CH. 8. Tree_based Methods_ISL 2019.11.23

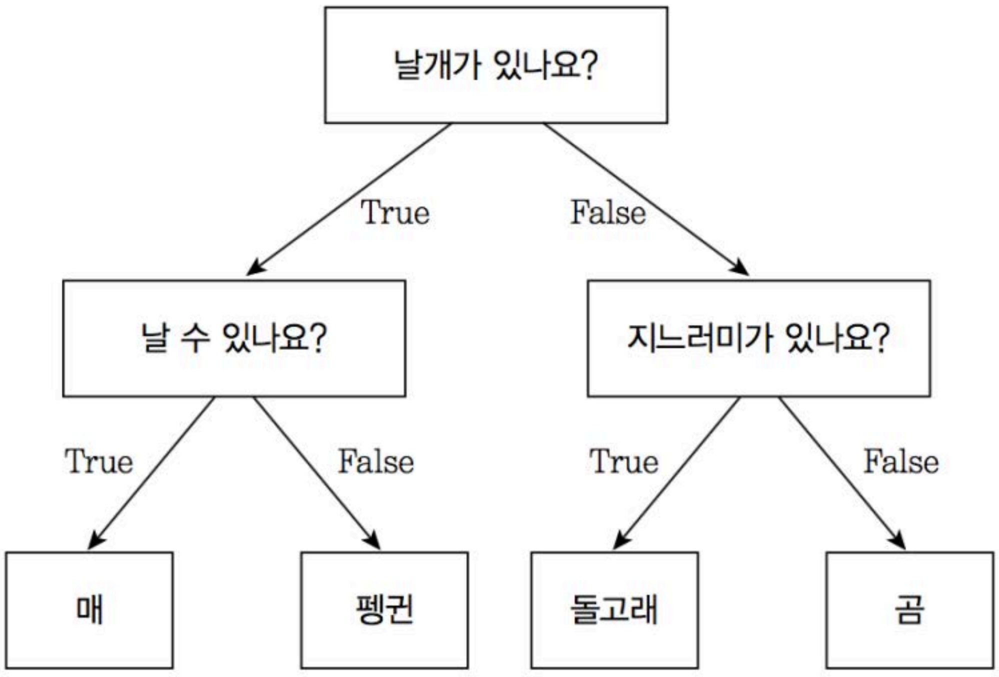
In Chapter 8, we describe tree-based methods for regression and classification.

Tree-based methods:

- involves stratifying or segmenting the predictor space into a number of simple regions.
- Simple and useful for interpretation.
- However, not competitive with the best supervised learning approaches in terms of prediction accuracy
- Combining a larger number of trees

  $\rightarrow$dramatic improvement in prediction accuracy (Cost : some loss in interpretation)

e



8.1 The Basics of Decision Trees

Decision Trees can be applied to both regression and classification problems.

8.1.1 Regression Trees

g Prediction Baseball Player's Salaries

Predict **Salary** with **Years, Hits.**

Preprocessing: missing Salary Values remove; Salary log-transform bell_shape distri.

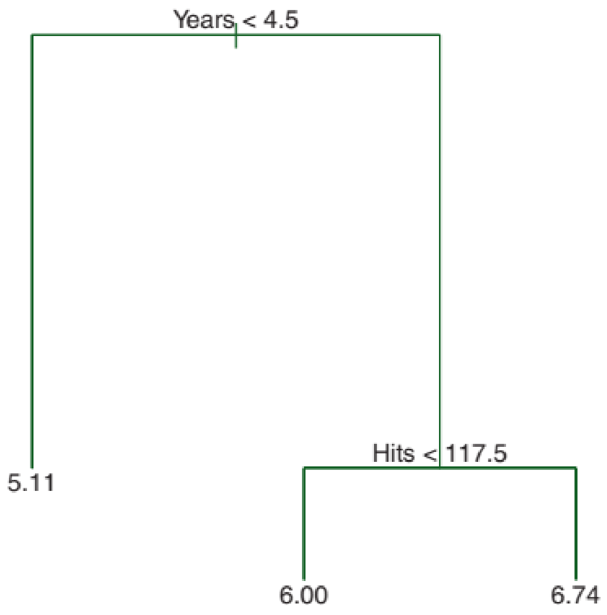


FIGURE 8.1. *For the* `Hitters` *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to* `Years<4.5`*, and the right-hand branch corresponds to* `Years>=4.5`*. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.*

It consists of a series of splitting rules, starting at the top of the tree.

Salary 예측 < years, hits . 결측치 제거, log transform

Predicted salary is given by the mean response in each leaf. (Note, this is log-transform

val)

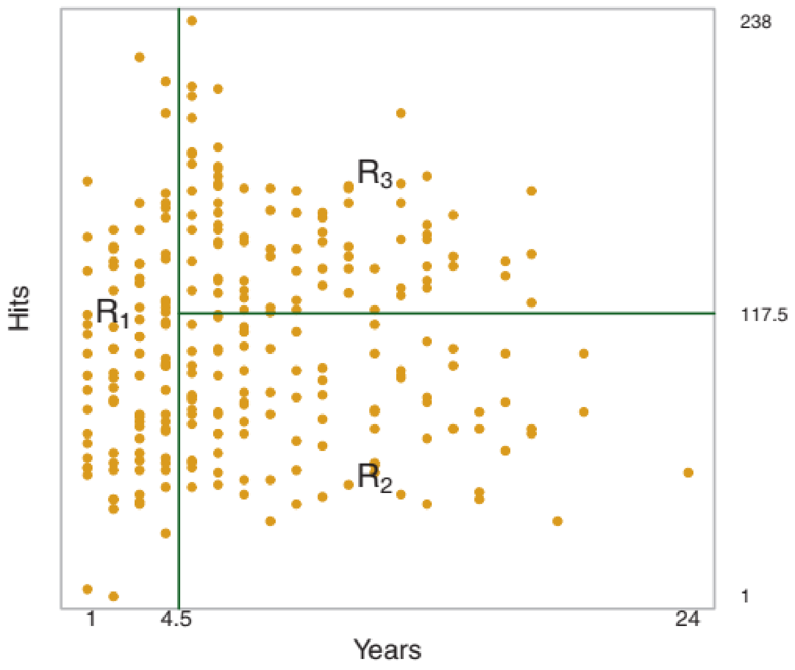


FIGURE 8.2. *The three-region partition for the* Hitters *data set from the regression tree illustrated in Figure 8.1.*

Overall, the tree stratifies the players into three regions of predictor space.

Each of the region R_1, R_2, R_3 is called **terminal nodes** or **leaves**.

Points along the tree where the predictor space is split: **internal nodes**. (Year=4.5, Hits=117.5)

Decision trees are typically drawn **upside down**.

$R_1 = \{X|Years < 4.5\}$,

$R_2 = \{X|Years \geq 4.5, Hits < 117.5\}$,

$R_3 = \{X|Years \geq 4.5, Hits \geq 117.5\}$

Predicted Salaries: $1000 * e^{5.107} = 165174, 1000 * e^{5.999} = 402,834, e^{6.740} = 845346$

- It can be interpreted as: Year is the most important factor in determining salary.
- For less experienced players: Hits is not important.

  For experienced players: Hits is important

This tree is probably an oversimplification of the relationship between Hits, Years, and Salary.

However, it is easier to **interpret**, and has a **nice graphical** representation.

Prediction via Stratification of the Feature Space

The process of building a regression tree (Roughly):

1. Divide the predictor space (the set of possible values for $X_1, X_2, \ldots, X_p$) into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

2. For every obs in R_J we make the same prediction. (The mean of the response values for the training obs in R_J).

That is:

IF the mean training response of R_1 is 10, and the mean of R_2 is 20

For $X = x$, if $x \in R_1 \rightarrow$ predict 10

if $x \in R_2 \rightarrow$ we predict 20.

How to construct the regions $R_1, R_2, \ldots, R_J$:

In theory, the regions could have **any shape**.

However, we choose to divide the predictor space into **high-dimensional rectangle**, or **boxes**. For **simplicity** and **ease of interpretation**.

The goal is to find boxes $R_1, R_2, \ldots, R_J$ that minimize the RSS. ($\sum_{j=1}^{J} \sum_{i \in R_J} (y_i - \hat{y}_{R_j})^2$).

$\hat{y}_{R_j}$: mean response for the training obs in jth box.

It is impossible to computationally infeasible to consider every possible partition of feature space into J boxes.

We use a **top-down** (begins at the top of the tree) **greedy** (at each step the best split is made) approach. a.k.a **recursive binary splitting**.

Recursive binary splitting:

1. select the predictor X_j and the cutpoint s s.t 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.

   In more detail:

   we consider all predictors $X_1, \ldots, X_P$ and all possible values of the cutpoint s for each of the predictors. and choose the predictor and s.t the resulting tree has the lowest RSS.

   In more more detail:

   for any j and s, we define the pair of half-planes

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

   find j, s s.t

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

   is minimized.
2. Repeat the step1 to split one of the previously identified regions.
3. continue until a stopping criterion is reached.

   (e.g until no region contains more than 5 obs)

4. Predict response: $\hat{y}_{R_j}$ is the prediction.

Tree Pruning

The process described above (Recursive binary splitting with a stopping criterion: no region contains more than 5 obs) is likely to **overfit** the data. (Too complex)

We could build the tree only with a different stopping criterion to build a smaller tree.

with a smaller variance, better interpretation at the cost of a little bias

However, this is short-sighted.
A seemingly worthless split early on in the tree might be followed by a very good split.

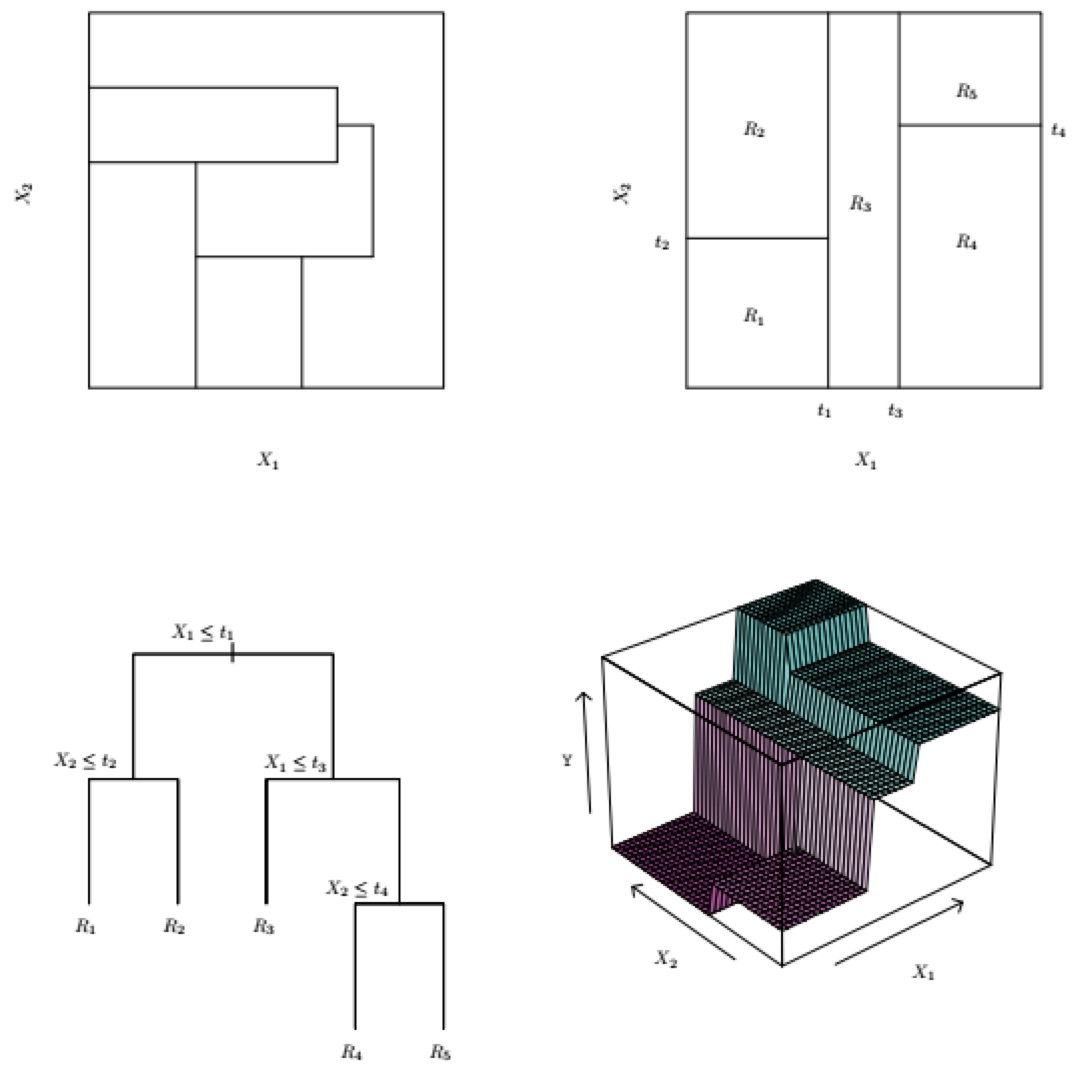


FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A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

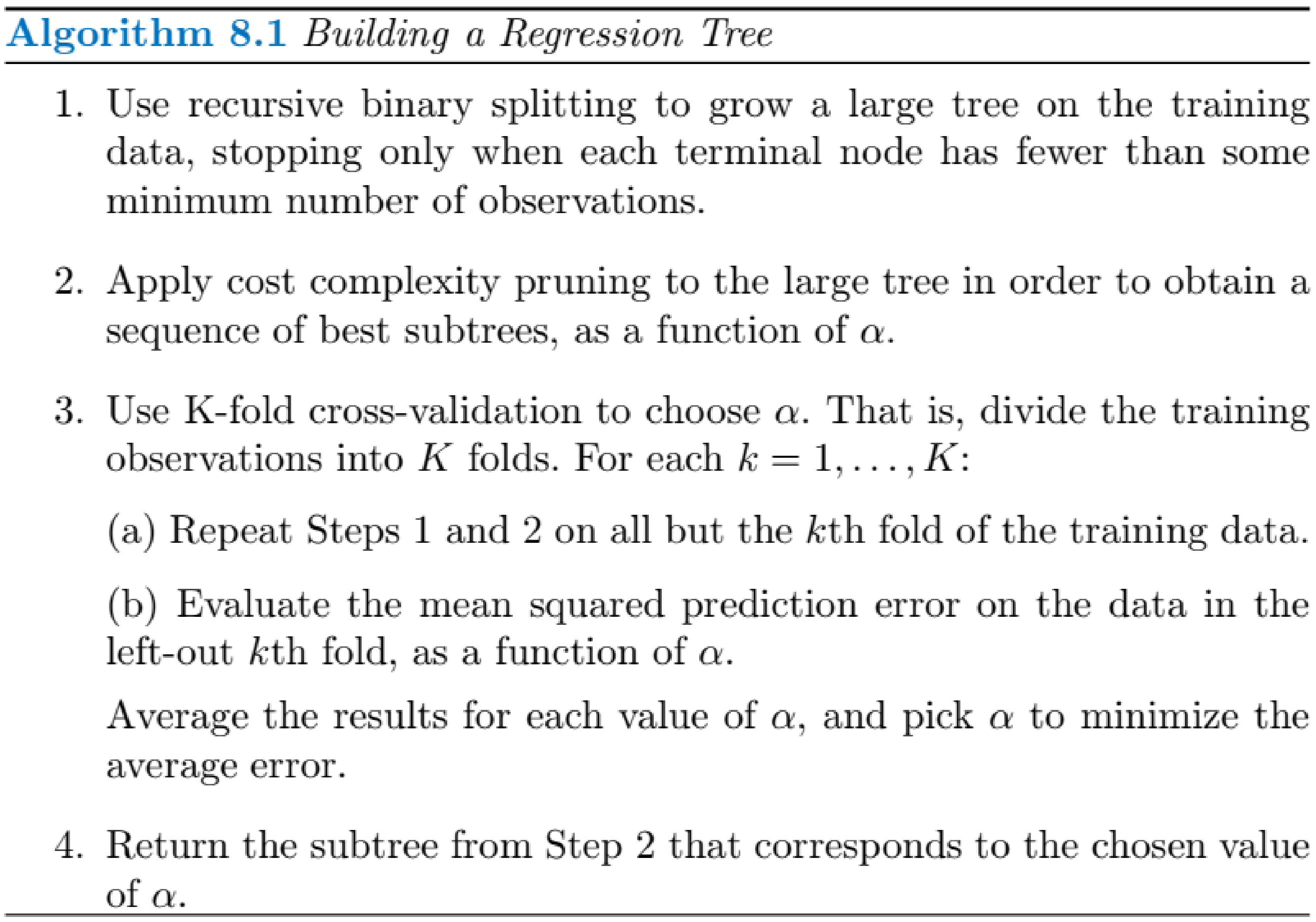
A better strategy is to grow a very large tree T_0, then *prune* it back to obtain a subtree that leads to the lowest test error.

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

Our problem: extremely large number of possible subtree. > cumbersome to use cross validation for all subtrees > too many

Therefore: need a way to select a small set of subtrees for consideration. Solution! **Cost complexity pruning** (weakest link pruning)

we consider a sequence of trees indexed by a nonnegative tuning parameter α .(instead of considering every possible subtree



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.

For each value of α there corresponds a subtree $T \subset T_0$ s.t

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

is as small as possible.

$|T|$: the number of **terminal nodes** of tree T.

R_m: the rectangle corresponding to the mth terminal node.

$\hat{y}_{R_m}$: the predicted response associated with R_m.

α: Tuning parameter, controls trade-off between the subtree's complexity and its fit to the training data.

$\alpha = 0 \rightarrow T$ is the same as T_0

α increase $\rightarrow$ smaller tree

It turns out that as α increase, branches get pruned from the tree in a nested and predictable fashion. $\rightarrow$ obtaining the whole sequence of subtrees as a function of α is easy.

$\rightarrow$ select α with validation set or by using CV.

$\rightarrow$ return to the full data set and obtain the subtree corresponding to α.

Example

fitting and pruning a regression tree on the Hitters data.

1. split Train/test data with a ratio of 1:1
2. Build large regression tree on train data and varied α to create subtrees with different numbers of terminal nodes.
3. perform 6-fold CV (train data 132 obs - multiple of 6) > estimate CVed MSE of trees(fn of alpha) three node tree> minimum CV/test error

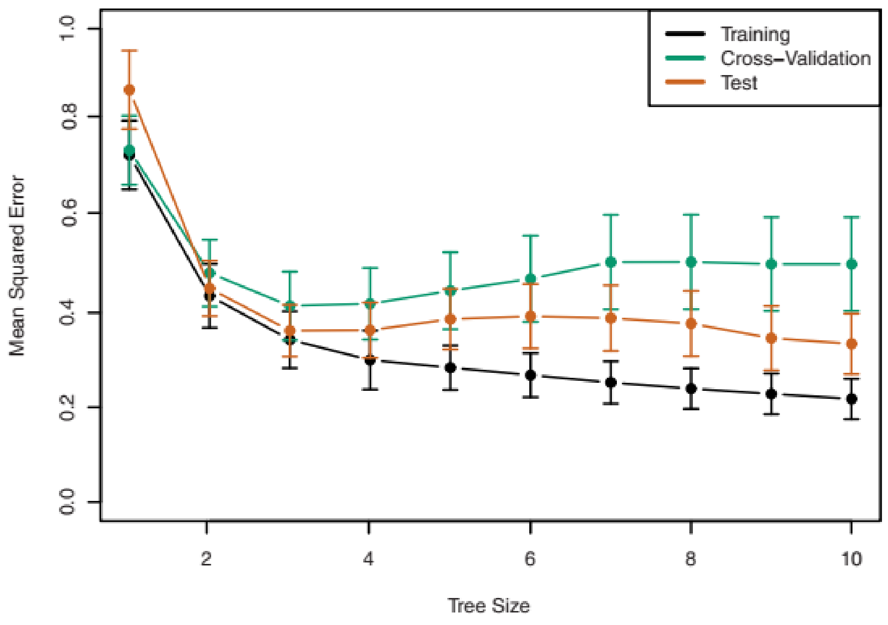


FIGURE 8.5. *Regression tree analysis for the* Hitters *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

8.1.2 Classification Trees

Classification tree: used to predict a qualitative response. (regression : quantitative)

we predict that each obs belongs to the **most commonly occurring class** of training obs in the region.

Also, *class proportions* among the training obs that fall into a region is a property of interest.

Just as in regression trees, we use **recursive binary splitting** to grow a classification tree.

Use**classification error rate** instead of RSS. when making binary splits classification error rate: fraction of the training obs that does not belong to the most common class.

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

$\hat{p}_{mk}$: the proportion of training obs in the mth region that are from the kth class.

However, in practice the classification error rate is not sensitive enough for tree growing.

Two other measures are preferable. (in practice)

Gini index:

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

A measure of total variance across the K classes.

If $\hat{p}_{mk}$ is close to 0 or 1 $\rightarrow$ Gini index takes on a small value.

(Therefore, the gini index is refereed to as a measure of node purity) small > node contains predominantly observations from a single class)

entropy:

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

Note: $0 \leq \hat{p}_{mk} \leq 1$ it follows that $0 \leq -\hat{p}_{mk} log \hat{p}_{mk}$ $(log \hat{p}_{mk} \leq 0)$

if the $\hat{p}_{mk}$ are all near zero or near one $\rightarrow$ entropy close to 0.

small when mth node is pure

Also Note: Gini index and entropy are similar numerically.

When building a classification tree, Gini index or the entropy is typically used because they are more sensitive to node purity.

But, use classification error rate when pruning the tree **if** prediction accuracy (of the final pruned tree) is the goal.

Example:

data: Heart data. Yes: heart disease, No: no heart disease

age, sex, chol etc.. 13 predictor variables.

CV > results in a tree with six terminal nodes

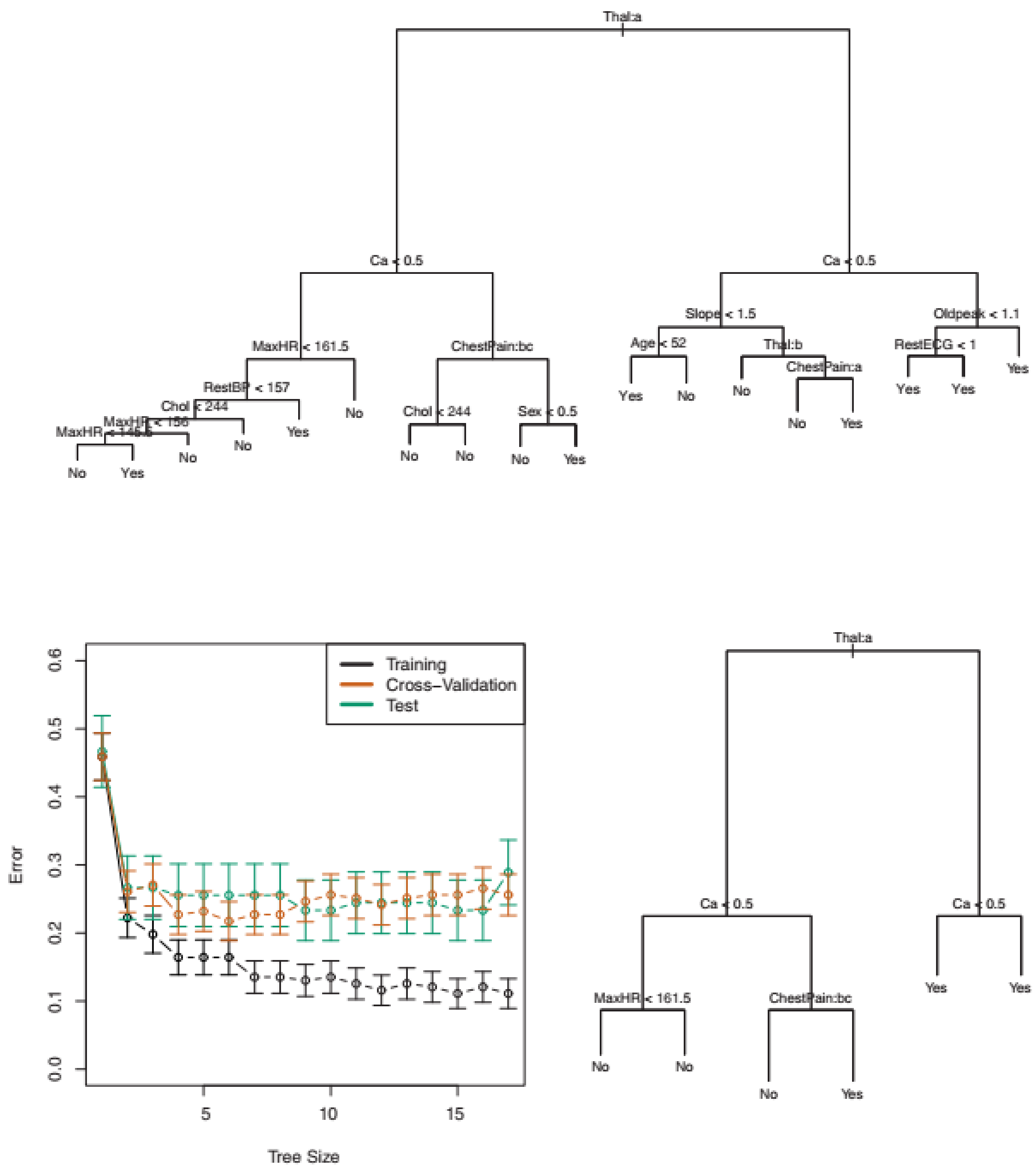


FIGURE 8.6. Heart *data. Top: The unpruned tree.* Bottom Left: *Cross -validation error, training, and test error, for different sizes of the pruned tree.* Bottom Right: *The pruned tree corresponding to the minimal cross-validation error.*

Some interesting Things:

1. Qualitative predictor variables can be used to construct decision trees as well. (Sex, Thal) A split on one of these var = assign a side (to one branch)to some qualities and remaining to the other side. Note: Chest pain bc $\rightarrow$ left side with 2nd and 3rd values of chest pain bc other to
2. right.

   Some of the split yields two terminal nodes with same predicted value. RestECG > all YES

   Why does this happen? it increases node purity. may not reduce classification error, but improves Gini index and entropy, ehich are more sensitive to node purity

Why is node purity important? lower node purity $\rightarrow$ more certainty about the predicted value.

8.1.3 Trees Versus Linear Models

Decision Trees have a very different flavor from the more classical approaches.

Decision Tree vs linear regression:

Which is better? depends on the problem.

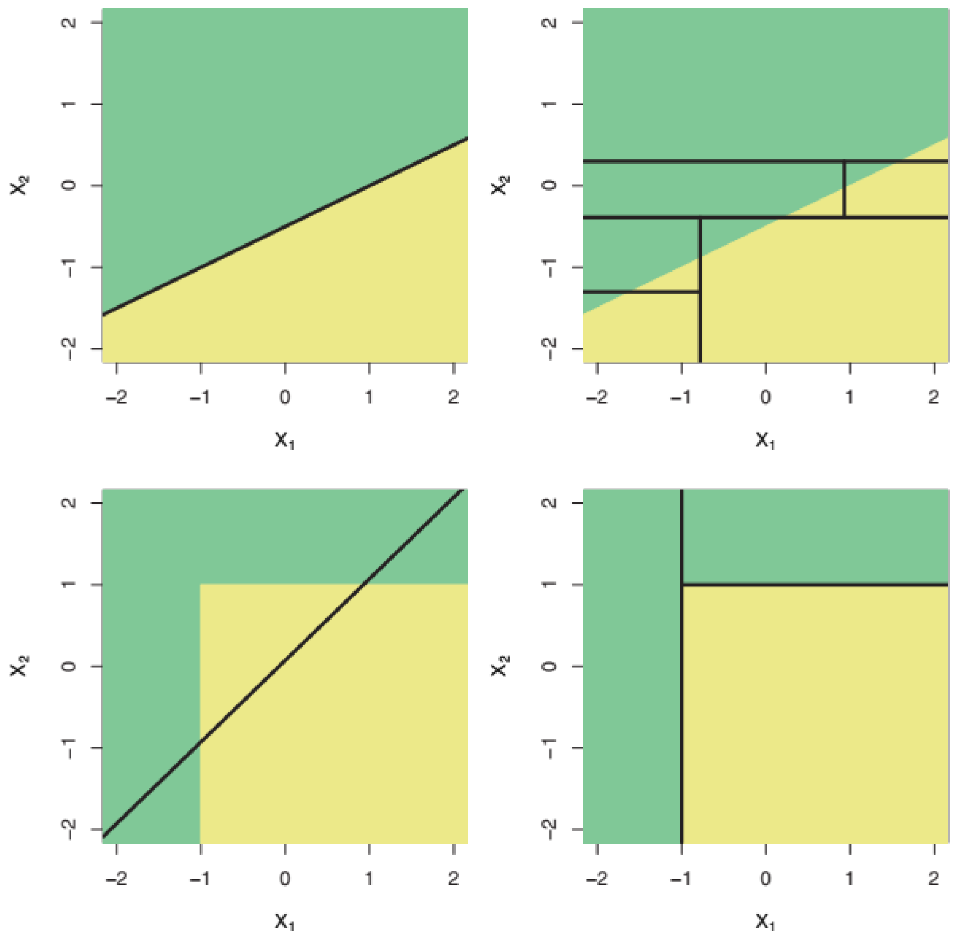


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: *Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).*

If the relationship between X and Y is approximately linear: Regression is better.

If the the relationship between X and Y is highly non-linear and complex: DT is better.

Sometimes, DT can be chosen for the sake of interpretability and visualization.

8.1.4 Advantages and Disadvantages of Trees

Adv:

1. Easy to explain.
2. Mirror human decision-making
3. Can be displayed graphically, (interpretability)
4. Handle categorical predictors without dummy variables

Dis-Adv:

1. Poor predictive accuracy (In general)
2. non-robust

The Dis-Advs can be overcome by using bagging, random forests, and boosting.

8.2 Bagging, Random Forests, Boosting

8.2.1 Bagging

Decision trees suffer from high variance.

e.g) if we split the training data into two parts at random and fit a decision tree to both halves, the results could be quite different.

Bootstrap aggregation or **bagging** is a general purpose procedure for reducing the variance of a statistical learning method.

It is particularly useful and frequently used in the context of decision trees.

Recall:

given a set of n independent obs $Z_1, Z_2, \ldots Z_n$. each with variance σ^2 .

$var(\bar{Z}) = \frac{\sigma^2}{n}$

Averaging a set of obs reduces variance.

Therefore, a natural way to reduce the variance and hence increase the prediction accuracy is to take **many training sets** from population, **build a separate training set** and **average the resulting prediction**.

In other words, calculate $\hat{y}^1(x), \hat{y}^2(x), \ldots, \hat{y}^B(x)$ using B separate training sets. Average them in order to a single low-variance statistical learning model.

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

In practice we do not have access to multiple training sets.

Instead we bootstrap:

1. take repeated samples from the single training data set. $\rightarrow$ generate B different bootstrapped training data sets. (중복가능)

2. train model on the bth bootstrapped training set to get $\hat{f}^{*b}(x)$.
3. average all the prediction to obtain:

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

This is called Bagging. It is particularly useful for decision trees.

In regression tree setting:

1. construct B regression trees with B bootstrapped training sets
2. average the resulting prediction

The trees are grown deep and not pruned. (High variance, low bias)

Averaging these B trees reduce variance. $\rightarrow$ improvements in accuracy

In Classification setting:

simplest method: in step 2. instead of averaging use 다수결.

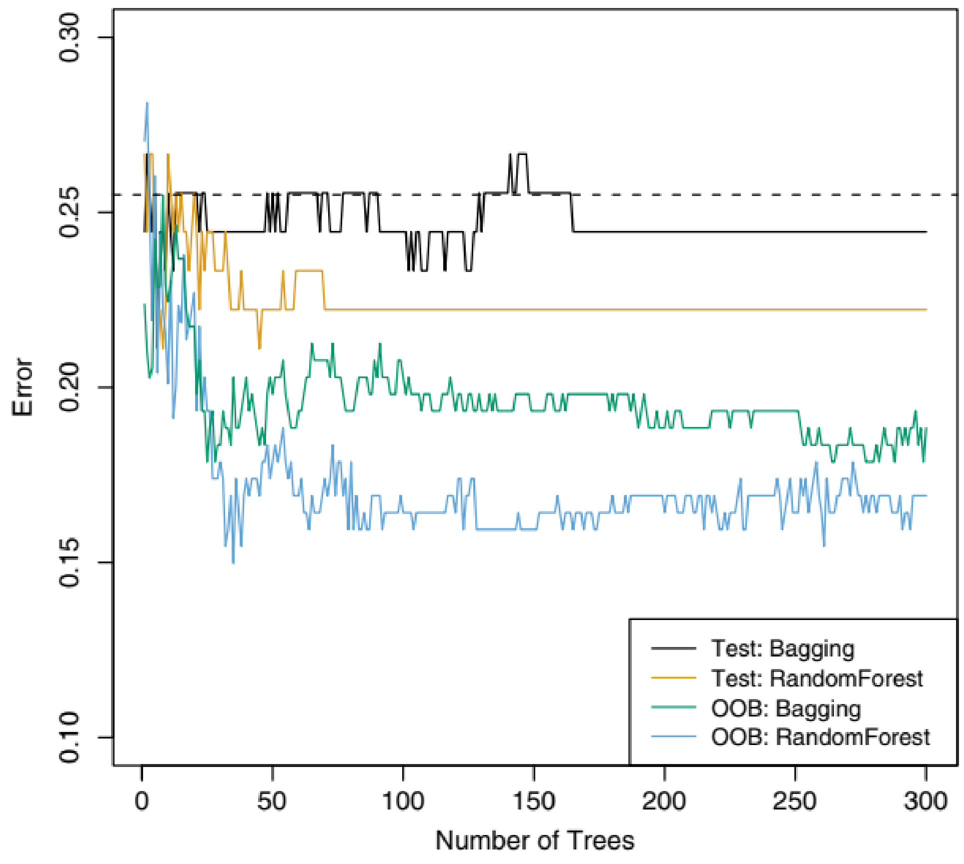


FIGURE 8.8. *Bagging and random forest results for the* `Heart` *data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.*

Example: the bagging test error is slightly lower.

Note: large B does not lead to overfitting.

Out-of-Bag Error Estimation

A straightforward way to estimate the test error of a bagged model. (without CV)

It can shown that on average: each bagged tree makes use of around two-thirds of the observations.

The remaining one third of the observations that are not used to fit a given bagged tree are refereed to as **out-of-bag**(OOB) observations.

1. use all tree to predict ith obs that is in OOB obs.
2. take average or take majority vote (regression or classification)
3. get OOB MSE (a valid estimate of the test error for bagged model, because the response for each obs is predicted using only the trees that were not fit using that obs.

Also, it can shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.

OOB approach for estimating test error is useful when data set is large and CV is not feasible.

Variable Importance Measures

When bagging is used it is difficult to interpret the resulting model.

bagging improves prediction accuracy at the expense of interpretability

One can obtain an overall summary of the importance of each predictor using RSS or the Gini index.

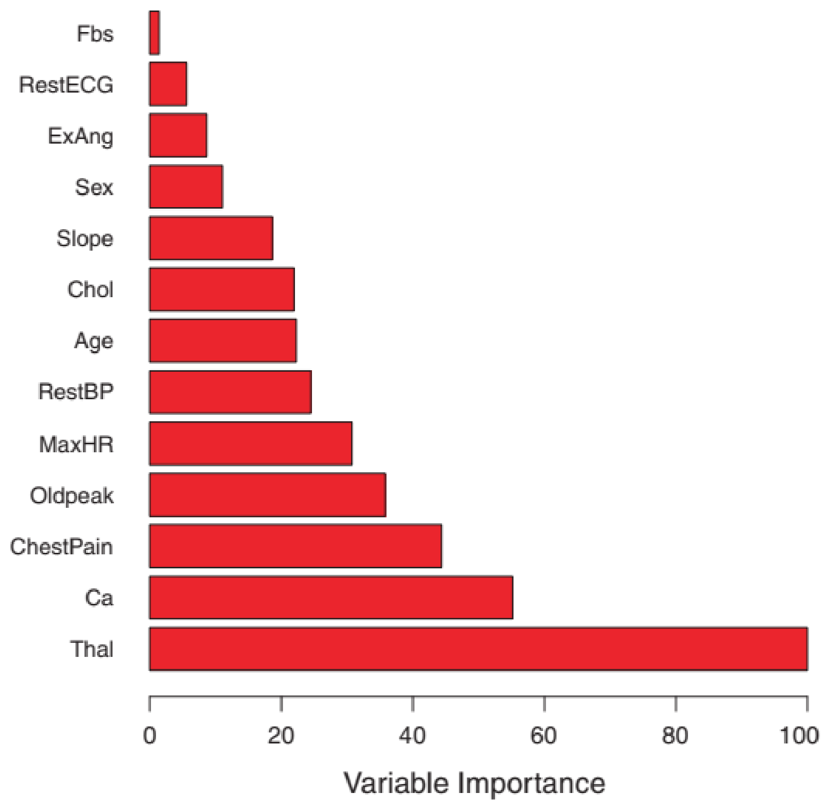


FIGURE 8.9. *A variable importance plot for the* Heart *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

A large value indicates an important predictor an important predictor.

(we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees)

8.2.2 Random Forests

Random forests is an improvement over bagged trees by a small tweak that decorrelates the trees.

1. Build a number of decision trees on bootstrapped training samples.
2. each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
3. A fresh sample of m predictors is taken at each split

Typically, we choose $m \approx \sqrt{p}$.

Why only allow a small number of predictors to split the tree?

If there is a very strong predictor in the data set, and other moderately strong predictors.

In bagging, most of the trees will use the very strong predictors in the top split.

Hence, all of the bagged trees will look similar. (Highly correlated)

>Thus similar shape > average cannot dramatically reduce variance

This is solved by forcing each split to consider only a small subset of predictors.

On average $(p - m)/p$ of the splits will not consider the strong predictor.

Other predictors will have more of a chance.

This is a process of 'decorrelating the trees.'

Thus, larger decrease in variance and the model become more reliable.

Example) high dimensional biological data set.

expression measurements of 4718 genes measured on tissue samples from 349 patients.

we wanted to use 500 gene expression with high variance to predict cancer type.

random forests is applied in this example.

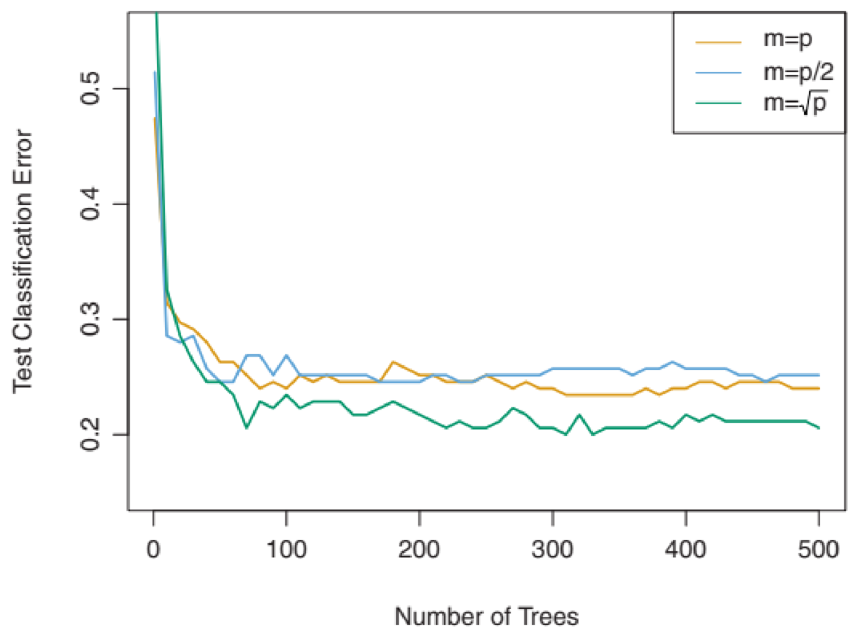


FIGURE 8.10. *Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node. Random forests $(m < p)$ lead to a slight improvement over bagging $(m = p)$. A single classification tree has an error rate of 45.7%.*

error rate of a single tree is 45.7%

null rate is 75.4% (simply classifying each obs to the dominant class overall)

using 400 trees is sufficient to give good performance.

the choice $m = \sqrt{p}$ gave a small improvement in test error over bagging (m = p) in this example.

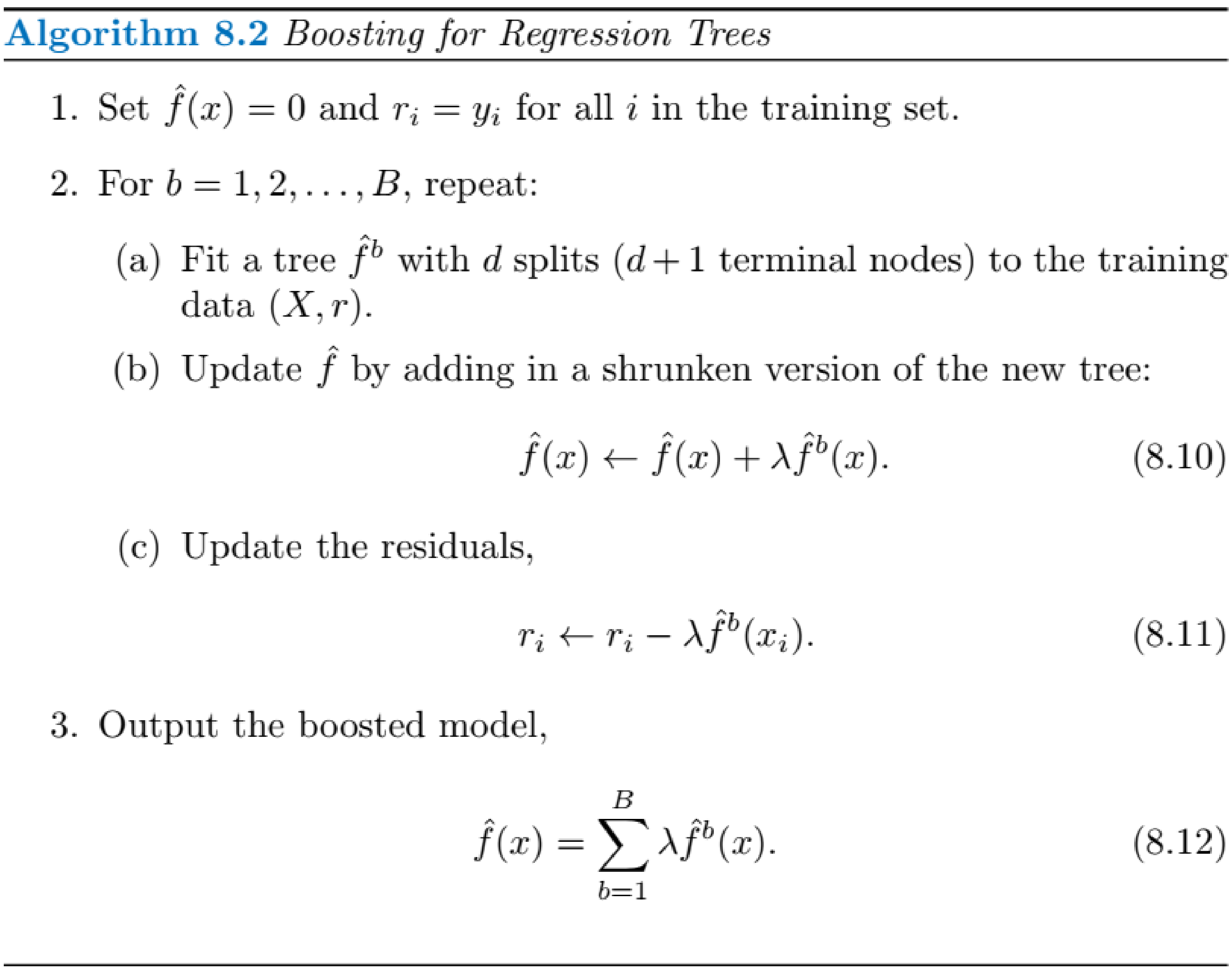
as with bagging, increased B does NOT cause overfitting, so we use sufficiently large B

8.2.3 Boosting

Another approach for improving the predictions resulting from a decision tree. (can be applied to many statistical learning methods)

In Boosting, the trees are grown sequentially: (tree grown using info from prev trees)

Also, Boosting does not involve bootstrap sampling.> instead, each tree is fit on a modified version of the origina data set.



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:
            $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$.        (8.10)
    (c) Update the residuals,
            $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$.        (8.11)
3. Output the boosted model,
            $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.        (8.12)
```

Tuning Parameters:

1. B: number of trees. (Caution: in boosting a large B can cause overfitting) > use CV to select B
2. λ : shrinkage parameter. Typically 0.01 or 0.001. > controls rate boosting learns

   very small λ can require a very large B to perform well.
3. d: number of splits in each tree. Controls the complexity of the boosted ensemble. Often d=1 works well. Also, d is the interaction depth.

Idea behind boosting:

fitting a single large decision tree to the data $\rightarrow$ potentially overfit

boosting learns slowly by fitting a decision tree to the residuals from prev models as Y (instead of outcome)

Note: in boosting the construction of each tree depends strongly on the trees that have already been grown.

*Using smaller trees : typically sufficient(even stump with d=1, + can aid in interpretability as well)

Boosting classification trees proceeds in a similar but slightly more complex way, details not included in ISL.

Example)

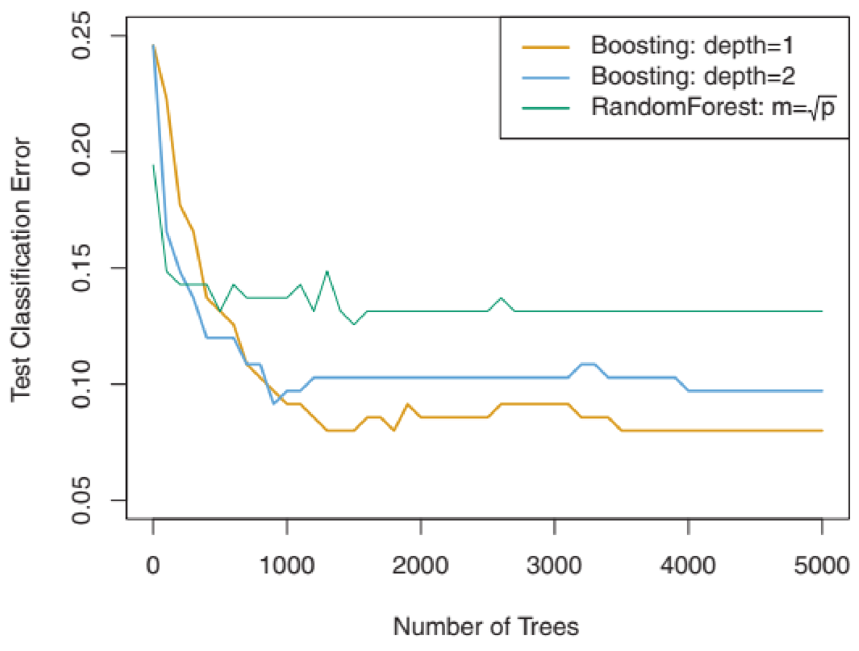


FIGURE 8.11. *Results from performing boosting and random forests on the 15-class gene expression data set in order to predict* cancer *versus* normal. *The test error is displayed as a function of the number of trees. For the two boosted models,* $\lambda = 0.01$. *Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant. The test error rate for a single tree is 24 %.*