STATISTICS 452/652: Statistical Learning and Prediction

October 12, 2020

# Lecture 13: Regression Trees

**(Reading: Section 8.1 (skip 8.1.2 for now))**

## 1 Goals of lecture

- We develop another relatively automatic prediction machine that has several interesting properties
  - Automatic variable selection
  - Automatic detection of interactions
  - Operates directly on existing variables, rather than linear combinations or other engineered features
- The method is based on step functions (regression on indicator functions), like what we saw in Lecture 8.
  - *This time we let the data choose regions in each $X_j$ optimally!*
- These methods are (were?) enormously popular in many applications
  - But they are not actually very good at prediction!
  - We will see ways to improve them in later lectures.

## 2 Decision trees (Not a statistical technique)

1. Used in medicine and other decision-making ventures
2. Simplest form is constructed from a sequence of *binary* queries
   (a) Start from "root"
   (b) Query 1

i. If "true" go left; if "false" go right

ii. Create 2 "nodes"

(c) Query 2 for left side, 3 for right side

(d) Doesn't have to be the same second-level query on both sides of the first query

i. In each case, if true go left, if false go right.

ii. Create 4 more nodes

(e) Sequentially split previous nodes with new queries

(f) Eventually terminate with some kind of action assigned to the "terminal" nodes

# 3 Regression Trees

Decision trees built into regression functions. See Breiman, Friedman, Stone, and Olshen (1984), *Classification and Regression Trees*.

1. Regression trees fit a piecewise-constant model (a step function) to features $X_1, \ldots, X_p$ that has been partitioned into regions.

   (a) We will use the data to select regions optimally

   (b) Unfortunately, investigating all possible regions in $p$-dimensional space is computationally infeasible

      i. Like doing all-subsets selection with infinitely many possible variables

   (c) Instead, do a forward-stepwise selection of partitions

2. Partitioning takes place *recursively*

   (a) Split data according to levels of one variable (say, $X_j$) into two subsets:

$$\begin{aligned} R_1 &= \{\text{all } i \text{ for which} x_{ij} < c_1\} \\ R_2 &= \{\text{all } i \text{ for which} x_{ij} \geq c_1\} \end{aligned}$$

      i. Choose split to produce subsets "optimally" (to be defined later)

   (b) Repeat until some stopping criterion:

      i. Split one of the subsets created previously into two new subsets according to levels of one variable

   (c) **SEE EXAMPLE: L13 - Tree Demo.R**

      i. Just for fun, try this with other data.

   (d) See top right plot in Fig. 1.

3. End result is $M$ subsets of the data representing regions within $X$, $R_1, R_2, \ldots, R_M$

(a) Within $R_m$ use sample mean of data as prediction

$$\bar{y}_m = \frac{1}{n_m} \sum_{i \text{ in } R_m} y_i$$

$$\hat{Y}(X) = \sum_{m=1}^{M} I(X \text{ in } R_m)\bar{y}_m$$

(Remember indicator functions and step functions)

(b) See bottom right plot in Fig. 1.

4. Resulting fit is very "jumpy" and coarse

   (a) Step function with separate mean in each region

   (b) *Letting data choose regions optimally (more on this soon)*

   (c) Intent is to model large changes in means, ignore small ones

      i. A bias-variance tradeoff

5. Major questions:

   (a) How to select splits

   (b) How large a tree to grow

## 3.1 Split selection for Recursive Partitioning

We will start from the beginning with all training data in the ROOT NODE. Process for all subsequent regions is identical to the first one ("recursive" means repeating process on results from previous step).

1. A given split into subsets $R_1$ and $R_2$ is done on one variable at a time and has the form

$$\begin{aligned} R_1 &= \{\text{all } i \text{ for which} x_{ij} < c_1\} \\ R_2 &= \{\text{all } i \text{ for which} x_{ij} \geq c_1\} \end{aligned}$$

2. Goal is to select a split—variable $j$ and split location $c_1$—that **maximally reduces the RSS after the split compared to before**

$$RSS(\text{Full Data}) = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

$$RSS(\text{Split}) = \sum_{i \text{ in } R_1} (y_i - \bar{y}_1)^2 + \sum_{i \text{ in } R_2} (y_i - \bar{y}_2)^2$$

3. Perform "exhaustive search" through all possible split points

   (a) Numeric explanatory variables

Figure 1: Structure of a regression tree (From ISLR)



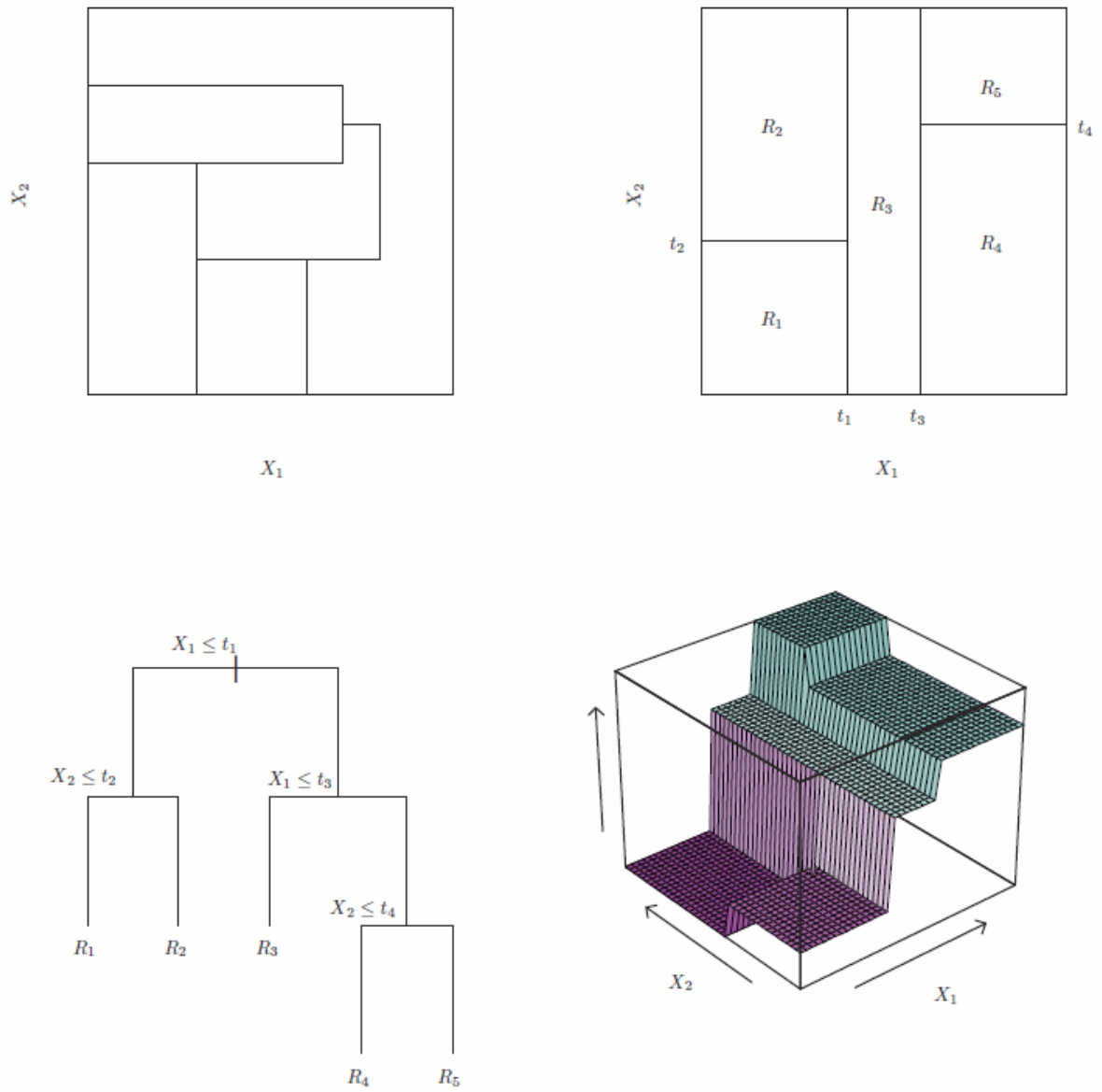



**FIGURE 8.3.** Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A perspective plot of the prediction surface corresponding to that tree.*

i. Compute $RSS$(Split) at all possible split locations $s$, where $s$ is the midpoint between any two consecutive values of a given variable $X_j$

ii. If all values of $x_{ij}$ are unique, then there are $n-1$ possible split locations in variable $X_j$

A. May be fewer if some values of $X_j$ have multiple observations of $Y$ (e.g., `pgg45` in prostate data)

(b) Categorical explanatory variables

i. Optimal search is across levels ordered according to mean $Y$ at each level

4. Once the best split is found, 2 new nodes are created

5. Repeat process on each region created by this split.

(a) New split may be on a different variable or on the same variable at a different level.

i. Splits of the same variable create additional means along that $X_j$

ii. Splits on a new variable create interaction between it and all previous split variables

A. Unless the *exact* same splits are made in both nodes of a previous split.

(b) See bottom plots in Fig. 1

## 3.2 Stopping rule

1. In theory, can keep splitting until each terminal node contains exactly one value of $x$ (possibly with multiple observations of $Y$).

(a) Each observation is its own prediction,

(b) Step-function interpolation, surely overfits

(c) Can take a long time to do the splits

2. Instead often terminate early using one or more rules.

(a) "Node size restrictions" put lower limit on number of obs in a terminal node

i. Don't allow a split to place fewer than $C_1$ observations in a node

(b) "Splitting restrictions" put a separate lower limit on the size of a node that may still be split

i. Don't consider splitting any node that has $< C_2$ observations in it

(c) Main R function, `rpart()`, uses default of $C_2 = 20$ (`minsplit`) , with $C_1 = 7$ (`minbucket`)

(d) Or stop splitting if no splits improve RSS "enough"

i. `rpart()` default is that no splits are made if best split improves $R^2$ by <1% (`cp=0.01`)

A. i.e., split reduces $RSS$ by $0.01RSS$(Full Data).
B. *I don't like this rule and almost always override it in the* `rpart()` *call by setting* `cp=0`.

3. Other rules exist
4. Parameters controlling rule are arbitrary
5. All rules the number of split searches!

### 3.3 Pruning a tree

In real life we PRUNE unnecessary (dead or unsightly) branches off of a tree when they make a tree look or perform worse than it could. In regression, we do the same thing.

1. My usual strategy for tree building is to deliberately build a tree that is too large, and then let CV tell us where we should have stopped.
   (a) Similar to running forward or stepwise to create full sequence of models with $0, 1, \ldots, p$ variables and letting an IC choose the best one
2. We can chop off (PRUNE) branches from the full tree according to how much they improve $RSS$ per added node
   (a) Much like using MSE for choosing regression model
3. Algorithm is a little complicated (see Appendix for details)
4. You can think of it this way: if the part of the tree below a given split does not actually improve CV error, then chop off the tree at that node.
5. Instead of minimizing CV error, can use the "1SE rule" like in LASSO
   (a) Use smallest model with CV error within 1SE of minimum.
6. After pruning, end result is a tree that is of "optimal" size.
   (a) All splits lead to eventual cost-complexity improvement of at least $\alpha$

## 4 Notes

### 4.1 LOADS of problems with regression trees

1. *Very* discrete, non-smooth predictions
   (a) Neighbouring observations can have very different predicted values. Recall tree demo.
2. *Very* variable and unstable

(a) If $p$ and $n$ are not both very small, *lots* of splits get considered in each node.

i. As in other variable selection procedures, there are often many possible splits with very similar reductions in $RSS$.

ii. Matter of luck (configuration of sample errors) which one gets chosen

iii. Small changes in data can result in different splits

A. If this happens early on, a *completely* different tree structure may result!

3. On the other hand, trees are consistent in an asymptotic sense

(a) "Best" splits stand out with near-infinite data

(b) Coarseness is overcome with many tiny jumps in gigantic data sets.

4. Trees have a hard time modeling smooth relationships (including linearity!)

(a) Piecewise constant, not a smooth change (slope) anywhere.

(b) Need to have multiple splits along one variable in order to approximate anything smooth

i. Split once on $X_j$

ii. Split each side on $X_j$ again (refine previous subset)

iii. Keep repeating

iv. Recall tree demo

(c) Often thrown off track due to so many other variables' splits to choose from

## 4.2 People are undaunted

Despite these flaws, regression trees are one of the most popular "modern" data mining tools around!

1. Automatic inclusion of interactions (no thinking required)

2. If tree is not too large, decision tree is easy to interpret

(a) If it's large, it's a nightmare! See Fig 2.

Figure 2: Gigantic Classification Tree



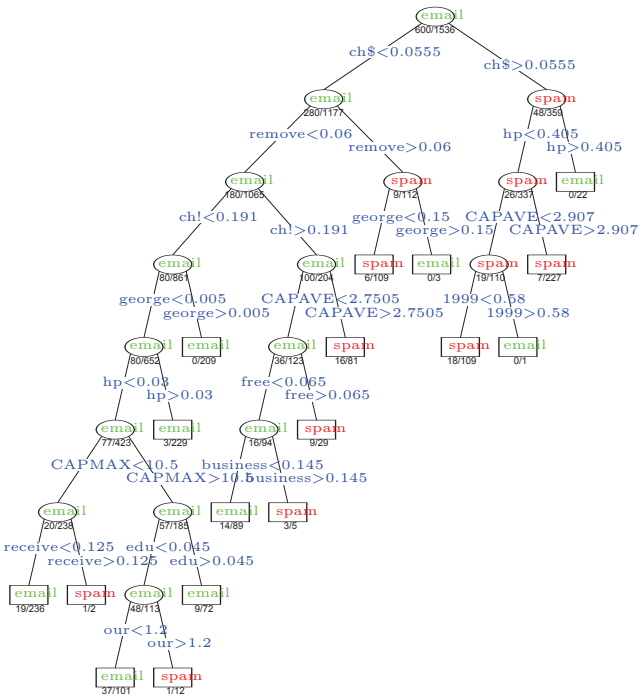



**FIGURE 9.5.** *The pruned tree for the* `spam` *example. The split variables are shown in blue on the branches, and the classification is shown in every node. The numbers under the terminal nodes indicate misclassification rates on the test data.*

## 4.3 Good news

1. Although single trees have lots of problems, regression trees can be altered into, or form the basis of, various prediction methods that are among the most successful that have been proposed thus far!

   (a) Random forests

   (b) Boosted trees

2. However, the cost is the pretty picture that these methods cannot construct...

3. There are many variants of regression trees that alter the base algorithm to improve performance in one way or another.

   (a) Still not better than random forests or boosting,

**Example: Regression Trees on Prostate Data (L13 - Regression Tree Prostate.R)**

Two main packages, `rpart` and `tree`, do standard regression trees (several others do versions of regression trees); `rpart` is the more modern of the two and is recommended. The `rpart()` function performs the analysis:

```
rpart(formula, data, weights, subset, na.action = na.rpart,
method, model = FALSE, x = FALSE, y = TRUE, parms, control,
cost, ...)
```

Also, `control` parameters can be added directly into the call:

```
rpart.control(minsplit = 20, minbucket = round(minsplit/3),
cp = 0.01, maxcompete = 4, maxsurrogate = 5, usesurrogate =
2, xval = 10, surrogatestyle = 0, maxdepth = 30, ...)
```

They do all pruning calculations using "relative" error, where the total $RSS$ for the root (first) node of the tree is considered "100%" or 1.0, and all values are displayed as fractions of the total error. This relates them to $R^2$ values. This does not affect the outcome at all.

In my example, I first do a simple analysis of the two variables, `lcavol` and `pgg45` so that we can compute a surface and compare it to other methods. The output from the fit is below.

```
> pr.tree <- rpart(lpsa ~ lcavol + pgg45, method="anova", data=prosta
> pr.tree
n= 97

node), split, n, deviance, yval
      * denotes terminal node

 1) root 97 127.917700 2.4783870
   2) lcavol< 2.46165 76  67.267100 2.1227440
     4) lcavol< -0.4785564 9   5.597501 0.6016839 *
     5) lcavol>=-0.4785564 67  38.049940 2.3270650
      10) pgg45< 5.5 32  17.504720 2.0033210
        20) lcavol< 0.7744616 15   8.761123 1.7709770 *
        21) lcavol>=0.7744616 17   7.219355 2.2083300 *
      11) pgg45>=5.5 35  14.124840 2.6230600
        22) lcavol< 1.050767 8   1.516646 2.1143990 *
        23) lcavol>=1.050767 27   9.925007 2.7737740 *
   3) lcavol>=2.46165 21  16.249280 3.7654770
     6) lcavol< 2.793517 10   2.885196 3.2839210 *
```

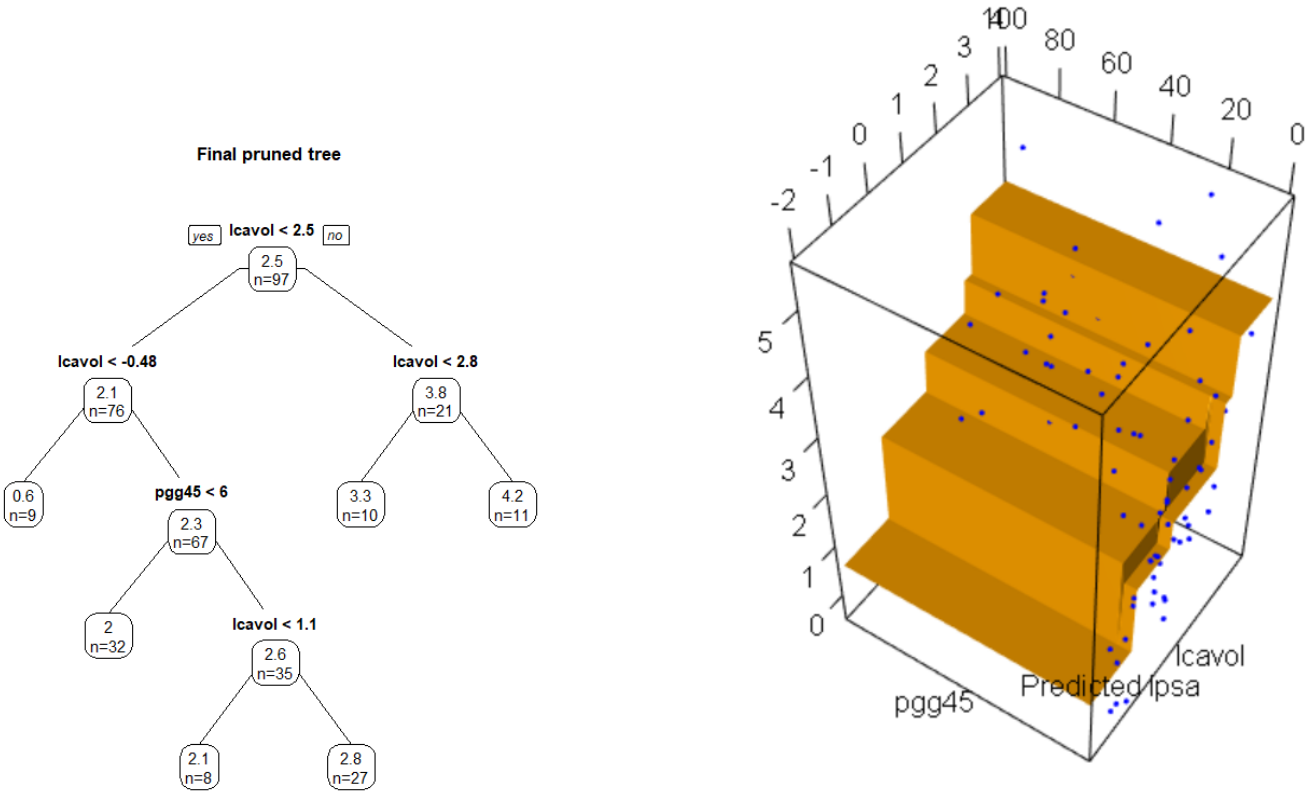
```
      7) lcavol>=2.793517 11   8.936978 4.2032550 *
> pr.tree$cptable
          CP nsplit rel error    xerror       xstd
1 0.34710828      0 1.0000000 1.0329093 0.16557818
2 0.18464743      1 0.6528917 0.8219941 0.11428573
3 0.05019151      2 0.4682443 0.6377170 0.08113814
4 0.03460901      3 0.4180528 0.5671979 0.07492809
5 0.02097586      4 0.3834438 0.5638782 0.07432116
6 0.01191581      5 0.3624679 0.5530571 0.07488120
7 0.01000000      6 0.3505521 0.5396268 0.07248884
> # I like this order better!
> pr.tree$cptable[,c(2:5,1)]
  nsplit rel error    xerror       xstd          CP
1      0 1.0000000 1.0329093 0.16557818 0.34710828
2      1 0.6528917 0.8219941 0.11428573 0.18464743
3      2 0.4682443 0.6377170 0.08113814 0.05019151
4      3 0.4180528 0.5671979 0.07492809 0.03460901
5      4 0.3834438 0.5638782 0.07432116 0.02097586
6      5 0.3624679 0.5530571 0.07488120 0.01191581
7      6 0.3505521 0.5396268 0.07248884 0.01000000
```

Printing the fitted object shows statistics on all of the splits and resulting nodes created by each split. For example, the root node contains 97 observations with $RSS = 127.9$ and a mean of 2.12. The first split leads to nodes 2) and 3), where the split is made on `lcavol`$< 2.46$ (76 obs) and `lcavol`$\geq 2.46$ (21 obs). Further splits within each of these nodes are indented beneath the respective regions.

The `cptable` element of the object shows the value of each split (`CP`), the resulting $RSS$ (`rel error`) and CV error (`xerror`), both divided by the root node $RSS$.

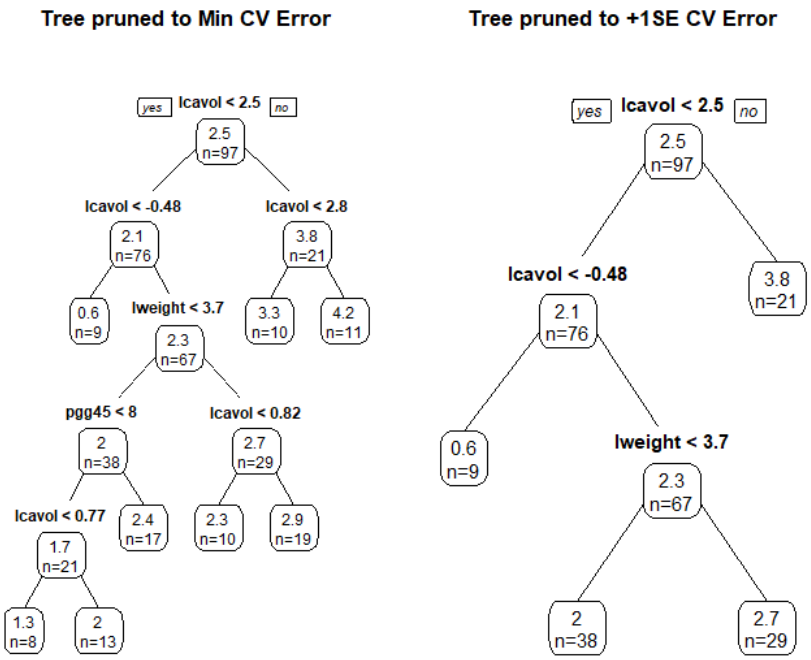
The pruned tree and resulting surface are shown in Figure (3) .

Figure 3: Fitted tree and surface for Prostate data using only `lcavol` and `pgg45`.




Next I do an analysis of the full data, including automatic calculation of the min-CV and 1SE optimal pruning. These produce the trees shown in Figure (4).

Figure 4: Fitted tree and surface for Prostate data using only `lcavol` and `pgg45`.




# 5 What to take away from this

1. Regression trees are a radically different form of surface, stepwise constant and not smooth
   (a) They are unstable, as small changes in data can result in very different trees
2. If there is enough data, they are low-bias, high-variance machines.
   (a) Not especially useful by themselves
   (b) GREAT in "ensembles" as we will see soon.
3. People use them, but there are almost always better prediction functions out there
   (a) The tree plot is kind of useful sometimes, though.

# 6 Appendix (Not covered, just for those who are curious)

1. Basic idea: is to ask what would happen if we stopped splitting at some internal node $q$
2. Compute *RSS in that node* (i.e., before it was split), $RSS(q)$

(a) Look at the entire "sub-tree" $T_q$ consisting of all subsequent splits and terminal nodes made from the data in node $q$

i. Let $|T_q|$ be the number of terminal nodes in $T_q$

(b) The "model" represented by $T_q$ is the set of means from all terminal nodes that eventually result from the split on $q$

i. $RSS(T_q)$ for this model is sum of $RSS$ in these terminal nodes

ii. This sum is smaller than $RSS(q)$, so $RSS(q) - RSS(T_q)$ represents the improvement in error resulting from not stopping at $q$

3. The "cost" of this improvement is added complexity from using $|T_q|$ means in $T_q$ rather than single mean in $q$, where $|T_q|$ means the number of terminal n

(a) $|T_q| - 1$ added mean parameters

4. The mean square for improvement caused by continuing past $q$ (called the COST-COMPLEXITY):

$$MS(q) = \frac{RSS(q) - RSS(T_q)}{|T_q| - 1} \tag{1}$$

5. TO USE THIS FOR PRUNING THE TREE:

(a) Find nodes for which there is little improvement from to continuing past the node

i. Order all nodes from worst to best in terms of cost-complexity

(b) Choose a threshold of cost complexity, $\alpha$, that must be surpassed in order to retain a split.

(c) Starting at the top, chop off all branches (sub-trees) below node $q$ if $MS(q) < \alpha$ for some chosen level $\alpha$

(d) $\alpha$ is a tuning parameter that controls the size of the tree

i. Balances decreasing $RSS$ against increasing number of parameters

A. Sort of like a regularization parameter for size of tree

B. Bias-Variance tradeoff!

ii. Can be chosen by CV

iii. Relates to `cp` parameter described earlier, but `cp` is computed differently in `rpart()`and relates only to result of immediate split at $q$, not eventual full subtree below it

# 7 Exercises

## Application

Refer to the Air Quality data described previously, and the analyses we have done with `Ozone` as the response variable, and the five explanatory variables (including the two engineered features).

Use regression trees to model the relationship between `Ozone` and all five explanatories as specified below.:

1. Fit a regression tree using only `Temp` and `Wind`, using `cp=0`.
   (a) **Print out the cp table.**
      i. According to minimum CV, **what value of `cp` should be used for optimal pruning, and what is the relative CV error at this value?**
   (b) **Plot both the min-CV pruned tree and the 1SE pruned tree.**
      i. Interpret the first split on the root node: **What variable and split location are used is used, how many observations go each way, and what are the new means in each node?**
2. Add three new models fitted to all 5 variables to the 10-fold CV comparison that has been used for LASSO, Ridge, and other methods: Full tree (`CP=0`), min-cv-pruned tree, and 1-se-pruned tree. Use the same folds as before.
   (a) **Compute the mean MSPE for each tree model and comment on the comparison.**
   (b) **ADD the three trees to the relative MSPE boxplots made previously. Comment on how well they perform compared to other methods.**