STATISTICAL LEARNING

CHAPTER 8: TREE-BASED METHODS

INSTRUCTOR: SEOKHO LEE

HANKUK UNIVERSITY OF FOREIGN STUDIES

2015 SPRING

- We describe **tree-based** methods for regression and classification
  - These involve *stratifying* or *segmenting* the predictor space into a number of simple regions
  - For prediction, we typically use the mean or the mode of the training observations in the region to which it belongs
  - Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as **decision tree** methods
- Tree-based methods are simple and useful for interpretation
- They are typically not competitive with the best supervised learning approaches in terms of prediction accuracy
  - **Bagging**, **random forest**, and **boosting** involves producing multiple trees which are then combined to yield a single consensus prediction
  - Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss in interpretation

Regression Trees

- Motivating example: Hitters data
  - We predict a baseball players' Salary based on
    - Years (the number of years that he has plead in the major leagues)
    - Hits (the number of hits that he made in the previous year)
  - Remove missing values, log-transform Salary
  - Figure 8.1 shows a regression tree fit to this data
  - Figure 8.2 represents the regions which split predictor spaces by tree fit of Figure 8.1
  - The regions R_1, R_2, and R_3 in Figure 8.2 are known as **terminal nodes** or **leaves**
  - The points along the tree where the predictors space is split are referred to as **internal nodes**
    - In Figure 8.1, the two internal nodes are indicated by the text Years$<$4.5 and Hits$<$117.5
  - We refer to the segments of the trees that connect the nodes as **branches**

Regression Trees

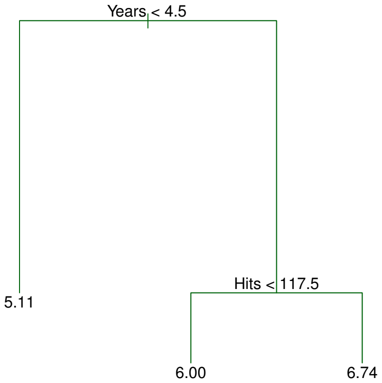

Figure 8.1: For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates 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$<$4.5, and the right-hand branch corresponds to Years$\geq$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.

Regression Trees

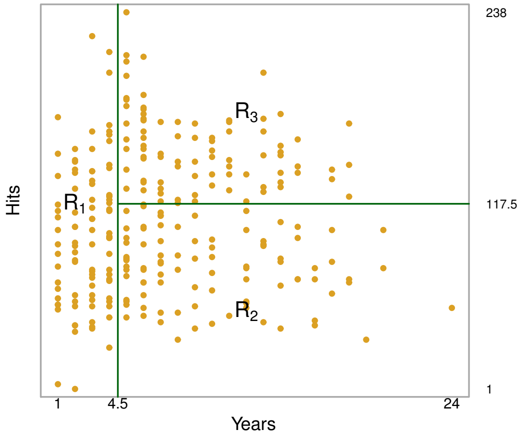

Figure 8.2: The three-region partition for the Hitters data set from the regression tree illustrated in Figure 8.1.

Prediction via Stratification of the Feature Space

- 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, $R_1, R_2, \ldots, R_J$
  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

- In Step 2 above, 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

Prediction via Stratification of the Feature Space

- The goal is to find boxes $R_1, R_2, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{8.1}$$

  - $\hat{y}_{R_j}$ 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**
    - This approach is **top-down** because it begins at the top of the tree (at which point all observations belong to a single region) 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

Prediction via Stratification of the Feature Space

- Recursive binary splitting
  - For any j and s, we define the pair of half-planes
$$R_1(j,s) = \{X|X_j < s\} \quad \text{and} \quad R_2(j,s) = \{X|X_j \geq s\} \tag{8.2}$$
  - We seek the value of j and s that minimize the equation
$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \tag{8.3}$$
    - Finding the values of j and s that minimize (8.3) can be done quite quickly, especially when the number of features p is not too large
  - Next, we repeat the process. In this time, instead of splitting the entire predictor space, we split one of the two previously identified regions so as to minimize RSS
  - The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations

Prediction via Stratification of the Feature Space

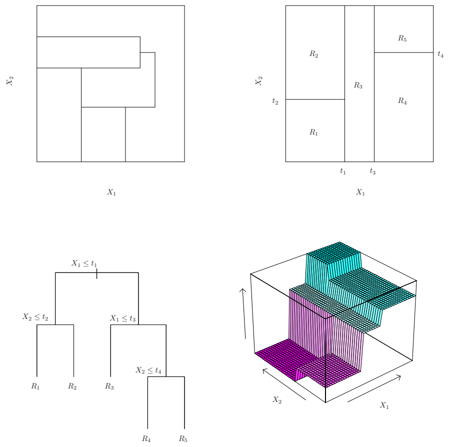

Figure 8.3: Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting. Top Right: The output of recursive binary splitting on a two-dimensional example. Bottom Left: A tree corresponding to the partition in the top right panel. Bottom right: A perspective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

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
  - 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
- One possible approach is to build the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold
  - It is short-sighted since a seemingly worthless split early on in the tree might be followed by a very good split
- A better strategy is **pruning**
  - First grow a very large tree T_0, and then **prune** it back in order to obtain a **subtree**
  - The best subtree can be selected using cross-validation or the validation set approach
  - However, considering every possible subtree would be too cumbersome

Tree Pruning

- **Cost complexity pruning** (also known as **weakest link pruning**)
  - 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

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T| \tag{8.4}$$

    is as small as possible
    - $|T|$ is the number of terminal nodes of the tree T
    - $\hat{y}_{R_m}$ is the mean of the training observation 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 $T = T_0$
    - As α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity (8.4) will tend to be minimized for a smaller subtree
  - (8.4) is reminiscent of the lasso (6.7) from Chapter 6

Tree Pruning

- **Cost complexity pruning** (also known as **weakest link pruning**)
    - As we increase α from 0 in (8.4), branches get pruned from the tree in a nested and predictable fashion
    - So obtaining the whole sequence of subtrees as a function of α is easy
    - We can select α using a validation set or using cross-validation. Then we return to the full data set and obtain the subtree corresponding to α
    - See Algorithm 8.1

Tree Pruning

```
**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 $\alpha$
3. Use $K$-fold cross-validation to choose $\alpha$. That is, divide the training
   observation into $K$ folds. For each $k = 1, \ldots, K$:
   (a) Repeat Steps 1 and 2 on all but the $k$th fold of the training data
   (b) Evaluate the mean squared prediction error on the data in the
       left-out $k$th fold, as a function of $\alpha$
   (c) Average the results for each value of $\alpha$, and pick $\alpha$ to minimize the
       average error
4. Return the subs tree from Step 2 that corresponds to the chosen value
   of $\alpha$
```

Tree Pruning

- Hitters data example
  - We build a regression tree using 9 of the features
  - We randomly divide the data set yielding 132 observations for the training set and 131 observations for the test set
  - For the training data, we obtain the pruned tree using 6-fold cross-validation
  - Figure 8.4 shows the unpruned regression tree using the training data
  - The pruned trees, according to the values of α, are applied to the test data to evaluate the test error (See Figure 8.5)

Tree Pruning

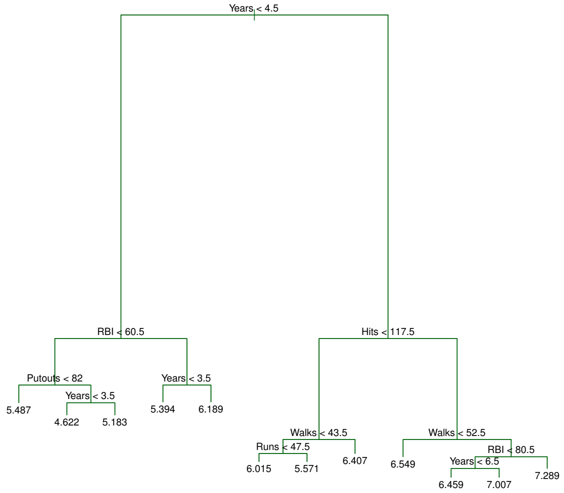

Figure 8.4: Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown.

Tree Pruning

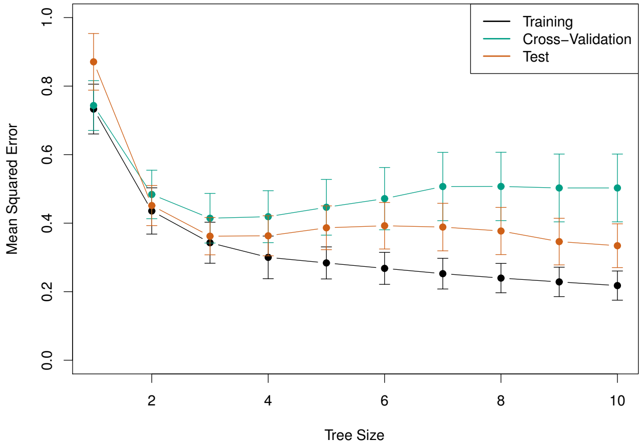

Figure 8.5: Regression tree analysis for the Hitters data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.

Classification Trees

- A **classification tree** is to predict a qualitative response rather than a quantitative one in the regression tree
    - In 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 a classification tree, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the **classification error rate**

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{8.5}$$

- $\hat{p}_{mk}$ represents the proportion of training observations in the *m*th region that are from the *k*th class
- It turns out that classification error is not sufficiently sensitive for tree-growing
- In practice two other measures are preferable

Classification Trees

- **Gini index**

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{8.6}$$

  - 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 0 or 1
  - The Gini index is referred to as a measure of node **purity**–a small value indicate that a node contains predominantly observations from a single class

- **Cross-entropy**

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk} \tag{8.7}$$

  - Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$
  - The cross-entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one
  - Like the Gini index, the cross-entropy will take on a small value if the mth node is pure

Classification Trees

- When building a classification tree, either the Gini index or the cross-entropy are typically used to evaluate the quality of a particular split rather than the classification error rate
- But the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal
- Heart data example (See Figure 8.6)
    - Response: the binary outcome HD (with or without chest pain)
    - 13 predictors, including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements
    - Cross-validation results in a tree with six terminal nodes

Classification Trees

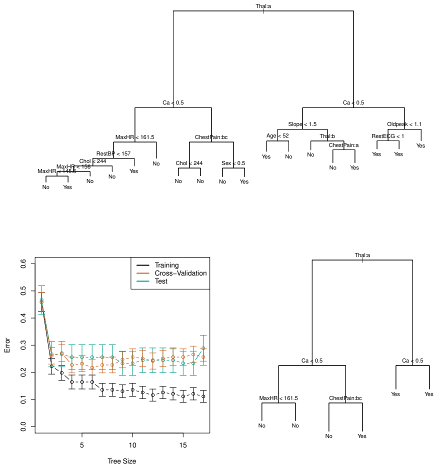

Figure 8.6: Heart data. Top: The unpruned tree. Bottom Left: Cross-validation, training, and test error, for different size of the pruned tree. Bottom Right: The pruned tree corresponding to the minimal cross-validation error.

Trees Versus Linear Models

- Linear regression assumes a model of the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j \tag{8.8}$$

- Regression trees assume a model of the form

$$f(X) = \sum_{m=1}^{M} c_m \cdot I(X \in R_m) \tag{8.9}$$

- Which model is better?
    - It depends on the problem at hand
    - See Figure 8.7 for classification

Trees Versus Linear Models

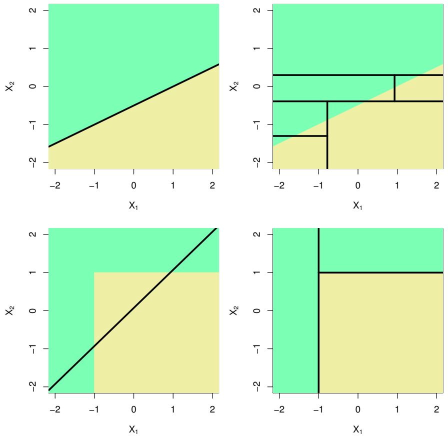

Figure 8.7: Top Row: A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right). Bottom Row: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

Advantages and Disadvantages of Trees

A. Trees are easy to explain to people. In fact, they are even easier to explain than linear regression!

A. Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.

A. Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small)

A. Trees can easily handle qualitative predictors without the need to create dummy variables

D. Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches see in this class

- The predictive performance of trees can be substantially improved by aggregating many decision trees, using methods like **bagging**, **random forests**, and **boosting**

Bagging

- **Bagging**, or **bootstrap aggregation**, is a general-purpose procedure for reducing the variance of a statistical learning method
  - It is well known that a natural way to reduce the variance and increase the prediction accuracy of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions
  - We could calculate $\hat{f}^1(x), \hat{f}^2(x), \ldots, \hat{f}^B(x)$ using B separate training sets, and average them

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

  - Of course, this is not practical because we generally do not have access to multiple training sets
  - Instead, we can bootstrap, by taking repeated samples from the (single) training data set, yielding

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

Bagging

- In bagging, these trees are grown deep, and are not pruned
  - Hence each individual tree has high variance, but low bias
  - Averaging these B trees reduces the variance
- Bagging procedure for classification problems
  - **Majority vote**: the overall prediction is the most commonly occurring class among the B predictions
- B can be very large. This will not lead to overfitting

Out-of-Bag Error Estimation

- Bagging can estimate the test error without the need to perform cross-validation or the validation set approach
- **Out-of-bag** observation
  - One can show that on average, each bagged tree makes use of around 2/3 of the observations
  - The remaining 1/3 observations not used to fit a given bagged tree are referred to as the **out-of-bag** (OOB) observations
  - OOB observations can be used to estimate the test error

Out-of-Bag Error Estimation

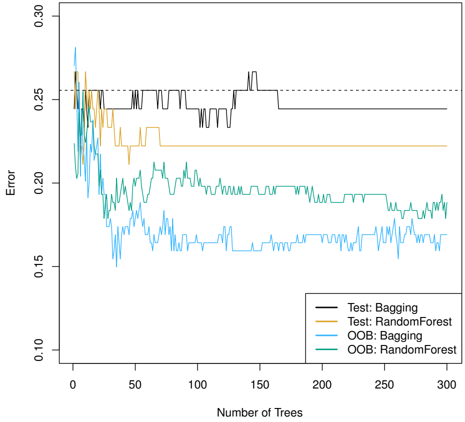

Figure 8.8: Bagging and random forest results for the Heart data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests war applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification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

Variable Importance Measures

- Bagging typically results in improved accuracy over prediction using a single tree. However, it is no longer possible to represent the resulting statistical learning procedure using a single tree
    - Bagging improves prediction accuracy at the expense of interpretability
- Instead, one can obtain an overall summary of the importance of each predictor using the RSS (regression) or the Gini index (classification)
    - In the case of bagging regression trees, we can record the total amount that the RSS (8.1) is decreased due to splits over a given predictor, averaged over all B trees
    - In the context of bagging classification trees, we can add up the total amount that the Gini index (8.6) is decreased by splits over a given predictor, averaged over all B trees
- Figure 8.9: a graphical representation of the **variable importances** in the Heart data

Variable Importance Measures

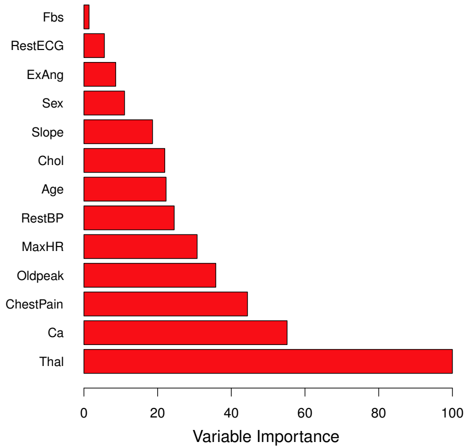

Figure 8.9: A variable importance plot for the Heart data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.

Random Forests

- **Random forests** provide an improvement over bagged trees by way of a small tweak that **decorrelates** the trees
  - When building bagged decision trees, 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
  - Typically we choose $m \approx \sqrt{p}$ (4 out of the 13 for the Heart data)

- Rationale for random forest
  - Suppose that there is one very strong predictor in the data set
  - Bagging
    - In the collection of bagged trees, most or all of the trees will use this strong predictor in the top split
    - Consequently, all of the trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated
    - Averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities
  - Random forest (See Figure 8.8)
    - On average $(p - m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance
    - We can think this process as **decorrelating** the trees, making the average of the resulting trees less variable and more reliable

Random Forests

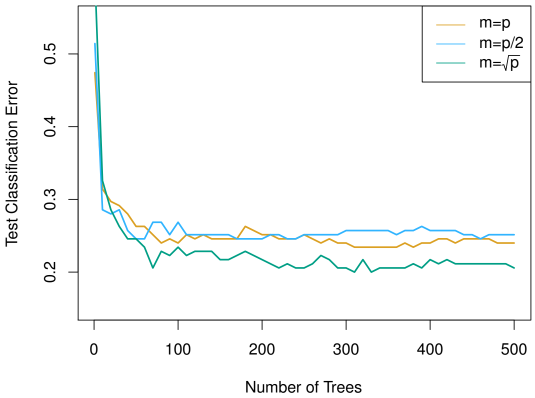

Figure 8.10: Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree 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%.

Bootsting

- **Boosting** is another approach for improving the predictions resulting from a decision tree
- Like bagging, boosting is a general approach that can be applied to many statistical learning methods
- 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 does not involve bootstrap sampling. Instead each tree is fit on a modified version of the original data set
  - Like bagging, boosting involves combining a large number of decision trees, $\hat{f}^1, \ldots, \hat{f}^B$
  - See Algorithm 8.2

Bootsting

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 verso 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)
```

Bootsting

- Ideas behind the boosting procedure
  - In bagging, fitting a single large decision tree to the data amounts to **fitting the data hard** and potentially overffiting
  - In boosting, given the current model, we fit a decision tree to the residuals from the model
  - Then we 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
  - By fitting small trees to the residuals, we slowly improve $\hat{f}$ in ares where it does not perform well
  - In general, statistical learning approaches that **learn slowly** tend to perform well
  - Not that in boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown

Bootsting

- 3 tuning parameter in boosting
  1. The number of trees B. Unlike bagging and random forests, boosting can over fit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B
  2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.0001, 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
  3. The number d of splits 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. In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable. 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

Boosting

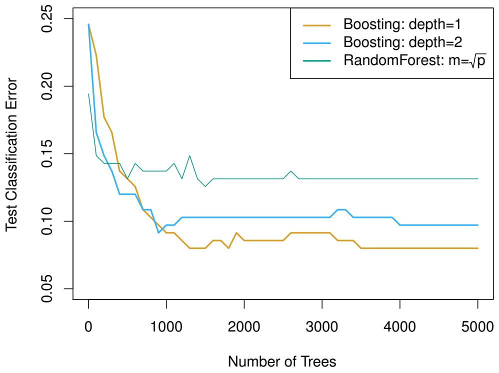

Figure 8.11: Results from performing boosting and random forests on the 15-class gene expression data se tin order to predict cancer versus normal. The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly out perform depth-2 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%.

Lab: Decision Trees