Unit 4 Lecture 2: Pruning and cross-validating decision trees (updated)

November 1, 2022

Today, we will learn how to select the complexity of decision trees based on cost complexity pruning and cross-validation, as implemented in the `rpart` package.

First, let's load some libraries:

```
library(rpart)       # for training decision trees
library(rpart.plot)  # for plotting decision trees
library(stat471)     # for creating CV plots for decision trees
library(tidyverse)   # for everything else
```

Regression trees

Like last time, we will be using the `hitters` data, splitting into training and testing:

```
hitters_data <- read_csv("hitters-data.csv")
set.seed(1) # set seed for reproducibility
train_samples <- sample(1:nrow(hitters_data), round(0.8 * nrow(hitters_data)))
hitters_train <- hitters_data %>% filter(row_number() %in% train_samples)
hitters_test <- hitters_data %>% filter(!(row_number() %in% train_samples))
```

As before, we fit a regression tree by calling `rpart`:

```
tree_fit <- rpart(Salary ~ ., data = hitters_train)
```

Tree pruning and cross validation

It turns out that in addition to growing the tree, behind the scenes `rpart` has already:

- used cost complexity pruning to get the nested sequence of trees
- applied 10-fold cross-validation to compute the CV estimates and standard errors for each value of α

All we need to do is call the `printcp` function to get a summary of all this information:

```
printcp(tree_fit)
```

```
##
## Regression tree:
## rpart(formula = Salary ~ ., data = hitters_train)
##
## Variables actually used in tree construction:
## [1] AtBat   CAtBat  CHits   CRBI    Errors  PutOuts Walks
##
## Root node error: 160.25/210 = 0.76309
##
## n= 210
##
```

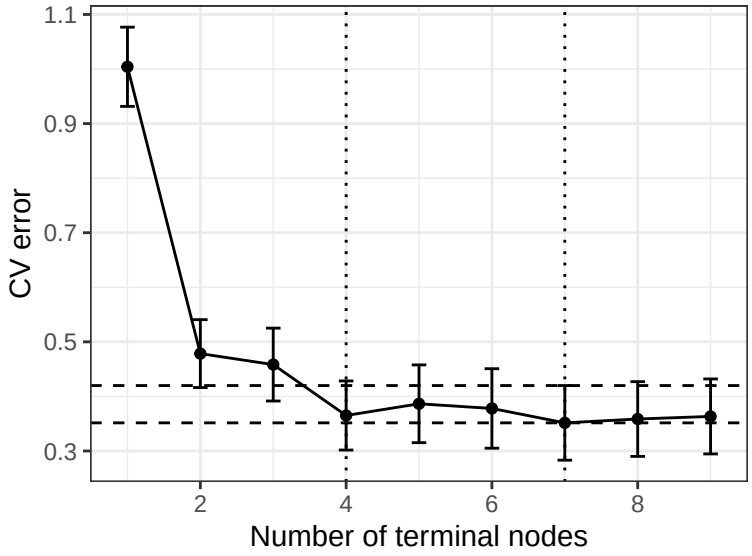
```
##         CP nsplit rel error  xerror     xstd
## 1 0.567669      0   1.00000 1.00411 0.072613
## 2 0.063293      1   0.43233 0.47843 0.062225
## 3 0.060590      2   0.36904 0.45832 0.066787
## 4 0.033764      3   0.30845 0.36500 0.063361
## 5 0.029146      4   0.27468 0.38646 0.071271
## 6 0.015175      5   0.24554 0.37791 0.072805
## 7 0.011737      6   0.23036 0.35152 0.068380
## 8 0.010248      7   0.21863 0.35856 0.068482
## 9 0.010000      8   0.20838 0.36327 0.068681
```

Let's focus on the table at the bottom of this output. Each row corresponds to a tree in the sequence obtained by pruning. Let's discuss each column in turn:

- The `CP` column is the "complexity parameter". It is related to, but not exactly the same as, the α parameter from the slides. Be careful! The terminology "complexity parameter" is a bit misleading because higher complexity parameters correspond to less complex models (just like lambda in penalized regression).
- `nsplit` is the number of splits in the tree. Note that `1+nsplit` is the number of terminal nodes in the tree.
- `rel error` is the RSS training error of the tree, normalized by the total variance of the response; equivalently, this is $1 - R^2$. The training error decreases as the complexity increases.
- `xerror` is the cross-validation error estimate.
- `xstd` is the cross-validation standard error.

The exact values of the complexity parameter are not so important; we might as well parameterize the trees based on the number of terminal nodes. Armed with all this information, we can produce a CV plot using `cv_tree()` from `stat471`:

```
tree_cv_info <- cv_tree(tree_fit)
tree_cv_info$cv_plot
```



NOTE: The two vertical lines indicate the numbers of terminal nodes corresponding to the minimum of the CV curve and the one-standard-error-rule. The bottom horizontal line indicates the minimum value of the CV curve, and the top horizontal line indicates one standard error above this minimum. Among the trees whose CV error estimate is at most one standard error above the minimum (i.e. below the top horizontal line), the one-standard-error rule chooses the tree with the smallest number of terminal nodes. Note that this is *slightly* different from the one-standard-error rule presented in Unit 2; this is the correct version and

should be used in conjunction with penalized regression, splines, etc.

The `tree_cv_info` object also has additional information, like `nleaves.1se` and `nleaves.min`. There are also fields called `CP.1se` and `CP.min`, containing the corresponding CP values. These will be useful below.

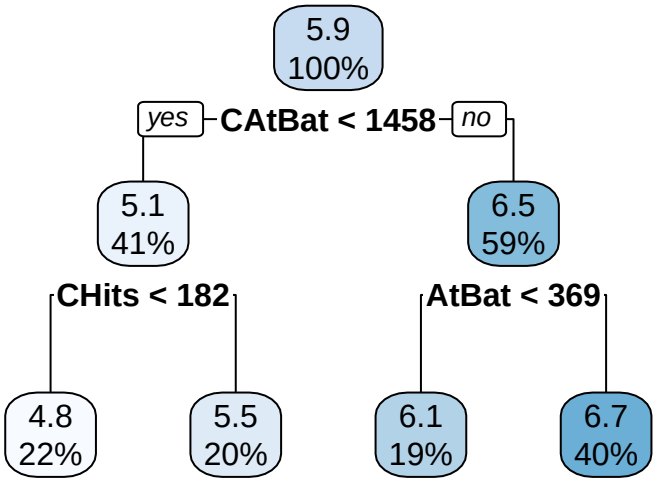
Extracting the pruned tree and making predictions

To actually get the optimal pruned tree, we need to use the function `prune`, specifying the complexity parameter CP. We can get this complexity parameter from the `tree_cv_info` object:

```
optimal_tree <- prune(tree_fit, cp = tree_cv_info$CP.1se)
```

As before, we can plot this tree using `rpart.plot`:

```
rpart.plot(optimal_tree)
```



That is a small tree! In the bias variance trade-off, sometimes less (complexity) is more (predictive performance).

Now we can make predictions on the test data and evaluate MSE using this tree:

```
pred <- predict(optimal_tree, newdata = hitters_test)
pred
```

```
##        1        2        3        4        5        6        7        8
## 6.660241 4.810335 4.810335 4.810335 6.056463 4.810335 6.660241 6.660241
##        9       10       11       12       13       14       15       16
## 6.056463 6.660241 6.660241 5.494350 6.660241 6.660241 5.494350 6.660241
##       17       18       19       20       21       22       23       24
## 4.810335 6.660241 6.056463 6.056463 6.660241 5.494350 6.660241 6.056463
##       25       26       27       28       29       30       31       32
## 6.056463 6.660241 4.810335 6.660241 6.660241 5.494350 5.494350 6.660241
##       33       34       35       36       37       38       39       40
## 5.494350 4.810335 6.056463 6.056463 6.660241 6.660241 6.056463 6.660241
##       41       42       43       44       45       46       47       48
## 6.056463 6.660241 6.660241 4.810335 6.660241 6.660241 4.810335 6.660241
##       49       50       51       52       53
## 6.056463 5.494350 4.810335 6.660241 6.660241
```

```
mean((pred - hitters_test$Salary)^2)
```

```
## [1] 0.3088943
```

Exercise: Classification trees

Let's continue with the heart disease data from last time:

```
heart_data <- read_csv("heart-data.csv")
set.seed(1) # set seed for reproducibility
train_samples <- sample(1:nrow(heart_data), round(0.8 * nrow(heart_data)))
heart_train <- heart_data %>% filter(row_number() %in% train_samples)
heart_test <- heart_data %>% filter(!(row_number() %in% train_samples))

# fit a classification tree
tree_fit <- rpart(AHD ~ .,
  method = "class",               # classification
  parms = list(split = "gini"),   # Gini index for splitting
  data = heart_train
)
```

Tree pruning and cross-validation

1. Produce the table of the trees in the sequence obtained from cost complexity pruning. How does `nsplit` vary with `CP`? Does this relationship make sense?

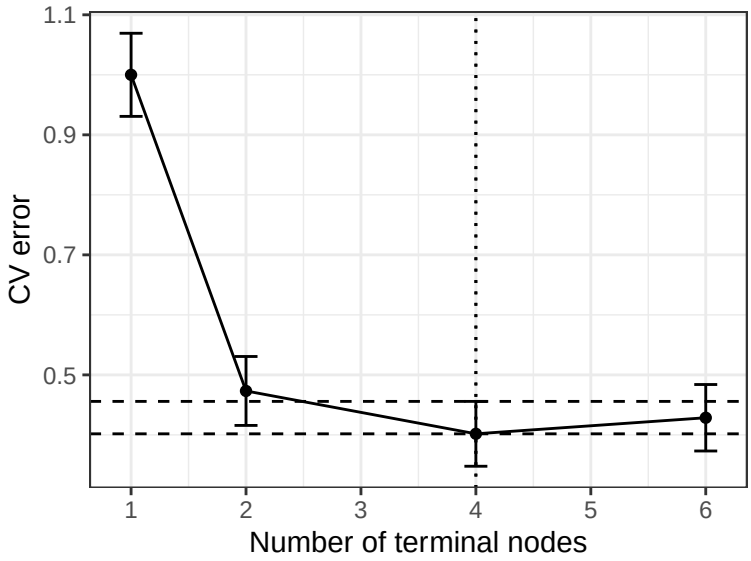
   ```
   printcp(tree_fit)
   ```

   ```
   ##
   ## Classification tree:
   ## rpart(formula = AHD ~ ., data = heart_train, method = "class",
   ##     parms = list(split = "gini"))
   ##
   ## Variables actually used in tree construction:
   ## [1] Ca        ChestPain Slope     Thal
   ##
   ## Root node error: 112/242 = 0.46281
   ##
   ## n= 242
   ##
   ##         CP nsplit rel error  xerror     xstd
   ## 1 0.526786      0   1.00000 1.00000 0.069256
   ## 2 0.053571      1   0.47321 0.47321 0.057444
   ## 3 0.035714      3   0.36607 0.40179 0.054040
   ## 4 0.010000      5   0.29464 0.42857 0.055385
   ```

   We see that, as `CP` decreases, `nsplit` increases. This makes sense because decreasing CP penalizes large trees less, and therefore these trees have more splits.

2. Produce the CV plot. How many terminal nodes would we choose based on the one-standard-error rule? Do we notice anything strange about the CV plot?

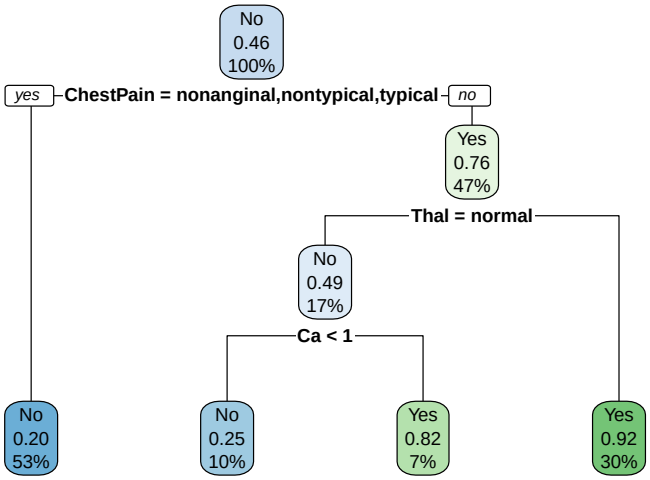
   ```
   tree_cv_info <- cv_tree(tree_fit)
   tree_cv_info$cv_plot
   ```



We would choose four terminal nodes based on the one-standard-error rule. What's strange about the CV plot is that some of the numbers of terminal nodes appear to have been skipped. This is because the cost complexity pruning algorithm sometimes prunes splits that are not at leaf nodes.

3. Extract and visualize the tree chosen by cross-validation. How many terminal nodes does the tree have?

```
optimal_tree <- prune(tree_fit, cp = tree_cv_info$CP.1se)
rpart.plot(optimal_tree)
```



We see that this tree has four terminal nodes, which matches with what the CV plot shows.

4. What is the test misclassification error of this decision rule?

```
# make predictions on test data
predictions <- predict(tree_fit, newdata = heart_test, type = "class")

# compute misclassification error
mean(predictions != heart_test$AHD)
```

```
## [1] 0.1967213
```

We get a test misclassification error of about 20%.