Classification and Regression Trees

Nate Wells

Math 243: Stat Learning

November 8th, 2021

Outline

In today's class, we will. . .

- Investigate pruning algorithms for improving accuracy of trees
- Create and prune decision trees in R

Section 1

Pruning

The general tree algorithm

1. Begin with the entire data set S and search every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squred error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

The general tree algorithm

1. Begin with the entire data set S and search every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squred error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeat step one on both S_1 and S_2.

The general tree algorithm

1. Begin with the entire data set S and search every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squred error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeat step one on both S_1 and S_2.
3. Repeat on the new regions.

The general tree algorithm

1. Begin with the entire data set S and search every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squred error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeat step one on both S_1 and S_2.
3. Repeat on the new regions.
4. . . .

The general tree algorithm

1. Begin with the entire data set S and search every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squred error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeat step one on both S_1 and S_2.
3. Repeat on the new regions.
4. ...
5. Stop?

The general tree algorithm

1. Begin with the entire data set S and search every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squred error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeat step one on both S_1 and S_2.
3. Repeat on the new regions.
4. ...
5. Stop?

How do we decide when to abort the algorithm?

The general tree algorithm

1. Begin with the entire data set S and search every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squred error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeat step one on both S_1 and S_2.
3. Repeat on the new regions.
4. ...
5. Stop?

How do we decide when to abort the algorithm?

Consider the RSS of a **big** tree. How might training and test RSS compare?

Subtrees

A **subtree** is a regression tree obtained by removing some of the branches and nodes from the full regression tree.

Subtrees

A **subtree** is a regression tree obtained by removing some of the branches and nodes from the full regression tree.

- Compare test and training RSS between full tree and a subtree.

Subtrees

A **subtree** is a regression tree obtained by removing some of the branches and nodes from the full regression tree.

- Compare test and training RSS between full tree and a subtree.

Like the best subset selection algorithm for linear models, we can improve training RSS by exhaustively searching all subtrees for the best performing model.

Subtrees

A **subtree** is a regression tree obtained by removing some of the branches and nodes from the full regression tree.

- Compare test and training RSS between full tree and a subtree.

Like the best subset selection algorithm for linear models, we can improve training RSS by exhaustively searching all subtrees for the best performing model.

- But this search is actually even more computationally expensive than best subset!

Subtrees

A **subtree** is a regression tree obtained by removing some of the branches and nodes from the full regression tree.

- Compare test and training RSS between full tree and a subtree.

Like the best subset selection algorithm for linear models, we can improve training RSS by exhaustively searching all subtrees for the best performing model.

- But this search is actually even more computationally expensive than best subset!
- So we instead restrict our attention to those subtrees most likely to improve RSS

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.
- We consider a sequence of trees indexed by a tuning parameter α.

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.
- We consider a sequence of trees indexed by a tuning parameter α.

For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.
- We consider a sequence of trees indexed by a tuning parameter α.

For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

- That is, α penalizes a tree based on its number of terminal nodes.

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.
- We consider a sequence of trees indexed by a tuning parameter α.

For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

- That is, α penalizes a tree based on its number of terminal nodes.
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.
- We consider a sequence of trees indexed by a tuning parameter α.

For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

- That is, α penalizes a tree based on its number of terminal nodes.
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.
- We can find the optimal value of α using cross-validation

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.
- We consider a sequence of trees indexed by a tuning parameter α.

For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

- That is, α penalizes a tree based on its number of terminal nodes.
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.
- We can find the optimal value of α using cross-validation

There are two ways to select the **best** subtree.

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.
- We consider a sequence of trees indexed by a tuning parameter α.

For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

- That is, α penalizes a tree based on its number of terminal nodes.
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.
- We can find the optimal value of α using cross-validation

There are two ways to select the **best** subtree.

1. Choose the tree with smallest MSE.

Pruning Algorithm

Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

- The goal is to find a tree of optimal size with the smallest error rate.
- We consider a sequence of trees indexed by a tuning parameter α.

For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

- That is, α penalizes a tree based on its number of terminal nodes.
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.
- We can find the optimal value of α using cross-validation

There are two ways to select the **best** subtree.

1. Choose the tree with smallest MSE.
2. Choose the *smallest* tree with MSE within 1 standard deviation of smallest MSE

Trees on Trees

We use a subset of the `pdxTrees` dataset from the `pdxTrees` repo (maintained by K. McConville, I. Caldwell, and N. Horton)

- To keep things manageable, we'll focus on trees in 3 parks nearby Reed.

```
library(pdxTrees)
my_pdxTrees <- get_pdxTrees_parks(park = c("Powel Park", "Woodstock Park", "Berkeley Park"))
```

Trees on Trees

We use a subset of the pdxTrees dataset from the pdxTrees repo (maintained by K. McConville, I. Caldwell, and N. Horton)

- To keep things manageable, we'll focus on trees in 3 parks nearby Reed.

```
library(pdxTrees)
my_pdxTrees <- get_pdxTrees_parks(park = c("Powel Park", "Woodstock Park", "Berkeley Park"))
```

- And use trees from another park as a test set:

```
my_pdxTrees_test <- get_pdxTrees_parks(park = c("Glenwood Park"))
```

Trees on Trees

We use a subset of the `pdxTrees` dataset from the `pdxTrees` repo (maintained by K. McConville, I. Caldwell, and N. Horton)

- To keep things manageable, we'll focus on trees in 3 parks nearby Reed.

```
library(pdxTrees)
my_pdxTrees <- get_pdxTrees_parks(park = c("Powel Park", "Woodstock Park", "Berkeley Park"))
```

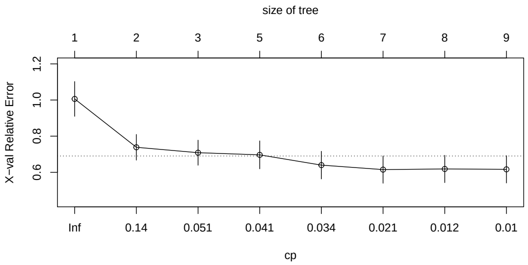
- And use trees from another park as a test set:

```
my_pdxTrees_test <- get_pdxTrees_parks(park = c("Glenwood Park"))
```

- Can we predict carbon sequestration based on `Tree_Height` and `Crown_Width_EW`?

Pruning Example

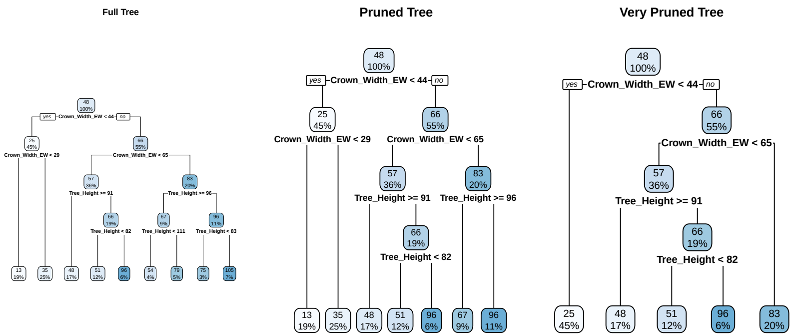
How does MSE vary as tree size changes?



- What are the test MSEs for the full tree and the subtrees with 5 and 7 terminal nodes?

```
## # A tibble: 3 x 4
##   model        .metric .estimator .estimate
##   <chr>        <chr>   <chr>          <dbl>
## 1 full         rmse    standard        20.3
## 2 pruned       rmse    standard        19.7
## 3 very pruned rmse    standard        20.1
```

Comparison


Full Tree
48
100%
yes
Crown_Width_EW < 44
no
25
45%
Crown_Width_EW < 29
66
55%
Crown_Width_EW < 65
57
36%
Tree_Height >= 91
83
20%
Tree_Height >= 96
66
19%
Tree_Height < 82
67
9%
Tree_Height < 111
96
11%
Tree_Height < 83
13
19%
35
25%
48
17%
51
12%
96
6%
54
4%
79
5%
75
3%
105
7%
Pruned Tree
48
100%
yes
Crown_Width_EW < 44
no
25
45%
Crown_Width_EW < 29
66
55%
Crown_Width_EW < 65
57
36%
Tree_Height >= 91
83
20%
Tree_Height >= 96
66
19%
Tree_Height < 82
13
19%
35
25%
48
17%
51
12%
96
6%
67
9%
96
11%
Very Pruned Tree
48
100%
yes
Crown_Width_EW < 44
no
66
55%
Crown_Width_EW < 65
57
36%
Tree_Height >= 91
66
19%
Tree_Height < 82
25
45%
48
17%
51
12%
96
6%
83
20%

Section 2

Trees in R

Creating Tree Models in R

There are two common packages for creating regression trees in R: `tree` and `rpart`.

Creating Tree Models in R

There are two common packages for creating regression trees in R: `tree` and `rpart`.

- The `tree` package is one of the oldest packages on CRAN. It is a (tiny) bit easier to use. But allows far less customization. ISLR uses `tree`. (Traditional)

Creating Tree Models in R

There are two common packages for creating regression trees in R: `tree` and `rpart`.

- The `tree` package is one of the oldest packages on CRAN. It is a (tiny) bit easier to use. But allows far less customization. ISLR uses `tree`. (Traditional)
- The `rpart` package is newer, computationally faster, and has more options. It also can be combined with other packages for **much** nicer plots. Applied Predictive Modeling uses `rpart`. (Recommended)

Trees using ‘rpart“

- To fit a tree using variables `Tree_Height`, `Crown_Width_EW`, `Crown_Width_NS`, `Crown_Base_Height`:

```
set.seed(1)
library(rpart)
tree_model1 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width_EW + Crown_Width_NS + Crown_Base_Height,
                     data = my_pdxTrees)
```

Trees using ‘rpart“

- To fit a tree using variables `Tree_Height`, `Crown_Width_EW`, `Crown_Width_NS`, `Crown_Base_Height`:

```
set.seed(1)
library(rpart)
tree_model1 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width_EW + Crown_Width_NS + Crown_Base_Height,
                     data = my_pdxTrees)
```

- We can change several features of the tree by adding a `control` argument:

```
set.seed(1)
tree_model2 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width_EW + Crown_Width_NS + Crown_Base_Height,
                     control = rpart.control(minsplit = 30, xval = 10, maxdepth = 8),
                     data = my_pdxTrees)
```

Trees using ‘rpart“

- To fit a tree using variables `Tree_Height`, `Crown_Width_EW`, `Crown_Width_NS`, `Crown_Base_Height`:

```
set.seed(1)
library(rpart)
tree_model1 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width_EW + Crown_Width_NS + Crown_Base_Height,
                     data = my_pdxTrees)
```

- We can change several features of the tree by adding a `control` argument:

```
set.seed(1)
tree_model2 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width_EW + Crown_Width_NS + Crown_Base_Height,
                     control = rpart.control(minsplit = 30, xval = 10, maxdepth = 8),
                     data = my_pdxTrees)
```

- `minsplit` is the minimum number of observations in a node
- `xval` is the number of cross-validation folds used
- `maxdepth` is the maximum depth of any node in the final tree

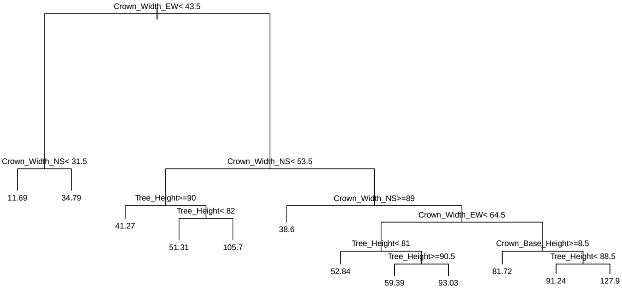
Plots using `plot`

- There are several options for visualizing trees with varying ease-of-use and aesthetics.
  - The base R `plot` function quickly generates plots, but...

Plots using plot

- There are several options for visualizing trees with varying ease-of-use and aesthetics.
  - The base R plot function quickly generates plots, but. . .

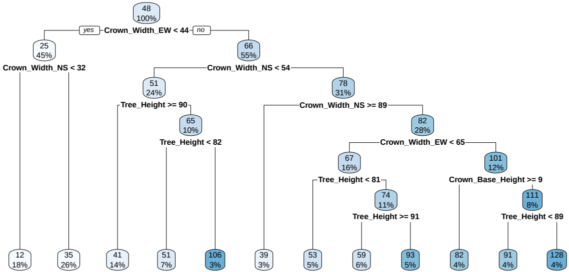
```
plot(tree_model1)
text(tree_model1, pretty = 0, cex = .5)
```



Plots using `rpart.plot`

- An alternative to `plot` is the `rpart.plot` function from the package of the same name:

```
library(rpart.plot)
rpart.plot(tree_model1)
```



- Some further customization available (see `?rpart.plot`)

Trees in R via `rpart` cont'd

- The `rpart` function automatically performs k-fold CV when choosing among potential splits.

Trees in R via `rpart` cont'd

- The `rpart` function automatically performs k-fold CV when choosing among potential splits.
- To access results, append `$cptable` to the `rpart` model object:

```
tree_model1$cptable
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.31073097      0 1.0000000 1.0105895 0.09666964
## 2 0.07370105      1 0.6892690 0.7679112 0.07560215
## 3 0.04577064      2 0.6155680 0.7211540 0.07009241
## 4 0.04342290      4 0.5240267 0.6668256 0.06922100
## 5 0.03450324      5 0.4806038 0.6378779 0.06854061
## 6 0.01877027      7 0.4115973 0.6624756 0.08409966
## 7 0.01778685      9 0.3740568 0.7124886 0.09350971
## 8 0.01000000     11 0.3384831 0.7070176 0.09248091
```

Trees in R via rpart cont'd

- The `rpart` function automatically performs *k*-fold CV when choosing among potential splits.
- To access results, append `$cptable` to the `rpart` model object:

```
tree_model1$cptable
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.31073097      0 1.0000000 1.0105895 0.09666964
## 2 0.07370105      1 0.6892690 0.7679112 0.07560215
## 3 0.04577064      2 0.6155680 0.7211540 0.07009241
## 4 0.04342290      4 0.5240267 0.6668256 0.06922100
## 5 0.03450324      5 0.4806038 0.6378779 0.06854061
## 6 0.01877027      7 0.4115973 0.6624756 0.08409966
## 7 0.01778685      9 0.3740568 0.7124886 0.09350971
## 8 0.01000000     11 0.3384831 0.7070176 0.09248091
```

- `CP` is the value of the complexity parameter
- `nsplit` is number of splits
- `rel error` is $1 - R^2$, using $R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$
- `xerror` is cross-validated estimate of relative error
- `xstd` is the standard deviation in `xerror` based on CV

Analyze Results

- The printcp function displays key model information

```
printcp(tree_model1)
```

```
##
## Regression tree:
## rpart(formula = Carbon_Sequestration_lb ~ Tree_Height + Crown_Width_EW +
##     Crown_Width_NS + Crown_Base_Height, data = my_pdxTrees)
##
## Variables actually used in tree construction:
## [1] Crown_Base_Height Crown_Width_EW    Crown_Width_NS    Tree_Height
##
## Root node error: 406713/307 = 1324.8
##
## n= 307
##
##         CP nsplit rel error  xerror     xstd
## 1 0.310731      0   1.00000 1.01059 0.096670
## 2 0.073701      1   0.68927 0.76791 0.075602
## 3 0.045771      2   0.61557 0.72115 0.070092
## 4 0.043423      4   0.52403 0.66683 0.069221
## 5 0.034503      5   0.48060 0.63788 0.068541
## 6 0.018770      7   0.41160 0.66248 0.084100
## 7 0.017787      9   0.37406 0.71249 0.093510
## 8 0.010000     11   0.33848 0.70702 0.092481
```

Analyze Results cont'd

- Detailed listing of model parts can be accessed via `summary`:

Analyze Results cont'd

- Detailed listing of model parts can be accessed via `summary`:

```
summary(tree_model1)
```

```
## Call:
## rpart(formula = Carbon_Sequestration_lb ~ Tree_Height + Crown_Width_EW +
##     Crown_Width_NS + Crown_Base_Height, data = my_pdxTrees)
##   n= 307
##
##           CP nsplit rel error    xerror       xstd
## 1 0.31073097      0 1.0000000 1.0105895 0.09666964
## 2 0.07370105      1 0.6892690 0.7679112 0.07560215
## 3 0.04577064      2 0.6155680 0.7211540 0.07009241
## 4 0.04342290      4 0.5240267 0.6668256 0.06922100
## 5 0.03450324      5 0.4806038 0.6378779 0.06854061
## 6 0.01877027      7 0.4115973 0.6624756 0.08409966
## 7 0.01778685      9 0.3740568 0.7124886 0.09350971
## 8 0.01000000     11 0.3384831 0.7070176 0.09248091
##
## Variable importance
##    Crown_Width_EW    Crown_Width_NS       Tree_Height Crown_Base_Height
##                38                28                24                10
##
## Node number 1: 307 observations,    complexity param=0.310731
##   mean=47.95081, MSE=1324.797
##   left son=2 (137 obs) right son=3 (170 obs)
##   Primary splits:
##       Crown_Width_EW    < 43.5 to the left,  improve=0.31073100, (0 missing)
##       Crown_Width_NS    < 49.5 to the left,  improve=0.28692940, (0 missing)
##       Tree_Height       < 78.5 to the left,  improve=0.16233240, (0 missing)
##       Crown_Base_Height < 4.5  to the left,  improve=0.05039755, (0 missing)
##   Surrogate splits:
##       Crown_Width_NS    < 43.5 to the left,  agree=0.788, adj=0.526, (0 split)
##       Tree_Height       < 45.5 to the left,  agree=0.739, adj=0.416, (0 split)
##       Crown_Base_Height < 5.5  to the left,  agree=0.629, adj=0.168, (0 split)
```

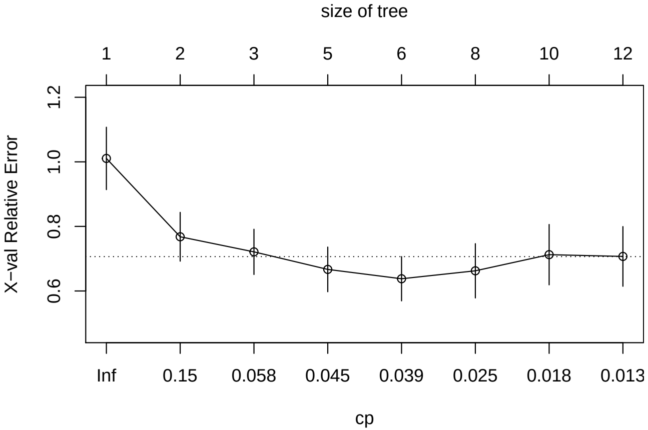
CV Plots

- We can plot the results of cross-validation using `plotcp`:

CV Plots

- We can plot the results of cross-validation using `plotcp`:

```
plotcp(tree_model1)
```



- The horizontal line is 1 SE above minimum relative error

Pruning

- Based on the CV plot, 6 splits with $CP = 0.039$ gives the lowest error
  - While 5 splits with $CP = 0.045$ gives least splits within 1 SE of best.

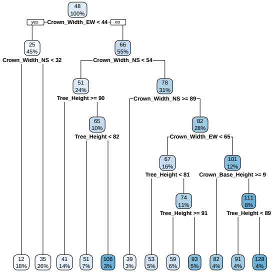
Pruning

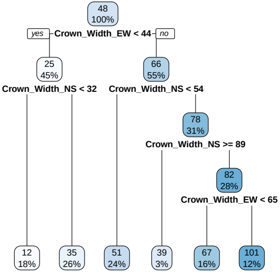
- Based on the CV plot, 6 splits with $CP = 0.039$ gives the lowest error
  - While 5 splits with $CP = 0.045$ gives least splits within 1 SE of best.
- We can prune our tree using the `prune` function with a given value of cp

Pruning

- Based on the CV plot, 6 splits with $CP = 0.039$ gives the lowest error
  - While 5 splits with $CP = 0.045$ gives least splits within 1 SE of best.
- We can prune our tree using the `prune` function with a given value of `cp`

```r
pruned_tree <- prune(tree_model1, cp = 0.039)
```





Test Error Rates

- How well do models do on the test data?

Test Error Rates

- How well do models do on the test data?
  - Let's build a results data frame:

```
results <- data.frame(model = "full",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(tree_model1, my_pdxTrees_test))
results <- rbind(results,
                 data.frame(model = "pruned",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(pruned_tree, my_pdxTrees_test)))
```

Test Error Rates

- How well do models do on the test data?
  - Let's build a results data frame:

```
results <- data.frame(model = "full",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(tree_model1, my_pdxTrees_test))
results <- rbind(results,
                 data.frame(model = "pruned",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(pruned_tree, my_pdxTrees_test)))
```

- And use `rmse` from `yardstick` to assess:

Test Error Rates

- How well do models do on the test data?
    - Let's build a results data frame:

```
results <- data.frame(model = "full",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(tree_model1, my_pdxTrees_test))
results <- rbind(results,
                 data.frame(model = "pruned",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(pruned_tree, my_pdxTrees_test)))
```

- And use `rmse` from `yardstick` to assess:

```
library(yardstick)
results %>% group_by(model) %>% rmse(truth = obs, estimate = preds)
```

```
## # A tibble: 2 x 4
##   model  .metric .estimator .estimate
##   <chr>  <chr>   <chr>          <dbl>
## 1 full   rmse    standard        21.1
## 2 pruned rmse    standard        19.2
```