Classification and Regression Trees

Nate Wells

Math 243: Stat Learning

November 8th, 2021

Outline

In today's class, we will...

- Discuss decision trees as a non-parametric model
- Investigate pruning algorithms for improving accuracy of trees

Section 1

Decision Trees

Guess my favorite film or novel

At the start of the term, you were tasked with asking each each other a series of yes-or-no questions in order to deduce each person's favorite book or movie.

Guess my favorite film or novel

At the start of the term, you were tasked with asking each each other a series of yes-or-no questions in order to deduce each person's favorite book or movie.

- Each of your guessing algorithms forms a (partial) decision tree.
  - Yes/No questions represent a branching point or node
  - The final guess represents the prediction

Guess my favorite film or novel

At the start of the term, you were tasked with asking each each other a series of yes-or-no questions in order to deduce each person's favorite book or movie.

- Each of your guessing algorithms forms a (partial) decision tree.
  - Yes/No questions represent a branching point or node
  - The final guess represents the prediction
- What makes an effective question?

Guess my favorite film or novel

At the start of the term, you were tasked with asking each each other a series of yes-or-no questions in order to deduce each person's favorite book or movie.

- Each of your guessing algorithms forms a (partial) decision tree.
  - Yes/No questions represent a branching point or node
  - The final guess represents the prediction
- What makes an effective question?
  - Separates data into roughly equal sizes
  - Data in each group relatively are similar
  - Later questions should be based on answers to earlier questions.
  - Early questions are general, later questions are specific.

My Favorite Book

Previously asked questions:

1. Is it set in the UK? **No**
2. Is it about a sick day? **No**
3. Does it take place on an island? **No**
4. Is it set in California? **No**
5. Are there any gunshots in the book? **No**
6. Is the book less than 200 pages? **No**

My Favorite Book

Previously asked questions:

1. Is it set in the UK? **No**
2. Is it about a sick day? **No**
3. Does it take place on an island? **No**
4. Is it set in California? **No**
5. Are there any gunshots in the book? **No**
6. Is the book less than 200 pages? **No**

As class, decide on up to 6 more questions to ask about the book. Then submit your guess on a slip of paper.

My Favorite Book

Previously asked questions:

1. Is it set in the UK? **No**
2. Is it about a sick day? **No**
3. Does it take place on an island? **No**
4. Is it set in California? **No**
5. Are there any gunshots in the book? **No**
6. Is the book less than 200 pages? **No**

As class, decide on up to 6 more questions to ask about the book. Then submit your guess on a slip of paper.

Section 2

Regression Trees

Regression Trees

Basic regression trees partition data into smaller groups that are homogeneous with respect to predictors.

Regression Trees

Basic regression trees partition data into smaller groups that are homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

Regression Trees

Basic regression trees partition data into smaller groups that are homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method.

Regression Trees

Basic regression trees partition data into smaller groups that are homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method.

1. The method begins with the entire data set S and searches every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squared error:

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

Regression Trees

Basic regression trees partition data into smaller groups that are homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method.

1. The method begins with the entire data set S and searches every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squared 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. The method then repeats step 1 for each of the two groups S_1 and S_2.

Regression Trees

Basic regression trees partition data into smaller groups that are homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method.

1. The method begins with the entire data set S and searches every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squared 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. The method then repeats step 1 for each of the two groups S_1 and S_2.
3. The method continues splitting groups until each subdivision has few observation (or another predetermined stopping condition is met)

Trees on Trees

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

Trees on Trees

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

- The data was collected by the Portland Parks and Rec's *Urban Forestry Tree Inventory Project*.
- The Tree Inventory Project has gathered data on Portland trees since 2010, collecting this data in the summer months with a team of over 1,300 volunteers and city employees.

pdxTrees Data

- The `pdxTrees` dataset is too large to install alongside the package. Instead, the package provides helper loading functions:
  - `get_pdxTrees_parks()` pulls data on 25,534 trees from 174 Portland parks
  - `get_pdxTrees_streets()` pulls data on 218,602 trees along Portland streets

pdxTrees Data

- The pdxTrees dataset is too large to install alongside the package. Instead, the package provides helper loading functions:
  - get_pdxTrees_parks() pulls data on 25,534 trees from 174 Portland parks
  - get_pdxTrees_streets() pulls data on 218,602 trees along Portland streets
- To keep things manageable, we'll focus on trees in parks nearby Reed.

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

pdxTrees Data

- The pdxTrees dataset is too large to install alongside the package. Instead, the package provides helper loading functions:
  - `get_pdxTrees_parks()` pulls data on 25,534 trees from 174 Portland parks
  - `get_pdxTrees_streets()` pulls data on 218,602 trees along Portland streets
- To keep things manageable, we'll focus on trees in parks nearby Reed.

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

- How many observations?

```
dim(my_pdxTrees)
```

```
## [1] 1039   34
```

pdxTrees Data

- The pdxTrees dataset is too large to install alongside the package. Instead, the package provides helper loading functions:
  - get_pdxTrees_parks() pulls data on 25,534 trees from 174 Portland parks
  - get_pdxTrees_streets() pulls data on 218,602 trees along Portland streets
- To keep things manageable, we'll focus on trees in parks nearby Reed.

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

- How many observations?

```
dim(my_pdxTrees)
```

```
## [1] 1039   34
```

- What variables are present?

```
names(my_pdxTrees)
```

pdxTrees Data

- The pdxTrees dataset is too large to install alongside the package. Instead, the package provides helper loading functions:
    - `get_pdxTrees_parks()` pulls data on 25,534 trees from 174 Portland parks
    - `get_pdxTrees_streets()` pulls data on 218,602 trees along Portland streets
- To keep things manageable, we'll focus on trees in parks nearby Reed.

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

- How many observations?

```
dim(my_pdxTrees)
```

```
## [1] 1039   34
```

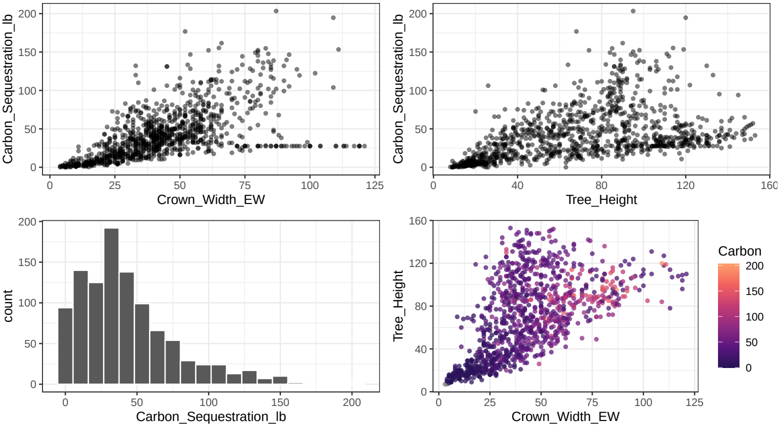
- What variables are present?

```
names(my_pdxTrees)
```

```
##  [1] "Longitude"                 "Latitude"
##  [3] "UserID"                    "Genus"
##  [5] "Family"                    "DBH"
##  [7] "Inventory_Date"            "Species"
##  [9] "Common_Name"               "Condition"
## [11] "Tree_Height"               "Crown_Width_NS"
## [13] "Crown_Width_EW"            "Crown_Base_Height"
## [15] "Collected_By"              "Park"
## [17] "Scientific_Name"           "Functional_Type"
## [19] "Mature_Size"               "Native"
## [21] "Edible"                    "Nuisance"
## [23] "Structural_Value"          "Carbon_Storage_lb"
## [25] "Carbon_Storage_value"      "Carbon_Sequestration_lb"
## [27] "Carbon_Sequestration_value" "Stormwater_ft"
## [29] "Stormwater_value"          "Pollution_Removal_value"
## [31] "Pollution_Removal_oz"      "Total_Annual_Services"
## [33] "Origin"                    "Species_Factoid"
```

Carbon Sequestration

- Can we predict carbon sequestration based on other tree features?



An Old Friend

This seems like a good time to implement linear regression:

An Old Friend

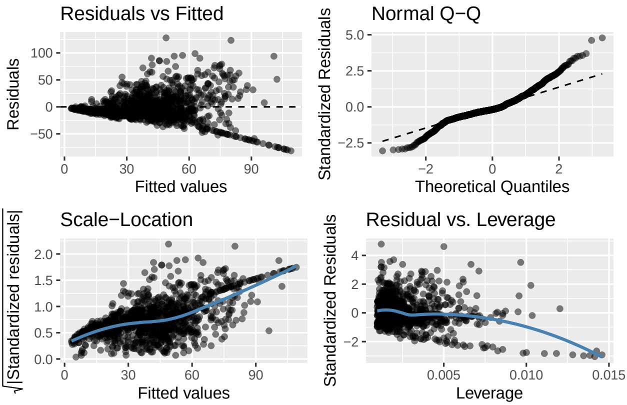
This seems like a good time to implement linear regression:

```
tree_lm<-lm(Carbon_Sequestration_lb ~Crown_Width_EW + Tree_Height, data=my_pdxTrees)
summary(tree_lm)
```

```
##
## Call:
## lm(formula = Carbon_Sequestration_lb ~ Crown_Width_EW + Tree_Height,
##     data = my_pdxTrees)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -81.46 -14.03  -4.92  11.51 127.87
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -1.06948    2.02087  -0.529    0.597
## Crown_Width_EW  0.79289    0.04624  17.146  < 2e-16 ***
## Tree_Height     0.12897    0.02820   4.573 5.38e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.74 on 1031 degrees of freedom
##   (5 observations deleted due to missingness)
## Multiple R-squared:  0.3699, Adjusted R-squared:  0.3687
## F-statistic: 302.7 on 2 and 1031 DF,  p-value: < 2.2e-16
```

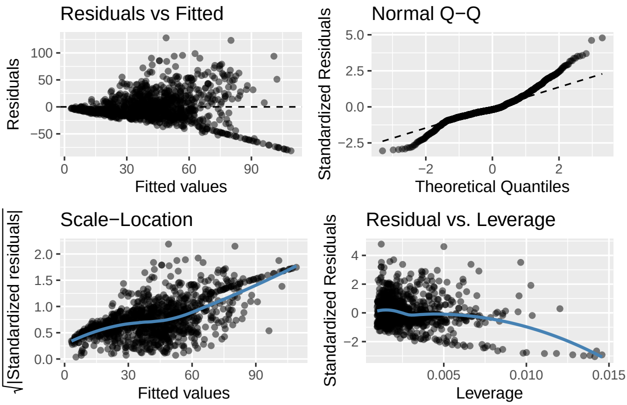
Diagnostic Plots

```
library(gglm)
gglm(tree_lm)
```



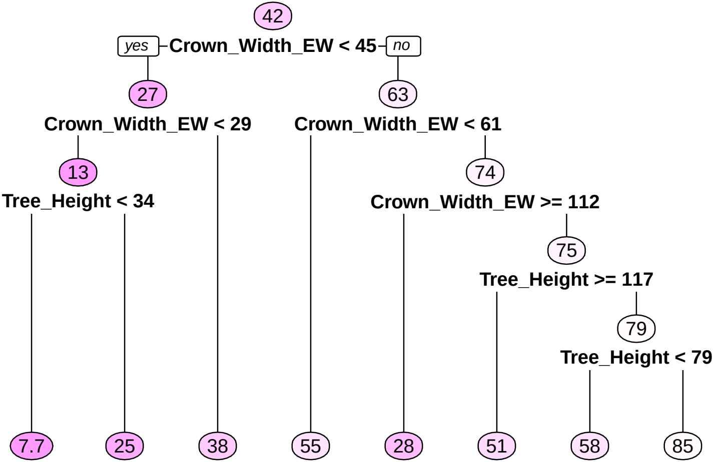
Diagnostic Plots

```
library(gglm)
gglm(tree_lm)
```

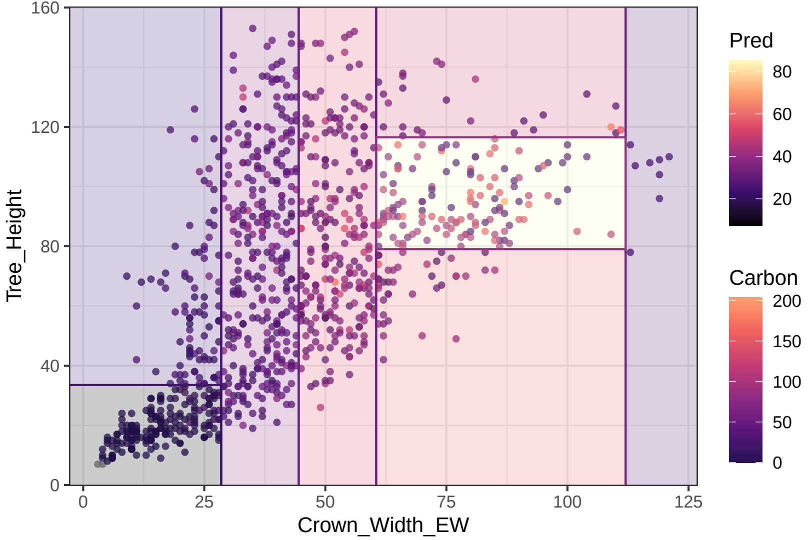


- Concerns?

Regression Tree



Another Visualization



Interpretation

- Crown_Width_EW is the most important factor contributing to Carbon_Sequestration_lb
- After accounting for width, Tree_Height has some impact on Carbon_Sequestration_lb
- Given a narrow tree, shorter trees tend to have lower Carbon_Sequestration_lb
- Given wide tree, moderately tall trees have largest Carbon_Sequestration_lb

Tree Accuracy

Let's create a test set consisting of parks further from Reed:

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

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
##   Tree_rMSE
## 1  16.35376
```

Tree Accuracy

Let's create a test set consisting of parks further from Reed:

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

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
##   Tree_rMSE
## 1  16.35376
```

And compared to the linear model:

```
##    lm_rMSE
## 1 31.77967
```

Tree Accuracy

Let's create a test set consisting of parks further from Reed:

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

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
##   Tree_rMSE
## 1  16.35376
```

And compared to the linear model:

```
##    lm_rMSE
## 1 31.77967
```

Why did the tree model outperform the linear model?

Tree Accuracy

Let's create a test set consisting of parks further from Reed:

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

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
##   Tree_rMSE
## 1  16.35376
```

And compared to the linear model:

```
##    lm_rMSE
## 1 31.77967
```

Why did the tree model outperform the linear model?

Nevertheless, what are some downsides to the tree model?

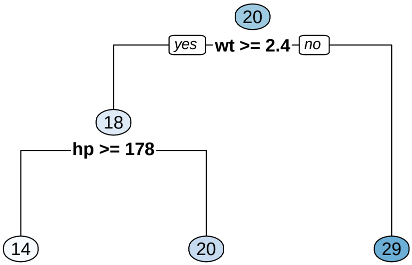
Extra Practice

The `mtcars` dataset gives the `mpg`, `hp`, and `wt` for 32 car models.

Extra Practice

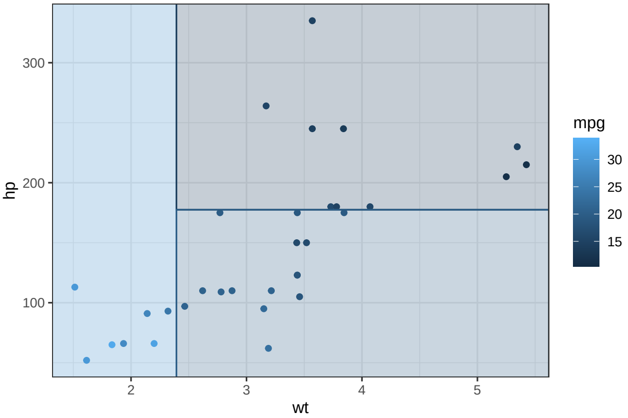
The `mtcars` dataset gives the `mpg`, `hp`, and `wt` for 32 car models.

In small groups, draw the predictor space corresponding to the following tree, predicting `mpg` based on `wt` and `hp`.



What would you expect the signs of the corresponding regression slopes to be?

Results




```
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 37.22727012 1.59878754 23.284689 2.565459e-20
## hp          -0.03177295 0.00902971 -3.518712 1.451229e-03
## wt          -3.87783074 0.63273349 -6.128695 1.119647e-06
```

Section 3

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 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 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 test 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 test 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 test 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