We now elaborate on Step 1 above. How do we construct the regions $R_1, \dots, R_J$?

- We choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity.
- The goal is to find boxes $R_1, \dots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.
- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes. For this reason, we take a top-down, greedy approach that is known as recursive binary splitting.
  - We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X | X_j < s\}$ and $\{X | X_j \geqslant s\}$ leads to the greatest possible reduction in RSS.

That is, for any j and s, we define a pair of half planes

$$R_1(j,s) = \{X | X_j < s\}, \quad R_2(j,s) = \{X | X_j \geqslant s\},$$

and we seek the value of j and s that minimize

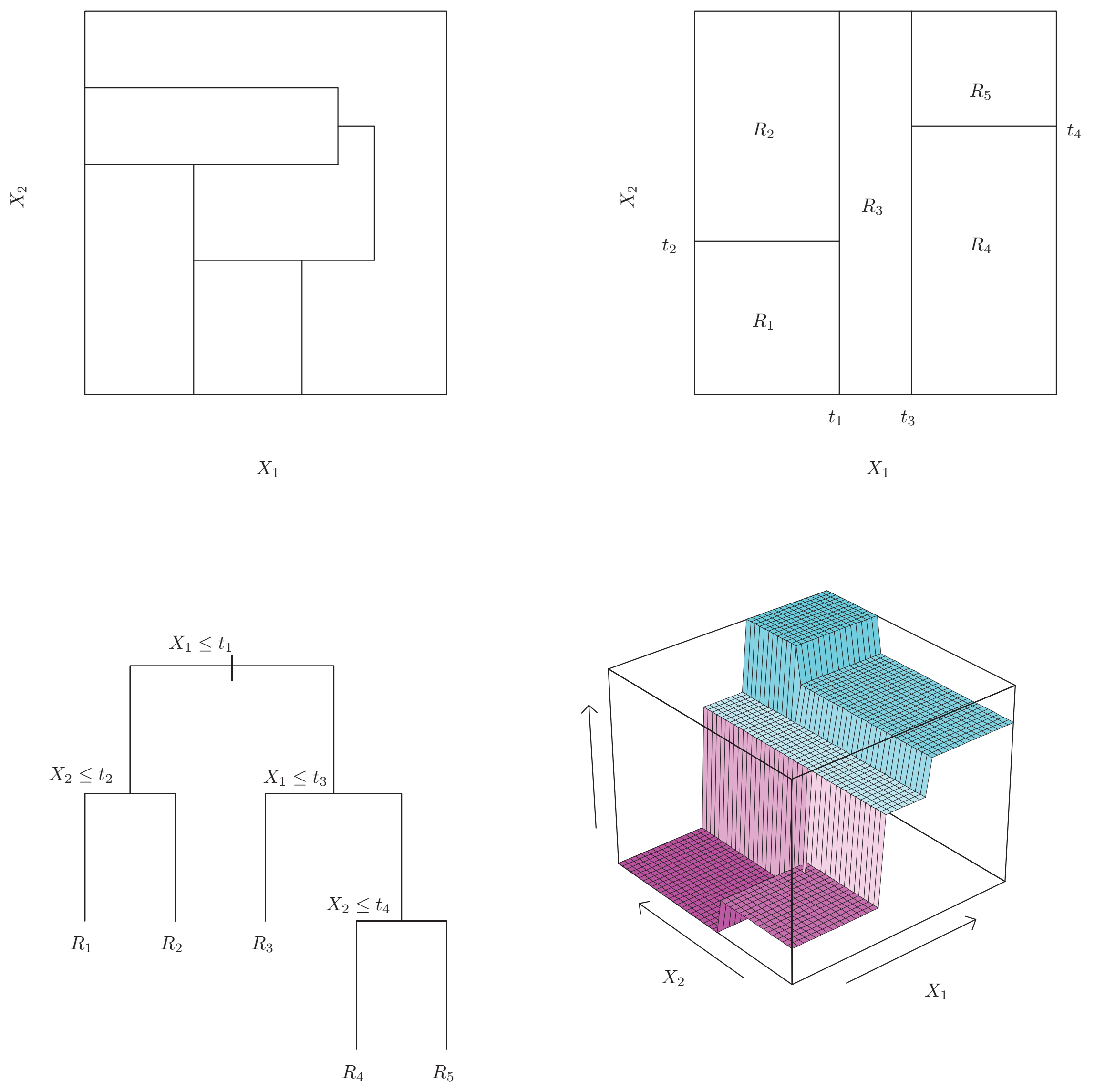
$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

where $\hat{y}_{R_i}$ is the mean response for the

training observations in $R_i(j,s)$, $i = 1,2$.

- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS.
- The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Once the regions $R_1, \ldots, R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.



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.

Tree Pruning

- The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.

This is because the resulting tree might be too complex.

- A smaller tree with fewer splits (that is, fewer regions $R_1, \dots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to build the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too short-sighted since a seemingly worthless split early on in the tree might be followed by a very good split—that is, a split that leads to a large reduction in RSS later on.
- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree.

Our goal is to select a subtree that leads to the lowest test error rate.

- Cost complexity pruning—also known as weakest link pruning—gives us a way to do just this. Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α.
- For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T| \quad (\ast)$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the predicted response associated with R_m—that is, the mean of the training observations in R_m.

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.

When $\alpha = 0$, then the subtree T will simply equal T_0, b/c then $(*)$ measures the training error. As α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity $(*)$ will tend to be minimized for a smaller subtree.

- We can select a value of α using a validation set or using cross-validation. We then return to the full data set and obtain the subtree corresponding to α. This process is summarized in the following algorithm.

Building a Regression Tree

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

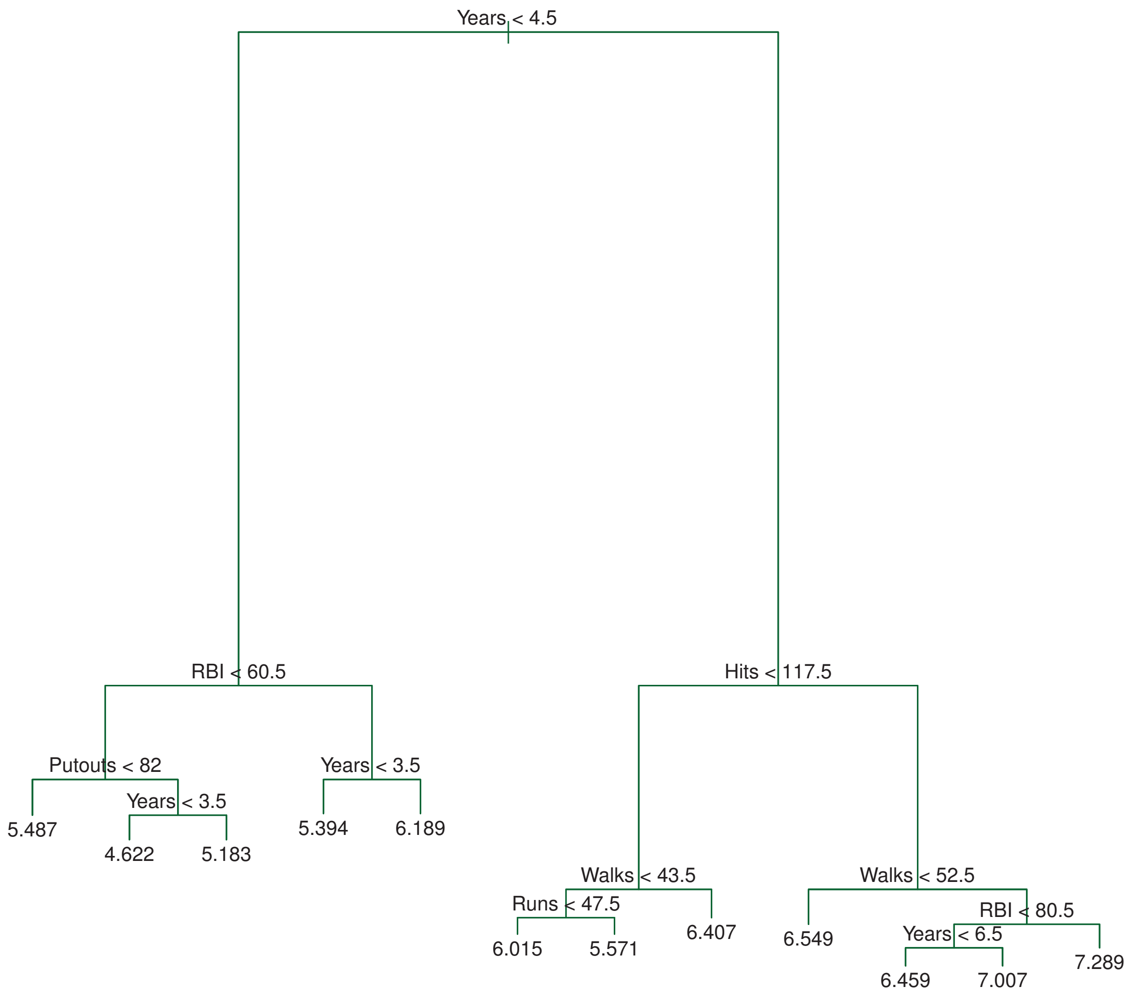
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

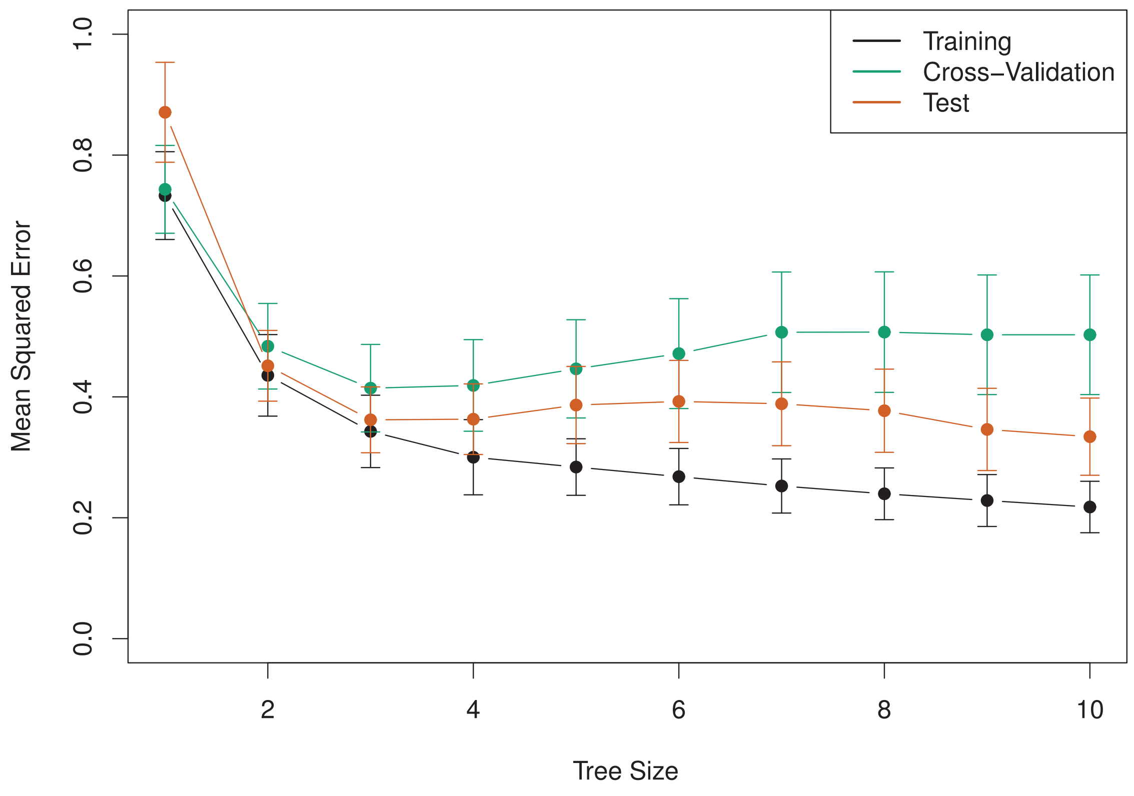
   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

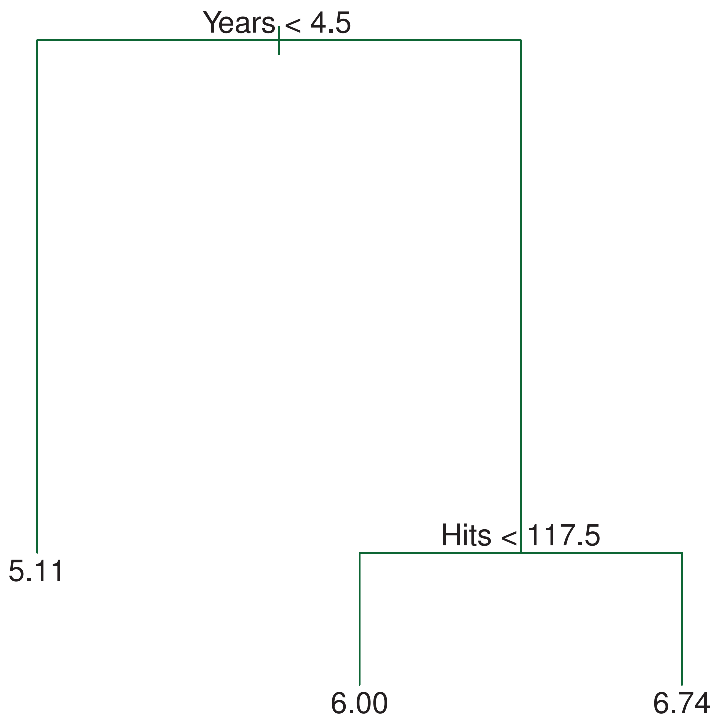
Following figures display the results of fitting and pruning a regression tree on the Hitters data, using nine of the features. First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set. We then built a large regression tree on the training data and varied α in order to create subtrees with different numbers of terminal nodes. Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees.



Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown.



Regression tree analysis for the Hitters data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.



The pruned tree containing three terminal nodes.

Classification Trees

- A classification tree is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.
- In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region.
- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits.
- A natural alternative to RSS is the classification error rate. This is simply the fraction of the training observations in that region that do not belong to the most common class:

  $$E = 1 - \max_k(\hat{p}_{mk})$$

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- However, classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.
- The Gini index is defined by

  $$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

  a measure of total variance across the K classes.

- The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node purity

A small value indicates that a node contains predominantly observations from a single class.

- An alternative to the Gini index is entropy, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Since $0 \leqslant \hat{p}_{mk} \leqslant 1$, it follows that $0 \leqslant -\hat{p}_{mk} \log \hat{p}_{mk}$. The entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one.

The entropy will take on a small value if the mth node is pure.