In this video, we'll see how to build a CART model in R. Let's start by reading in the data file "stevens.csv".

We'll call our data frame stevens and use the read.csv function to read in the data file "stevens.csv".

Remember to navigate to the directory on your computer containing the file "stevens.csv" first.

Now, let's take a look at our data using the str function.

We have 566 observations, or Supreme Court cases, and nine different variables.

Docket is just a unique identifier for each case, and Term is the year of the case.

Then we have our six independent variables: the circuit court of origin, the issue area of the case, the type of petitioner, the type of respondent, the lower court direction, and whether or not the petitioner argued that a law or practice was unconstitutional.

The last variable is our dependent variable, whether or not Justice Stevens voted to reverse the case: 1 for reverse, and 0 for affirm.

Now before building models, we need to split our data into a training set and a testing set.

We'll do this using the sample.split function, like we did last week for logistic regression.

First, we need to load the package caTools with library(caTools).

Now, so that we all get the same split, we need to set the seed.

Remember that this can be any number, as long as we all use the same number.

Let's set the seed to 3000.

Now, let's create our split.

We'll call it spl, and we'll use the sample.split function, where the first argument needs to be our outcome variable, stevens$Reverse, and then the second argument is the SplitRatio, or the percentage of data that we want to put in the training set.

In this case, we'll put 70% of the data in the training set.

Now, let's create our training and testing sets using the subset function.

We'll call our training set Train, and we'll take a subset of stevens, only taking the observations for which spl is equal to TRUE.

We'll call our testing set Test, and here take a subset of stevens, but this time, taking the observations for which spl is equal to FALSE.

Now, we're ready to build our CART model.

First we need to install and load the rpart package and the rpart plotting package.

Remember that to install a new package, we use the install.packages function, and then in parentheses and quotes, give the name of the package we want to install.

In this case, rpart.

After you hit Enter, a CRAN mirror should pop up asking you to pick a location near you.

Go ahead and pick the appropriate location.

In my case, I'll pick Pennsylvania in the United States, and hit OK.

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

Now, let's install the package rpart.plot.

Again, some lines should run in your R Console, and when you're back to the blinking cursor, load the package with library(rpart.plot).

Now we can create our CART model using the rpart function.

We'll call our model StevensTree, and we'll use the rpart function, where the first argument is the same as if we were building a linear or logistic regression model.

We give our dependent variable-- in our case, Reverse-- followed by a tilde sign, and then the independent variables separated by plus signs.

So Circuit + Issue + Petitioner + Respondent + LowerCourt + Unconst.

We also need to give our data set that should be used to build our model, which in our case is Train.

Now we'll give two additional arguments here.

The first one is method = "class".

This tells rpart to build a classification tree, instead of a regression tree.

You'll see how we can create regression trees in recitation.

The last argument we'll give is minbucket = 25.

This limits the tree so that it doesn't overfit to our training set.

We selected a value of 25, but we could pick a smaller or larger value.

We'll see another way to limit the tree later in this lecture.

Now let's plot our tree using the prp function, where the only argument is the name of our model, StevensTree.

You should see the tree pop up in the graphics window.

The first split of our tree is whether or not the lower court decision is liberal.

If it is, then we move to the left in the tree.

And we check the respondent.

If the respondent is a criminal defendant, injured person, politician, state, or the United States, we predict 0, or affirm.

You can see here that the prp function abbreviates the values of the independent variables.

If you're not sure what the abbreviations are, you could create a table of the variable to see all of the possible values.

prp will select the abbreviation so that they're uniquely identifiable.

So if you made a table, you could see that CRI stands for criminal defendant, INJ stands for injured person, etc.

So now moving on in our tree, if the respondent is not one of these types, we move on to the next split, and we check the petitioner.

If the petitioner is a city, employee, employer, government official, or politician, then we predict 0, or affirm.

If not, then we check the circuit court of origin.

If it's the 10th, 1st, 3rd, 4th, DC or Federal Court, then we predict 0.

Otherwise, we predict 1, or reverse.

We can repeat this same process on the other side of the tree if the lower court decision is not liberal.

Comparing this to a logistic regression model, we can see that it's very interpretable.

A CART tree is a series of decision rules which can easily be explained.

Now let's see how well our CART model does at making predictions for the test set.

So back in our R Console, we'll call our predictions PredictCART, and we'll use the predict function, where the first argument is the name of our model, StevensTree.

The second argument is the new data we want to make predictions for, Test.

And we'll add a third argument here, which is type = "class".

We need to give this argument when making predictions for our CART model if we want the majority class predictions.

This is like using a threshold of 0.5.

We'll see in a few minutes how we can leave this argument out and still get probabilities from our CART model.

Now let's compute the accuracy of our model by building a confusion matrix.

So we'll use the table function, and first give the true outcome values-- Test$Reverse, and then our predictions, PredictCART.

To compute the accuracy, we need to add up the observations we got correct, 41 plus 71, divided by the total number of observations in the table, or the total number of observations in our test set.

So the accuracy of our CART model is 0.659.

If you were to build a logistic regression model, you would get an accuracy of 0.665 and a baseline model that always predicts Reverse, the most common outcome, has an accuracy of 0.547.

So our CART model significantly beats the baseline and is competitive with logistic regression.

It's also much more interpretable than a logistic regression model would be.

Lastly, to evaluate our model, let's generate an ROC curve for our CART model using the ROCR package.

First, we need to load the package with the library function, and then we need to generate our predictions again, this time without the type = "class" argument.

We'll call them PredictROC, and we'll use the predict function, giving just as the two arguments StevensTree and newdata = Test.

Let's take a look at what this looks like by just typing PredictROC and hitting Enter.

For each observation in the test set, it gives two numbers which can be thought of as the probability of outcome 0 and the probability of outcome 1.

More concretely, each test set observation is classified into a subset, or bucket, of our CART tree.

These numbers give the percentage of training set data in that subset with outcome 0 and the percentage of data in the training set in that subset with outcome 1.

We'll use the second column as our probabilities to generate an ROC curve.

So just like we did last week for logistic regression, we'll start by using the prediction function.

We'll call the output pred, and then use prediction, where the first argument is the second column of PredictROC, which we can access with square brackets, and the second argument is the true outcome values, Test$Reverse.

Now we need to use the performance function, where the first argument is the outcome of the prediction function, and then the next two arguments are true positive rate and false positive rate, what we want on the x and y-axes of our ROC curve.

Now we can just plot our ROC curve by typing plot(perf).

If you switch back to your graphics window, you should see the ROC curve for our model.

In the next quick question, we'll ask you to compute the test set AUC of this model.