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

November 4, 2021

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)            # install.packages("rpart")
library(rpart.plot)       # install.packages("rpart.plot")
library(tidyverse)
```

Regression trees

Like last time, we will be using the `Hitters` data from the `ISLR` package, splitting into training and testing:

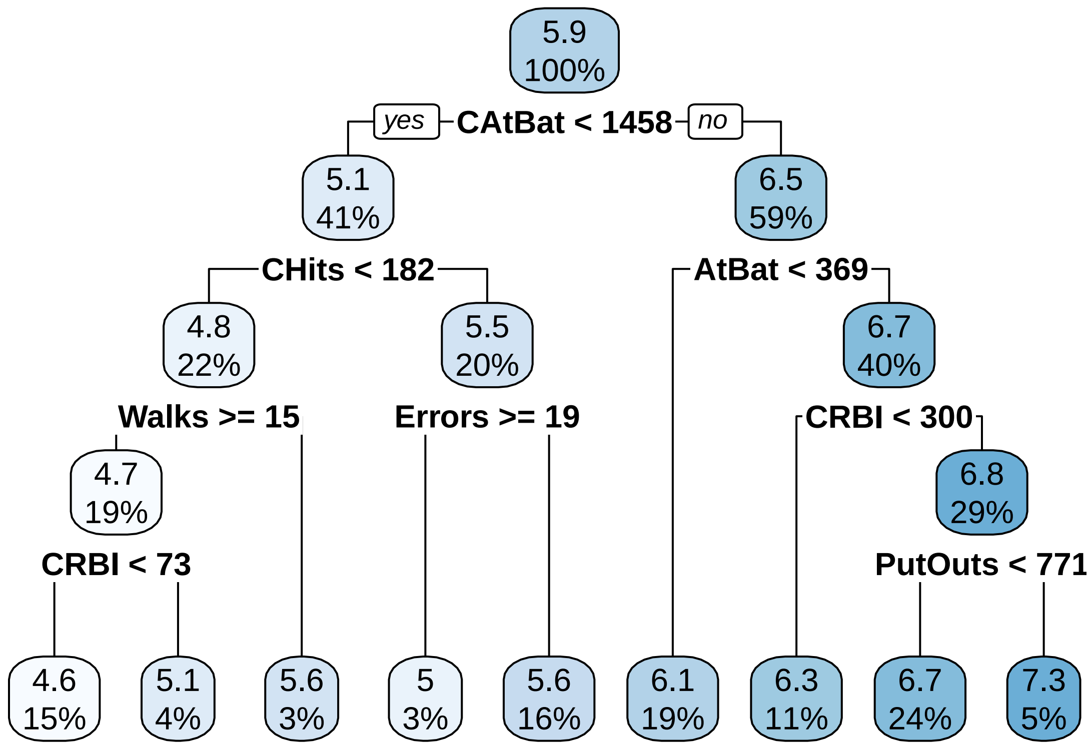
```
Hitters = ISLR2::Hitters %>%
  as_tibble() %>%
  filter(!is.na(Salary)) %>%   # remove NA values (in general not necessary)
  mutate(Salary = log(Salary)) # log-transform the salary
Hitters
```

```
## # A tibble: 263 x 20
##    AtBat  Hits HmRun  Runs   RBI Walks Years CAtBat CHits CHmRun CRuns  CRBI
##    <int> <int> <int> <int> <int> <int> <int>  <int> <int>  <int> <int> <int>
##  1   315    81     7    24    38    39    14   3449   835     69   321   414
##  2   479   130    18    66    72    76     3   1624   457     63   224   266
##  3   496   141    20    65    78    37    11   5628  1575    225   828   838
##  4   321    87    10    39    42    30     2    396   101     12    48    46
##  5   594   169     4    74    51    35    11   4408  1133     19   501   336
##  6   185    37     1    23     8    21     2    214    42      1    30     9
##  7   298    73     0    24    24     7     3    509   108      0    41    37
##  8   323    81     6    26    32     8     2    341    86      6    32    34
##  9   401    92    17    49    66    65    13   5206  1332    253   784   890
## 10   574   159    21   107    75    59    10   4631  1300     90   702   504
## # ... with 253 more rows, and 8 more variables: CWalks <int>, League <fct>,
## #   Division <fct>, PutOuts <int>, Assists <int>, Errors <int>, Salary <dbl>,
## #   NewLeague <fct>
```

```
set.seed(1) # set seed for reproducibility
train_samples = sample(1:nrow(Hitters), round(0.8*nrow(Hitters)))
Hitters_train = Hitters %>% filter(row_number() %in% train_samples)
Hitters_test = Hitters %>% filter(!(row_number() %in% train_samples))
```

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

```
tree_fit = rpart(Salary ~ ., data = Hitters_train)
rpart.plot(tree_fit)
```



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:

- `CP` is the "complexity parameter" α. Be careful! This terminology 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. The built-in function to produce the CV plot is not as nice as the one built into `cv.glmnet`, so we'll make our own using `ggplot`:

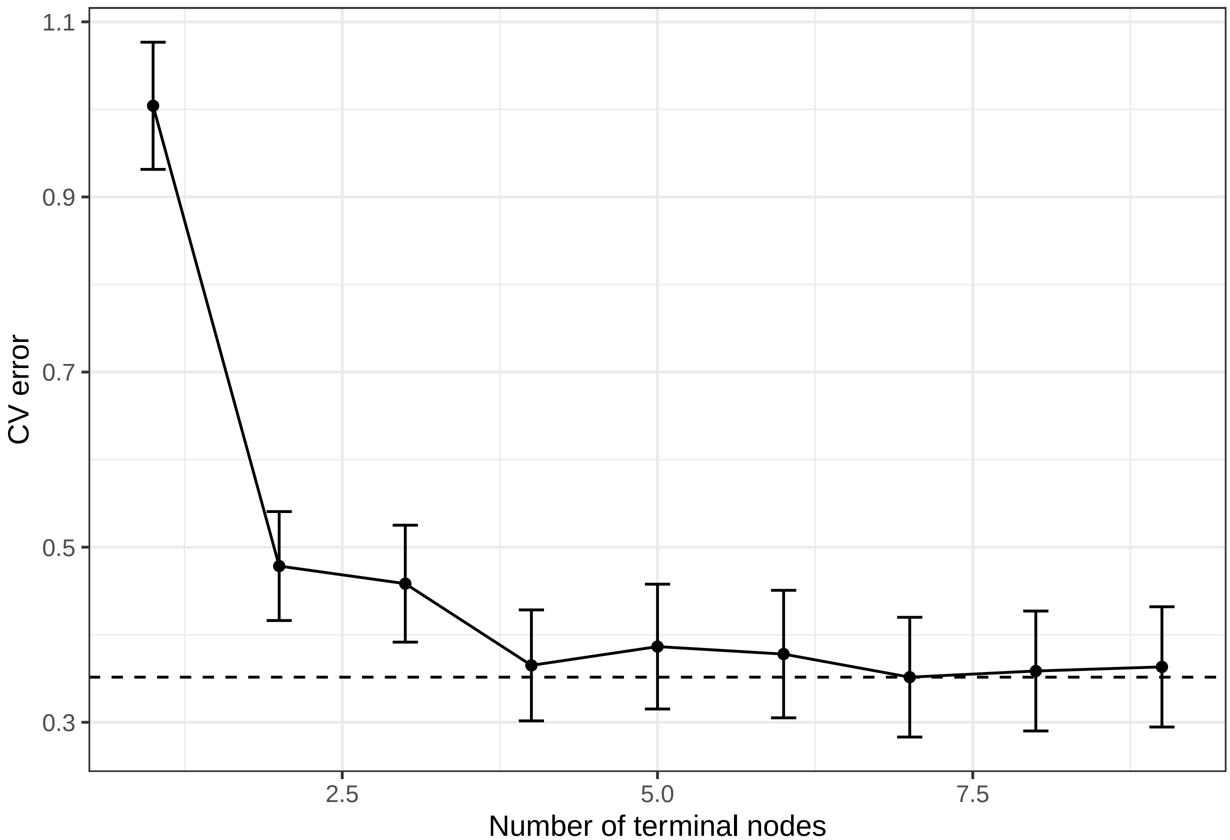
```
cp_table = printcp(tree_fit) %>% as_tibble()
```

```
##
## 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
```

```
cp_table
```

```
## # A tibble: 9 x 5
##       CP nsplit `rel error` xerror   xstd
##    <dbl>  <dbl>       <dbl>  <dbl>  <dbl>
## 1 0.568       0       1      1.00  0.0726
## 2 0.0633      1       0.432  0.478 0.0622
## 3 0.0606      2       0.369  0.458 0.0668
## 4 0.0338      3       0.308  0.365 0.0634
## 5 0.0291      4       0.275  0.386 0.0713
## 6 0.0152      5       0.246  0.378 0.0728
## 7 0.0117      6       0.230  0.352 0.0684
## 8 0.0102      7       0.219  0.359 0.0685
## 9 0.01        8       0.208  0.363 0.0687
```

```
cp_table %>%
  ggplot(aes(x = nsplit+1, y = xerror,
             ymin = xerror - xstd, ymax = xerror + xstd)) +
  geom_point() + geom_line() +
  geom_errorbar(width = 0.2) +
  xlab("Number of terminal nodes") + ylab("CV error") +
  geom_hline(aes(yintercept = min(xerror)), linetype = "dashed") +
  theme_bw()
```



Audience participation: How many terminal nodes would we choose based on the one-standard-error rule?

Unfortunately, we don't have a convenient lambda.1se field of the output to directly extract the optimal complexity parameter based on the one standard error rule. Nevertheless, we can find it pretty simply using `dplyr`:

```
optimal_tree_info = cp_table %>%
  filter(xerror - xstd < min(xerror)) %>%
  arrange(nsplit) %>%
  head(1)
optimal_tree_info
```

```
## # A tibble: 1 x 5
##       CP nsplit `rel error` xerror   xstd
##    <dbl>  <dbl>       <dbl>  <dbl>  <dbl>
## 1 0.0338      3       0.308  0.365 0.0634
```

Audience participation: What is the above code is doing? Why is `nsplit` two rather than three as suggested by the plot above?

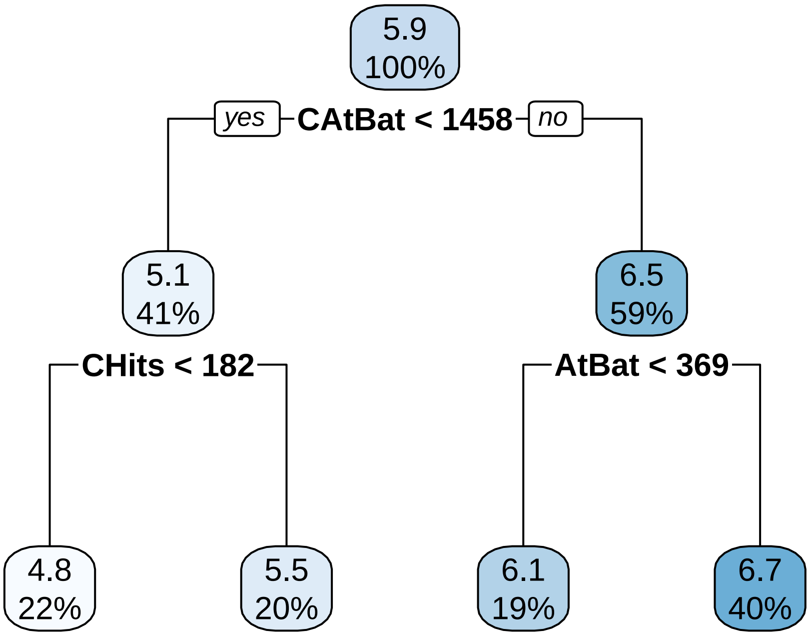
Extracting the pruned tree and making predictions

To actually get the optimal pruned tree, we need to use the function `prune`, specifying the complexity parameter

```
optimal_tree = prune(tree = tree_fit, cp = optimal_tree_info$CP)
```

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:

```
# download the data
url = "https://raw.githubusercontent.com/JWarmenhoven/ISLR-python/master/Notebooks/Data/Heart.csv"
Heart = read_csv(url, col_types = "-iffiiiiiddiiff")

# split into train/test
set.seed(1) # set seed for reproducibility
train_samples = sample(1:nrow(Heart), round(0.8*nrow(Heart)))
Heart_train = Heart %>% filter(row_number() %in% train_samples)
Heart_test = Heart %>% 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. What exactly is the interpretation of the `CP` column in this case? Do the values make sense?
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?
3. Extract and visualize the tree chosen by cross-validation. In words, how would you summarize the resulting decision rule?
4. What is the test misclassification error of this decision rule?