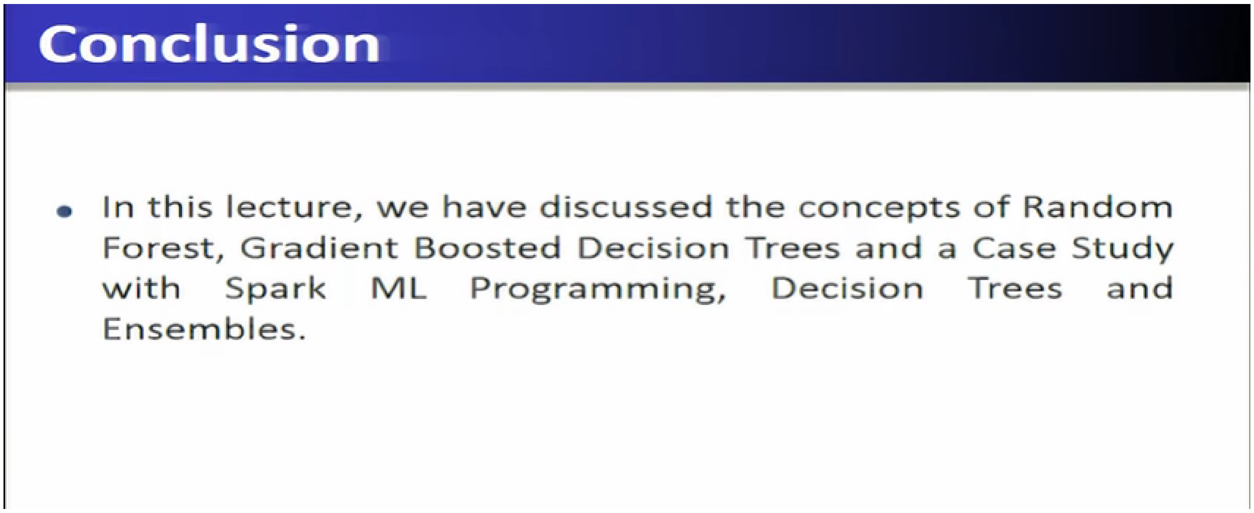
Cross Validation

- Then we can view this model to make predictions at any other dataset. In ParamGridBuilder, we can use several parameters. For example, maximum depths and some other parameters of a decision tree, for example, minInstancesPerNode and select some other grid here, but in this simple example, we did not do it for the simplicity. And if we evaluate only one parameter, the training is much faster.

```
In [13]: from pyspark.ml import Pipeline

In [14]: pipeline = Pipeline(stages = [decisionTree])

 In [ ]:

In [15]: from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
         from pyspark.ml.evaluation import MulticlassClassificationEvaluator

In [16]: paramGrid = ParamGridBuilder()\
              .addGrid(decisionTree.maxDepth, [1, 2, 4, 5, 6, 7, 8])\
              .build()

 In [ ]: #paramGrid = ParamGridBuilder()\
         #    .addGrid(decisionTree.maxDepth, [1, 2, 4, 5, 6, 7, 8])\
         #    .addGrid(decisionTree.minInstancesPerNode, [1, 2, 4, 5, 6, 7, 8])\
         #    .build()
```

Then we can view this model, to make the prediction at any other data sets. So, in parameter great builder, we can use several parameters. For example, maximum depth and some other parameter of the decision tree, for example, minimum instances per node and select some other grid here. But, in this example, we did not do it, due to the simplicity and if we evaluate only one parameter, the training is much faster.

Refer Slide Time :(40:09)



Conclusion

- In this lecture, we have discussed the concepts of Random Forest, Gradient Boosted Decision Trees and a Case Study with Spark ML Programming, Decision Trees and Ensembles.

So, conclusion in this lecture, we have discussed the concept of random forests, gradient boosted decision trees and we have also covered a, case study using a spark ml programming, on decision trees and, and other learning tree ensembles. Thank you.