In CART, the value of minbucket can affect the model's out-of-sample accuracy.

As we discussed earlier in the lecture, if minbucket is too small, over-fitting might occur.

But if minbucket is too large, the model might be too simple.

So how should we set this parameter value?

We could select the value that gives the best testing set accuracy, but this isn't right.

The idea of the testing set is to measure model performance on data the model has never seen before.

By picking the value of minbucket to get the best test set performance, the testing set was implicitly used to generate the model.

Instead, we'll use a method called K-fold Cross Validation, which is one way to properly select the parameter value.

This method works by going through the following steps.

First, we split the training set into k equally sized subsets, or folds.

In this example, k equals 5.

Then we select k - 1, or four folds, to estimate the model, and compute predictions on the remaining one fold, which is often referred to as the validation set.

We build a model and make predictions for each possible parameter value we're considering.

Then we repeat this for each of the other folds, or pieces of our training set.

So we would build a model using folds 1, 2, 3, and 5 to make predictions on fold 4, and then we would build a model using folds 1, 2, 4, and 5 to make predictions on fold 3, etc.

So ultimately, cross validation builds many models, one for each fold and possible parameter value.

Then, for each candidate parameter value, and for each fold, we can compute the accuracy of the model.

This plot shows the possible parameter values on the x-axis, and the accuracy of the model on the y-axis.

This line shows the accuracy of our model on fold 1.

We can also compute the accuracy of the model using each of the other folds as the validation sets.

We then average the accuracy over the k folds to determine the final parameter value that we want to use.

Typically, the behavior looks like this-- if the parameter value is too small, then the accuracy is lower, because the model is probably over-fit to the training set.

But if the parameter value is too large, then the accuracy is also lower, because the model is too simple.

In this case, we would pick a parameter value around six, because it leads to the maximum average accuracy over all parameter values.

So far, we've used the parameter minbucket to limit our tree in R. When we use cross validation in R, we'll use a parameter called cp instead.

This is the complexity parameter.

It's like Adjusted R-squared for linear regression, and AIC for logistic regression, in that it measures the trade-off between model complexity and accuracy on the training set.

A smaller cp value leads to a bigger tree, so a smaller cp value might over-fit the model to the training set.

But a cp value that's too large might build a model that's too simple.

Let's go to R, and use cross validation to select the value of cp for our CART tree.

In our R console, let's try cross validation for our CART model.

To do this, we need to install and load two new packages.

First, we'll install the package "caret".

You should see some lines run in your R console, and then when you're back to the blinking cursor, load the package with library(caret).

Now, let's install the package "e1071".

So again, install.packages("e1071").

Again, you should see some lines run in your R console, and when you're back to the cursor, load the package with library(e1071).

Now, we'll define our cross validation experiment.

First, we need to define how many folds we want.

We can do this using the trainControl function.

So we'll say numFolds = trainControl, and then in parentheses, method = "cv", for cross validation, and then number = 10, for 10 folds.

Then we need to pick the possible values for our cp parameter, using the expand.grid function.

So we'll call it cpGrid, and then use expand.grid, where the only argument is .cp = seq(0.01,0.5,0.01).

This will define our cp parameters to test as numbers from 0.01 to 0.5, in increments of 0.01.

Now, we're ready to perform cross validation.

We'll do this using the train function, where the first argument is similar to that when we're building models.

It's the dependent variable, Reverse, followed by a tilde symbol, and then the independent variables separated by plus signs-- Circuit + Issue + Petitioner + Respondent + LowerCourt + Unconst.

Our data set here is Train, with a capital T, and then we need to add the arguments method = "rpart", since we want to cross validate a CART model, and then trControl = numFolds, the output of our trainControl function, and then tuneGrid = cpGrid, the output of the expand.grid function.

If you hit Enter, it might take a little while, but after a few seconds, you should get a table describing the cross validation accuracy for different cp parameters.

The first column gives the cp parameter that was tested, and the second column gives the cross validation accuracy for that cp value.

The accuracy starts lower, and then increases, and then will start decreasing again, as we saw in the slides.

At the bottom of the output, it says, "Accuracy was used to select the optimal model using the largest value.

The final value used for the model was cp = 0.18." This is the cp value we want to use in our CART model.

So now let's create a new CART model with this value of cp, instead of the minbucket parameter.

We'll call this model StevensTreeCV, and we'll use the rpart function, like we did earlier, to predict Reverse using

all of our independent variables: Circuit, Issue, Petitioner, Respondent, LowerCourt, and Unconst.

Our data set here is Train, and then we want method = "class", since we're building a classification tree, and cp = 0.18.

Now, let's make predictions on our test set using this model.

We'll call our predictions PredictCV, and we'll use the predict function to make predictions using the model StevensTreeCV, the newdata set Test, and we want to add type = "class", so that we get class predictions.

Now let's create our confusion matrix, using the table function, where we first give the true outcome, Test$Reverse, and then our predictions, PredictCV.

So the accuracy of this model is 59 + 64, divided by the total number in this table, 59 + 18 + 29 + 64, the total number of observations in our test set.

So the accuracy of this model is 0.724.

Remember that the accuracy of our previous CART model was 0.659.

Cross validation helps us make sure we're selecting a good parameter value, and often this will significantly increase the accuracy.

If we had already happened to select a good parameter value, then the accuracy might not of increased that much.

But by using cross validation, we can be sure that we're selecting a smart parameter value.