Chapter 8 Tree-based Methods

C. Devon Lin

Queen's University, Nov 7, 2019

Outline

- 8.1 Decision Trees
- 8.2 Bagging, Random Forests, Boosting

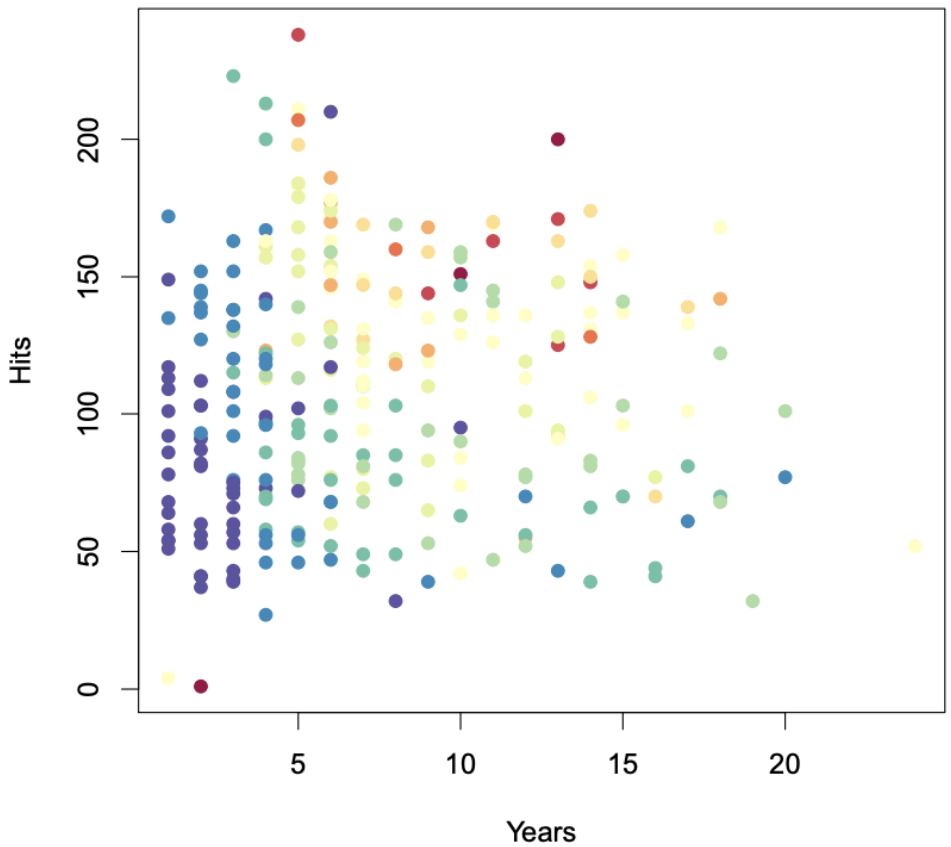
Reference: 8.1–8.3 of ISLR

8.1 Decision Trees

- Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one.

Baseball salary data

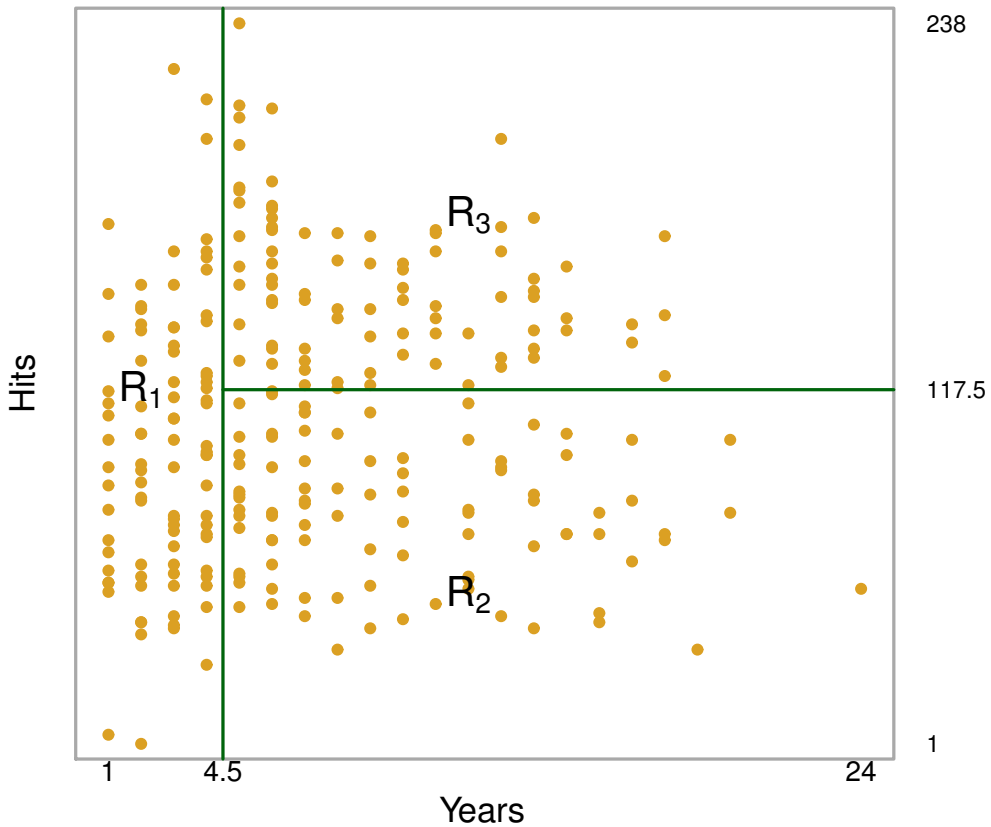
Salary is color-coded from low (blue, green) to high (yellow,red)



Decision tree

A regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.

Regions of predictor space



Terminology for Trees

- **Terminal nodes**: the regions of predictor space
- **Internal nodes**: The points along the tree where the predictor space is split.
- **Branch**: the segments of the trees that connect the nodes
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.

Interpretations

- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary
- But among players who have been in the major leagues for five or more years, the number of Hits made in the previous year does affect Salary, and players who made more Hits last year tend to have higher salaries.

Process of building a regression tree

1. We divide the predictor space. That is, the set of possible values for $X_1, X_2, \ldots, X_p$ into J distinct and non-overlapping regions, $R_1, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j

Building a regression tree

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model
- The goal is to find boxes $R_1, R_2, \ldots, R_J$ that minimize the

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

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Building a regression tree

- 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.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step

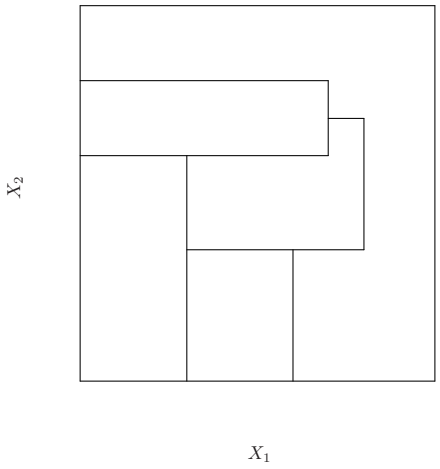
Building a regression tree

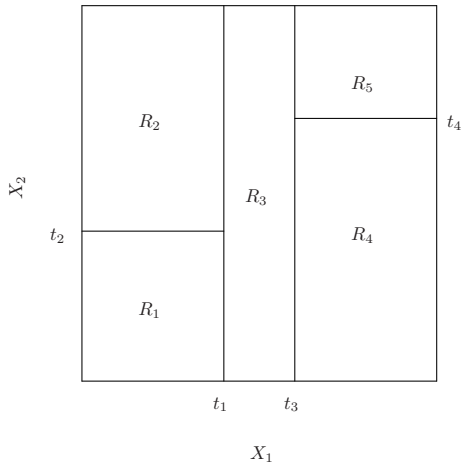
- We first select the predictor X_j and the cut-point s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cut-point in order to split the 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.

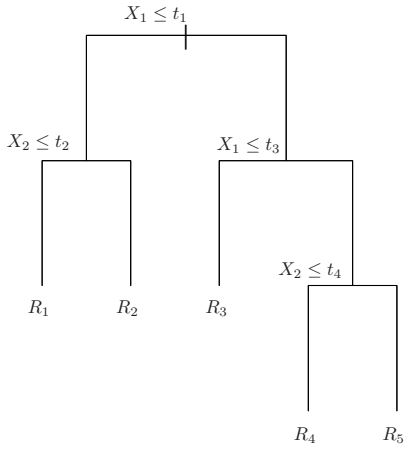
Predictions

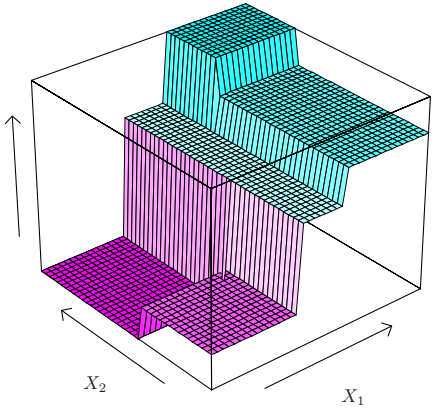
- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

A five-region example









Pruning a tree

- 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
- A smaller tree with fewer splits (that is, fewer regions) might lead to lower variance and better interpretation at the cost of a little bias.
- A good strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree
- *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.

Pruning a tree

- 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

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

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 mean of the training observations in R_m.

Building a Regression Tree

Algorithm 8.1 *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.

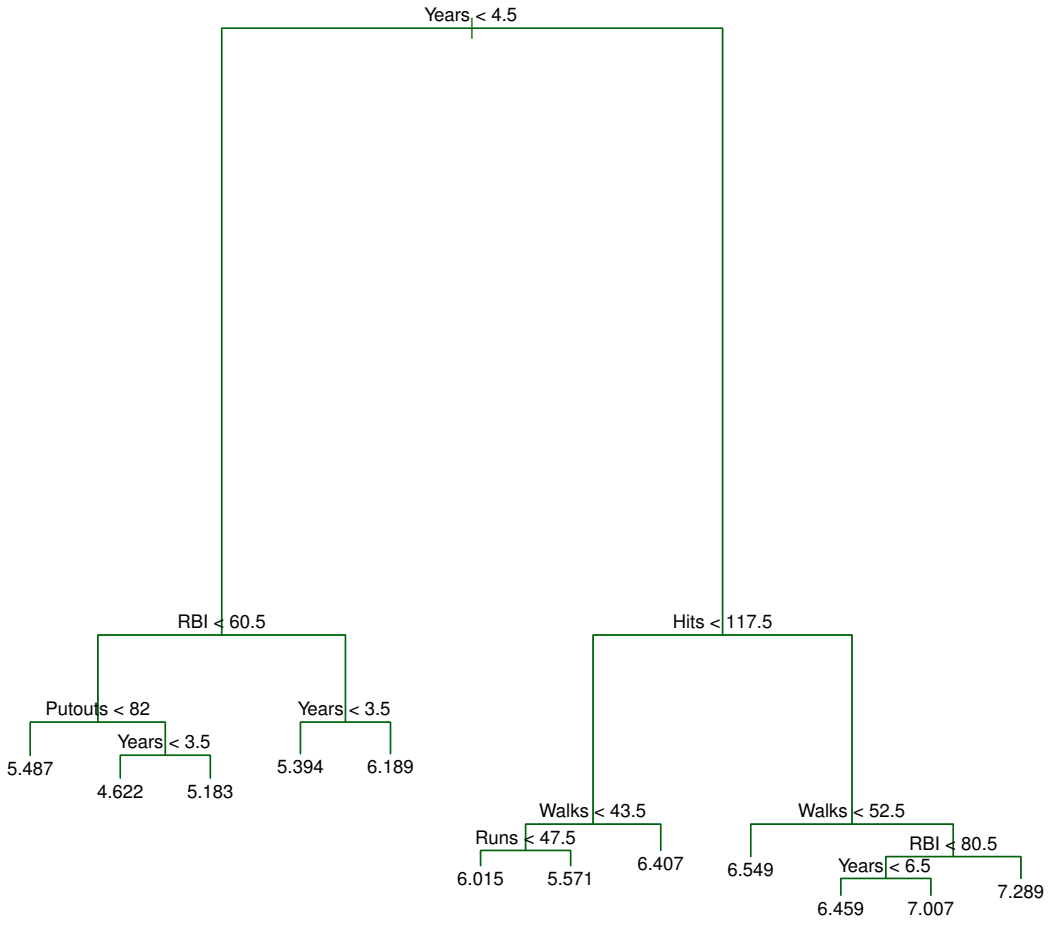
Choosing a best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

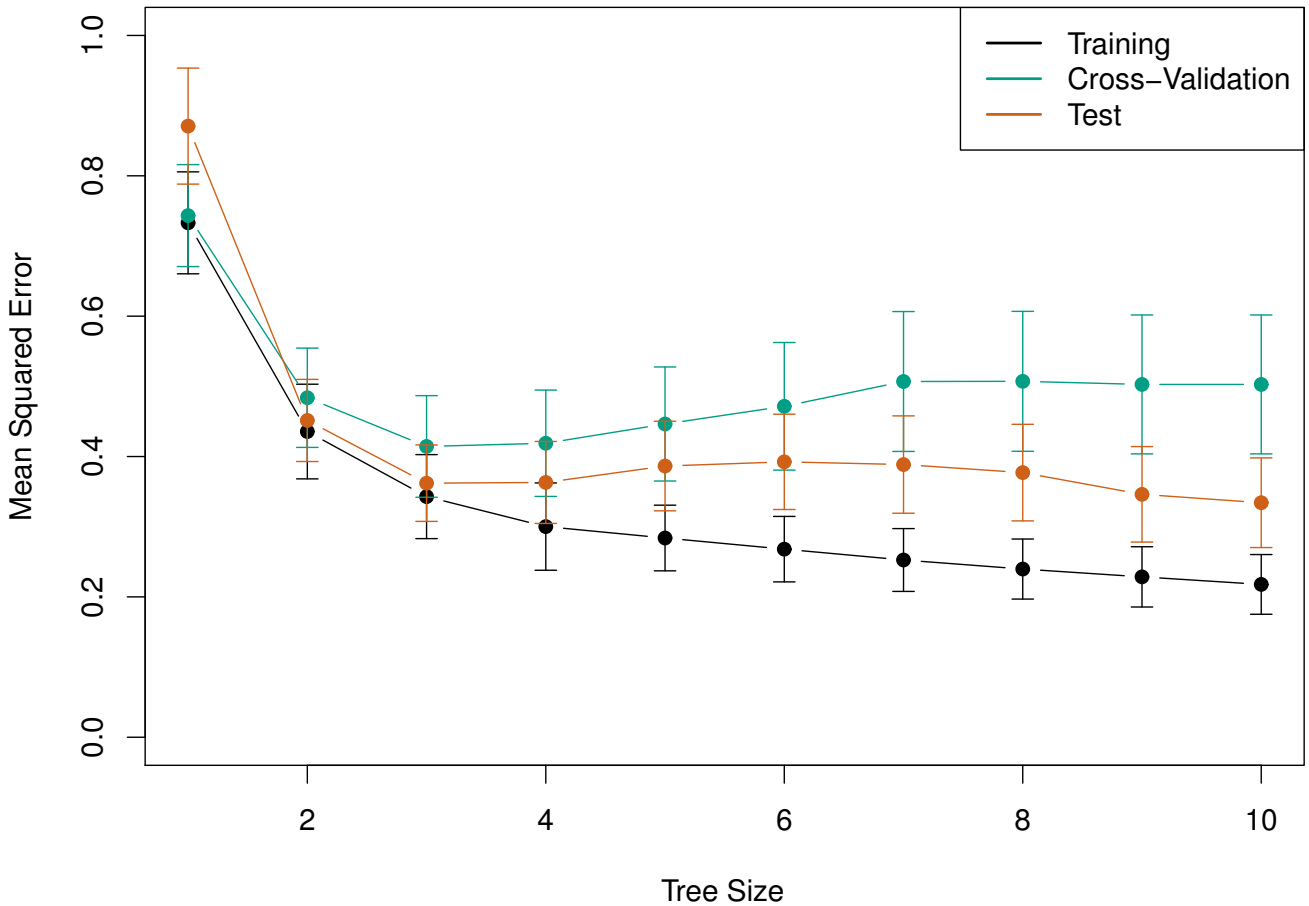
Baseball example

- 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 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 as a function of α.

Tree for baseball example



Regression tree analysis for the Hitters data



Regression tree analysis for Boston data

```
library (MASS)
set.seed (11112)
train = sample (1: nrow(Boston ), nrow(Boston)/2)
tree.boston = tree(medv ~ .,Boston ,subset =train)
summary(tree.boston)

# fitted tree
plot(tree.boston )
text(tree.boston ,pretty =0)
```

Regression tree analysis for Boston data

```
##use the cv.tree() function to see whether pruning the tree will impro
cv.boston =cv.tree(tree.boston )
plot(cv.boston$size,cv.boston$dev ,type='b')

#prune the tree
prune.boston =prune.tree(tree.boston ,best = 8)
plot(prune.boston )
text(prune.boston ,pretty =0)
```

Regression tree analysis for Boston data

```
# prediction
yhat=predict(tree.boston,newdata =Boston [-train ,])
boston.test=Boston [-train ,"medv"]
plot(yhat,boston.test)
abline(0,1)
mean((yhat-boston.test)^2)
```

Classification Trees

- 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.

Classification error rate

- 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:

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

  where $\hat{p}_{mk}$ 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.

Gini index and cross-entropy

- 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.

- Cross-entropy is given by

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

Classification tree analysis for carseats data

```
## create a new variable
High=ifelse(Sales <=8," No"," Yes ")

##
Carseats =data.frame(Carseats ,High)

##fit a classification tree in order to predict  High using all variabl
tree.carseats =tree(High ~ .-Sales,Carseats )

## summary of the fitting
summary(tree.carseats )

### plotting
plot(tree.carseats )
text(tree.carseats ,pretty =0)
tree.carseats
```

Classification tree analysis for carseats data

```
##
cv.carseats =cv.tree(tree.carseats ,FUN=prune.misclass)
names(cv.carseats)
cv.carseats

##plot the error rate as a function of both size and k.
par(mfrow =c(1,2))
plot(cv.carseats$size ,cv.carseats$dev ,type="b")
plot(cv.carseats$k ,cv.carseats$dev ,type="b")
```

Classification tree analysis for carseats data

```
##
prune.carseats =prune.misclass(tree.carseats ,best =8)
tree.carseats

## plot the classification after pruning
plot(prune.carseats )
text(prune.carseats ,pretty =0)

## prediction
tree.pred=predict (prune.carseats , Carseats.test ,type=" class ")
table(tree.pred ,High.test)
```

Bagging

- Bagging is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- We generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$ the prediction at a point x. We then average all the predictions to obtain

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

This is called *Bagging*.

Bagging classification trees

- The above prescription applied to regression trees
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.

Random forest

- When building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- One choice is $m = \sqrt{p}$ - the number of predictors considered at each split is approximately equal to the square root of the total number of predictors
- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.

Boosting

- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model
- Boosting works in a similar way as bagging, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Boosting algorithm for regression trees

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Boosting

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Tuning parameters for boosting

- The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
- The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance
- The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Summary

- Decision trees: regression and classification
- Bagging, random forest and boosting