In this video, we'll introduce a method that is similar to CART called Random Forests.

This method was designed to improve the prediction accuracy of CART and works by building a large number of CART trees.

Unfortunately, this makes the method less interpretable than CART, so often you need to decide if you value the interpretability or the increase in accuracy more.

To make a prediction for a new observation, each tree in the forest votes on the outcome and we pick the outcome that receives the majority of the votes.

So how does Random Forests build many CART trees?

We can't just run CART multiple times because it would create the same tree every time.

To prevent this, Random Forests only allows each tree to split on a random subset of the available independent variables.

And each tree is built from what we call a bagged or bootstrapped sample of the data.

This just means that the data used as the training data for each tree is selected randomly with replacement.

Let's look at an example.

Suppose we have five data points in our training set.

We'll call them 1, 2, 3, 4, and 5.

For the first tree, we'll randomly pick five data points randomly sampled with replacement.

So the data could be 2, 4, 5, 2, and 1.

Each time we pick one of the five data points regardless of whether or not it's been selected already.

These would be the five data points we use when constructing the first CART tree.

Then we repeat this process for the second tree.

This time the data set might be 3, 5, 1, 5, and 2.

And we would use this data when building the second CART tree.

Then we would repeat this process for each additional tree we want to create.

So since each tree sees a different set of variables and a different set of data, we get what's called a forest of many different trees.

Just like CART, Random Forests has some parameter values that need to be selected.

The first is the minimum number of observations in a subset, or the minbucket parameter from CART.

When we create a random forest in R, this will be called nodesize.

A smaller value of nodesize, which leads to bigger trees, may take longer in R. Random Forests is much more computationally intensive than CART.

The second parameter is the number of trees to build, which is called intree in R. This should not be set too small, but the larger it is the longer it will take.

A couple hundred trees is typically plenty.

A nice thing about Random Forests is that it's not as sensitive to the parameter values as CART is.

In the next video, we'll talk about a nice way to pick the CART parameter.

For Random Forests, as long as this selection is a reasonable it's OK.

Let's switch to R and create a Random Forest model to predict the decisions of Justice Stevens.

In our R console, let's start by installing and loading the package "randomForest." We first need to install the package using the install.packages function for the package "randomForest." You should see a few lines run in your R console and then when you're back to the blinking cursor, load the package with the library command.

Now we're ready to build our Random Forests model.

We'll call it StevensForest and use the randomForest function, first giving our dependent variable, Reverse, followed by a tilde sign, and then our independent variable separated by plus signs.

Circuit.

Issue.

Petitioner.

Respondent.

LowerCourt.

And Unconst.

We'll use the data set Train.

For Random Forests we need to give two additional arguments.

These are nodesize, also known as minbucket for CART, and we'll set this equal to 25, the same value we used for our CART model.

And then we need to set the parameter ntree.

This is the number of trees to build.

And we'll build 200 trees here.

Then hit Enter.

You should see an interesting warning message here.

In CART, we added the argument method="class", so that it was clear that we're doing a classification problem.

As I mentioned earlier, trees can also be used for regression problems, which you'll see in the recitation.

The Random Forest function does not have a method argument.

So when we want to do a classification problem, we need to make sure outcome is a factor.

Let's convert the variable Reverse to a factor variable in both our training and our testing sets.

We do this by typing the name of the variable we want to convert-- in our case Train$Reverse-- and then type as.factor and then in parentheses the variable name, Train$Reverse.

And just repeat this for the test set as well.

Test$Reverse=as.factor(Test$Reverse) Now let's try creating our Random Forest again.

Just use the up arrow to get back to the Random Forest line and hit Enter.

We didn't get a warning message this time so our model is ready to make predictions.

Let's compute predictions on our test set.

We'll call our predictions PredictForest and use the predict function to make predictions using our model, StevensForest, and the new data set Test.

Let's look at the confusion matrix to compute our accuracy.

We'll use the table function and first give the true outcome, Test$Reverse, and then our predictions, PredictForest.

Our accuracy here is (40+74)/(40+37+19+74).

So the accuracy of our Random Forest model is about 67%.

Recall that our logistic regression model had an accuracy of 66.5% and our CART model had an accuracy of 65.9%.

So our Random Forest model improved our accuracy a little bit over CART.

Sometimes you'll see a smaller improvement in accuracy and sometimes you'll see that Random Forests can significantly improve in accuracy over CART.

We'll see this a lot in the recitation in the homework assignments.

Keep in mind that Random Forests has a random component.

You may have gotten a different confusion matrix than me because there's a random component to this method.