Lecture 9
Tree-Based Methods

CHUNG-MING KUAN

Department of Finance & CRETA

National Taiwan University

May 5, 2022

Lecture Outline

1. Introduction
2. Regression Trees
   - A Simple Tree
   - Recursive Binary Splitting
   - Tree Pruning
3. Classification Trees
   - Measures for Node Impurity
   - Tree Pruning
4. Bagging
5. Random Forest
6. Boosting

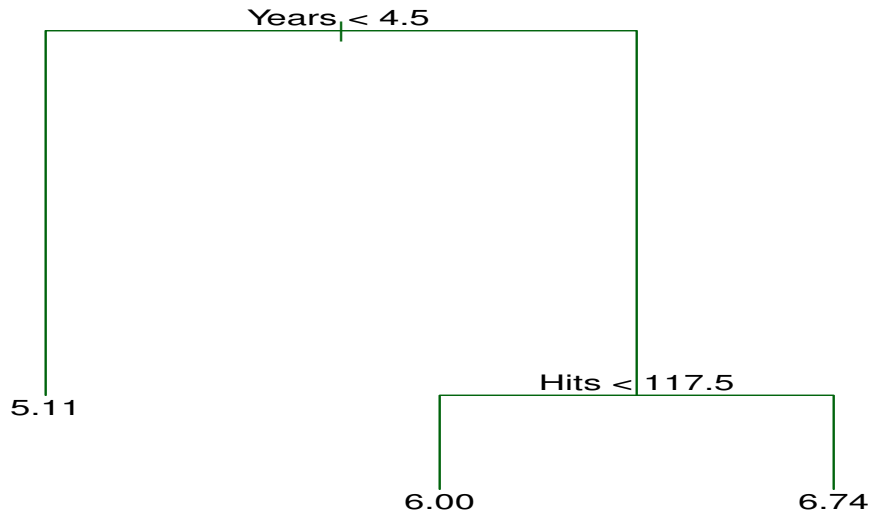
Introduction

Unlike conventional linear/nonlinear regressions, tree-based methods do not rely on models but use a set of rules to determine their predictions. In particular, these methods postulate a sequence of splitting rules that partition the sample space of predictors into non-overlapping regions and make predictions based only on the observations in each region.

- This process can be depicted as a "tree" growing "branches" and "leaves". Taking each split of a predictor as a decision, this process leads to a decision tree; for regression/classification problems, such trees are referred to as regression/classification trees.
- When growing a decision tree, every predictor is a candidate for splitting at each node, and each splitting is determined by choosing a predictor (from all predictors) and a cutpoint (from all possible values of that predictor) that yield the best fit of data.

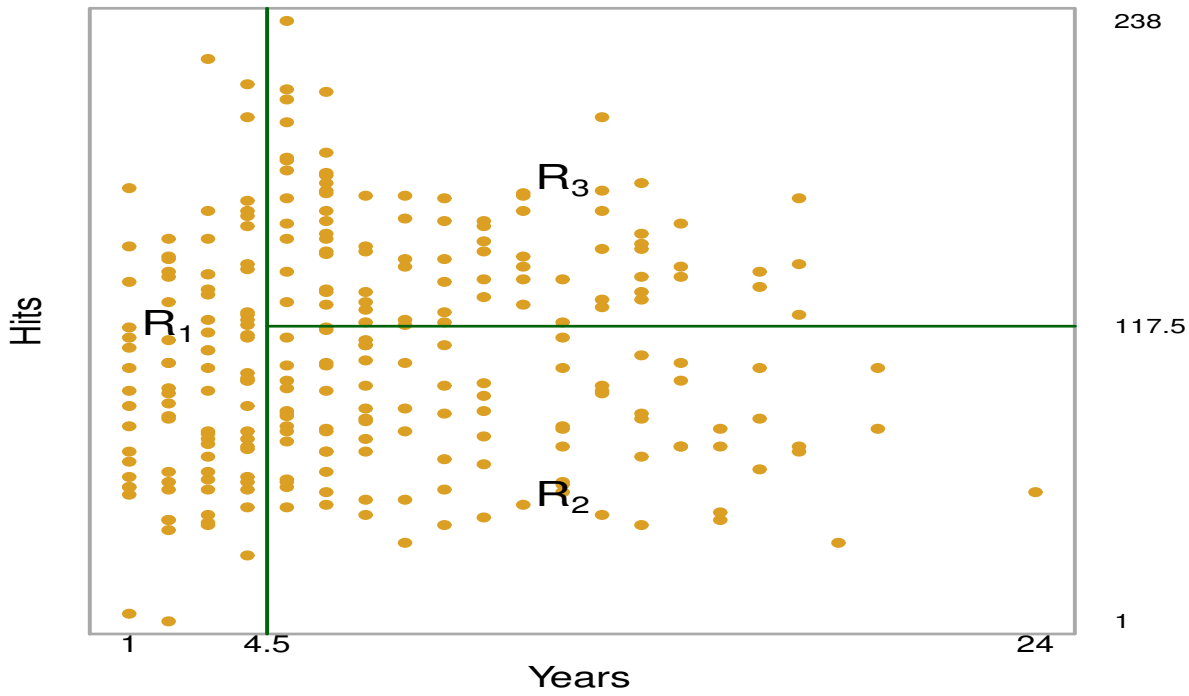
Example of a Simple Tree

Consider prediction of y (say, the salary of baseball players) based on two predictors: x_1 (Years of experience) and x_2 (number of Hits). Below is a tree with these two predictors and three terminal nodes (leaves). The predicted value for each leaf is the mean response of y_i whose associated (x_{i1}, x_{i2}) fall in that leaf.



Source: Figure 8.1 of JWHT (2021)

The decision tree above is obtained by partitioning the sample space $\boldsymbol{X}$ into 3 regions: R_1, R_2, and R_3 in two steps: (1) Splitting "Years" at the cutpoint 4.5 to partition $\boldsymbol{X}$ into R_1 and R_1^c, and (2) Splitting "Hits" at the cutpoint 117.5 to obtain another two regions R_2 and R_3 in R_1^c.



Source: Figure 8.2 of JWHT (2021)

Growing a Simple Tree

Suppose there are two predictors x_1 and x_2. First conduct a split based on x_1 at the cutpoint s_1. This amounts to partitioning the sample space, $\boldsymbol{X} = \{(x_{i1}, x_{i2}), i = 1, \ldots, n\}$, into two regions:

$$R_1(1; s_1) = \{(x_{i1}, x_{i2}) : x_{i1} < s_1\},$$
$$R_1^c(1; s_1) = \{(x_{i1}, x_{i2}) : x_{i1} \geq s_1\}.$$

Let $\hat{y}_{R_1}$ and $\hat{y}_{R_1^c}$ denote the respective mean responses of $R_1(1; s_1)$ and $R_1^c(1; s_1)$, i.e., the sample averages of those y_i in $R_1(1; s_1)$ and $R_1^c(1; s_1)$. Clearly, different cutpoints s_1 result in different partitions (regions) and hence different mean responses.

For each partition of the sample space, the residual sum of squares (RSS) is:

$$\text{RSS}_1(s_1) = \sum_{\{i:\,(x_{i1},x_{i2})\in R_1(1;s_1)\}} (y_i - \hat{y}_{R_1})^2 + \sum_{\{i:\,(x_{i1},x_{i2})\in R_1^c(1;s_1)\}} (y_i - \hat{y}_{R_1^c})^2.$$

In practice, s_1 is one of the grid points in the ranges of x_1. This RSS is a measure of data fitness based on the split at s_1. By varying s_1, we obtain a sequence of $\text{RSS}_1(s_1)$.

Similarly, Consider the split based on x_2 at the cutpoint s_2. The resulting regions in the sample space are:

$$R_1(2; s_2) = \{(x_{i1}, x_{i2}) : x_{i2} < s_2\},$$
$$R_1^c(2; s_2) = \{(x_{i1}, x_{i2}) : x_{i2} \geq s_2\},$$

with the respective mean responses: $\hat{y}_{R_1}$ and $\hat{y}_{R_1^c}$, the sample averages of those y_i in $R_1(2; s_2)$ and $R_1^c(2; s_2)$. The RSS of the split at s_2 is

$$\text{RSS}_1(s_2) = \sum_{\{i:\,(x_{i1},x_{i2})\in R_1(2;s_2)\}} (y_i - \hat{y}_{R_1})^2 + \sum_{\{i:\,(x_{i1},x_{i2})\in R_1^c(2;s_2)\}} (y_i - \hat{y}_{R_1^c})^2.$$

Here, s_2 is one of the grid points in the ranges of x_2.

Let s_1^* denote the cutpoint that minimizes $\text{RSS}_1(s_1)$ and s_2^* the cutpoint that minimizes $\text{RSS}_1(s_2)$. The minimum of $\text{RSS}_1(s_1^*)$ and $\text{RSS}_1(s_2^*)$ is thus the best fit of data; we then choose the corresponding s_1^* or s_2^* as the cutpoint for the first split.

In the example above, the predictor x_1 (Years) and the cutpoint $s_1^* = 4.5$ result in two regions: $R_1(1; 4.5)$ and $R_1^c(1; 4.5)$ which give the best fit of data. As a result, the split based on x_1 at 4.5 leads to better fit of data than any split based on x_2.

The second split:

1. Partition $R_1(1; 4.5)$ by splitting x_1 or x_2, while holding $R_1^c(1; 4.5)$ fixed.
2. Partition $R_1^c(1; 4.5)$ by splitting x_1 or x_2, while holding $R_1(1; 4.5)$ fixed.

For example, splitting x_1 in $R_1(1; 4.5)$ yields:

$$R_2(1; s_1) = \{(x_{i1}, x_{i2}) \in R_1(1; 4.5) : x_{i1} < s_1 < 4.5\},$$
$$R_2^c(1; s_1) = \{(x_{i1}, x_{i2}) \in R_1(1; 4.5) : 4.5 > x_{i1} \geq s_1\}.$$

The resulting RSS is:

$$\text{RSS}_2(s_1 = 4.5; s_1) = \sum_{\{i:\,(x_{i1},x_{i2})\in R_2(1;s_1)\}} (y_i - \hat{y}_{R_2})^2 + \sum_{\{i:\,(x_{i1},x_{i2})\in R_2^c(1;s_1)\}} (y_i - \hat{y}_{R_2^c})^2 + \sum_{\{i:\,(x_{i1},x_{i2})\in R_1^c(1;4.5)\}} (y_i - \hat{y}_{R_1^c})^2.$$

Splitting x_2 in $R_1(1; 4.5)$ yields

$$R_2(2; s_2) = \{(x_{i1}, x_{i2}) \in R_1(1; 4.5) : \ x_{i2} < s_2\},$$
$$R_2^c(2; s_2) = \{(x_{i1}, x_{i2}) \in R_1(1; 4.5) : \ x_{i2} \geq s_2\},$$

and the resulting RSS:

$$\text{RSS}_2(s_1 = 4.5; s_2) = \sum_{\{i: \, (x_{i1}, x_{i2}) \in R_2(2; s_2)\}} (y_i - \hat{y}_{R_2})^2 +$$
$$\sum_{\{i: \, (x_{i1}, x_{i2}) \in R_2^c(2; s_2)\}} (y_i - \hat{y}_{R_2^c})^2 + \sum_{\{i: \, (x_{i1}, x_{i2}) \in R_1^c(1; 4.5)\}} (y_i - \hat{y}_{R_1^c})^2.$$

Similarly, we partition $R_1^c(1; 4.5)$, while holding $R_1(1; 4.5)$ fixed. The resulting partitions are:

$$R_2(1; s_1) = \{(x_{i1}, x_{i2}) \in R_1^c(1; 4.5) : 4.5 \le x_{i1} < s_1\},$$
$$R_2^c(1; s_1) = \{(x_{i1}, x_{i2}) \in R_1^c(1; 4.5) : x_{i1} \ge s_1 > 4.5\},$$

or

$$R_2(2; s_2) = \{(x_{i1}, x_{i2}) \in R_1^c(1; 4.5) : x_{i2} < s_2\},$$
$$R_2^c(2; s_2) = \{(x_{i1}, x_{i2}) \in R_1^c(1; 4.5) : x_{i2} \ge s_2\}.$$

These two partitions yield another 2 $\text{RSS}_2(s_1 = 4.5; s_1)$ and $\text{RSS}_2(s_1 = 4.5; s_2)$. The minimum of these 4 RSS_2 gives the best fit of data, and the corresponding cutpoint (s_1^* or s_2^*) determines the second split.

In this example, the second split is based on x_2 (Hits) at $s_2 = 117.5$, leading to the regions R_2 and R_3 in Figure 8.2; in our notations,

$$R_2 = R_2(2; 117.5) = \{(x_{i1}, x_{i2}) \in R_1^c(1; 4.5) : x_{i2} < 117.5\},$$
$$R_3 = R_2^c(2; 117.5) = \{(x_{i1}, x_{i2}) \in R_1^c(1; 4.5) : x_{i2} \geq 117.5\}.$$

The resulting RSS is:

$$\text{RSS}(s_1 = 4.5; s_2 = 117.5) = \sum_{\{i:\,(x_{i1},x_{i2}) \in R_1(1;4.5)\}} (y_i - \hat{y}_{R_1})^2 + \sum_{\{i:\,(x_{i1},x_{i2}) \in R_2\}} (y_i - \hat{y}_{R_2})^2 + \sum_{\{i:\,(x_{i1},x_{i2}) \in R_3\}} (y_i - \hat{y}_{R_3})^2.$$

The predictions are the mean responses of these 3 regions: $\hat{y}_{R_1} = 5.11$, $\hat{y}_{R_2} = 6.0$, and $\hat{y}_{R_3} = 6.74$.

Recursive Binary Splitting

More generally, when there are p predictors $\boldsymbol{x} = (x_1, \ldots, x_p)$, the recursive binary splitting partitions the sample space $\boldsymbol{X}$ into distinct and non-overlapping hyper-cubes. Write $R_1(j; s) = \{\boldsymbol{x} \in \boldsymbol{X} : x_j < s\}$ and $R_1^c(j; s) = \{\boldsymbol{x} \in \boldsymbol{X} : x_j \geq s\}$ as hyper-cubes in the first step.

Step 1

For each $j = 1, \ldots, p$, find s that minimizes:

$$\text{RSS}_1(j; s) = \sum_{i:\, \boldsymbol{x}_i \in R_1(j;s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, \boldsymbol{x}_i \in R_1^c(j;s)} (y_i - \hat{y}_{R_1^c})^2,$$

where $\hat{y}_{R_1}$ and $\hat{y}_{R_1^c}$ are the respective mean responses of $R_1(j, s)$ and $R_1^c(j; s)$. The first split is based on x_{j^*} and s^* that minimize $\text{RSS}_1(j; s)$, $j = 1, \ldots, p$, with the partitions $R_1^* = R_1(j^*; s^*)$ and $(R_1^*)^c = R_1^c(j^*; s^*)$.

Step 2

For each $j = 1, \ldots, p$, while holding $(R_1^*)^c$ fixed, partition R_1^* into

$$R_2(j; s) = \{\boldsymbol{x} \in R_1^* : x_j < s\}, \quad R_2^c(j; s) = \{\boldsymbol{x} \in R_1^* : x_j \geq s\},$$

or, while holding R_1^* fixed, partition $(R_1^*)^c$ into

$$R_2(j; s) = \{\boldsymbol{x} \in (R_1^*)^c : x_j < s\}, \quad R_2^c(j; s) = \{\boldsymbol{x} \in (R_1^*)^c : x_j \geq s\}.$$

Note that we use $R_2(j; s)$ and $R_2^c(j; s)$ to denote partitions in R_1^* or in $(R_1^*)^c$. For $j = 1, \ldots, p$, we find s that minimizes:

$$\text{RSS}_2(j; s) = \sum_{i:\, \boldsymbol{x}_i \in R_2(j;s)} (y_i - \hat{y}_{R_2})^2 + \sum_{i:\, \boldsymbol{x}_i \in R_2^c(j;s)} (y_i - \hat{y}_{R_2^c})^2 + \sum_{i:\, \boldsymbol{x}_i \in (R_1^*)^c} (y_i - \hat{y}_{(R_1^*)^c})^2.$$

We also find s that minimizes:

$$\text{RSS}_2(j;s) = \sum_{i:\, \boldsymbol{x}_i \in R_2(j;s)} (y_i - \hat{y}_{R_2})^2 + \sum_{i:\, \boldsymbol{x}_i \in R_2^c(j;s)} (y_i - \hat{y}_{R_2^c})^2 + \sum_{i:\, \boldsymbol{x}_i \in R_1^*} (y_i - \hat{y}_{R_1^*})^2.$$

The desired split is based on x_{j^*} and s^* that lead to the smallest $\text{RSS}_2(j;s)$. Using similar notations as R_1^* and $(R_1^*)^c$, the partitioned hyper-cubes at this step are: $\{R_1^*,\ R_2^*,\ (R_2^*)^c\}$ or $\{(R_1^*)^c,\ R_2^*,\ (R_2^*)^c\}$.

Step 3

As in Step 2, we partition one of the 3 hyper-cubes: $\{R_1^*, R_2^*, (R_2^*)^c\}$ or $\{(R_1^*)^c, R_2^*, (R_2^*)^c\}$, and obtain 4 hyper-cubes that minimize RSS.

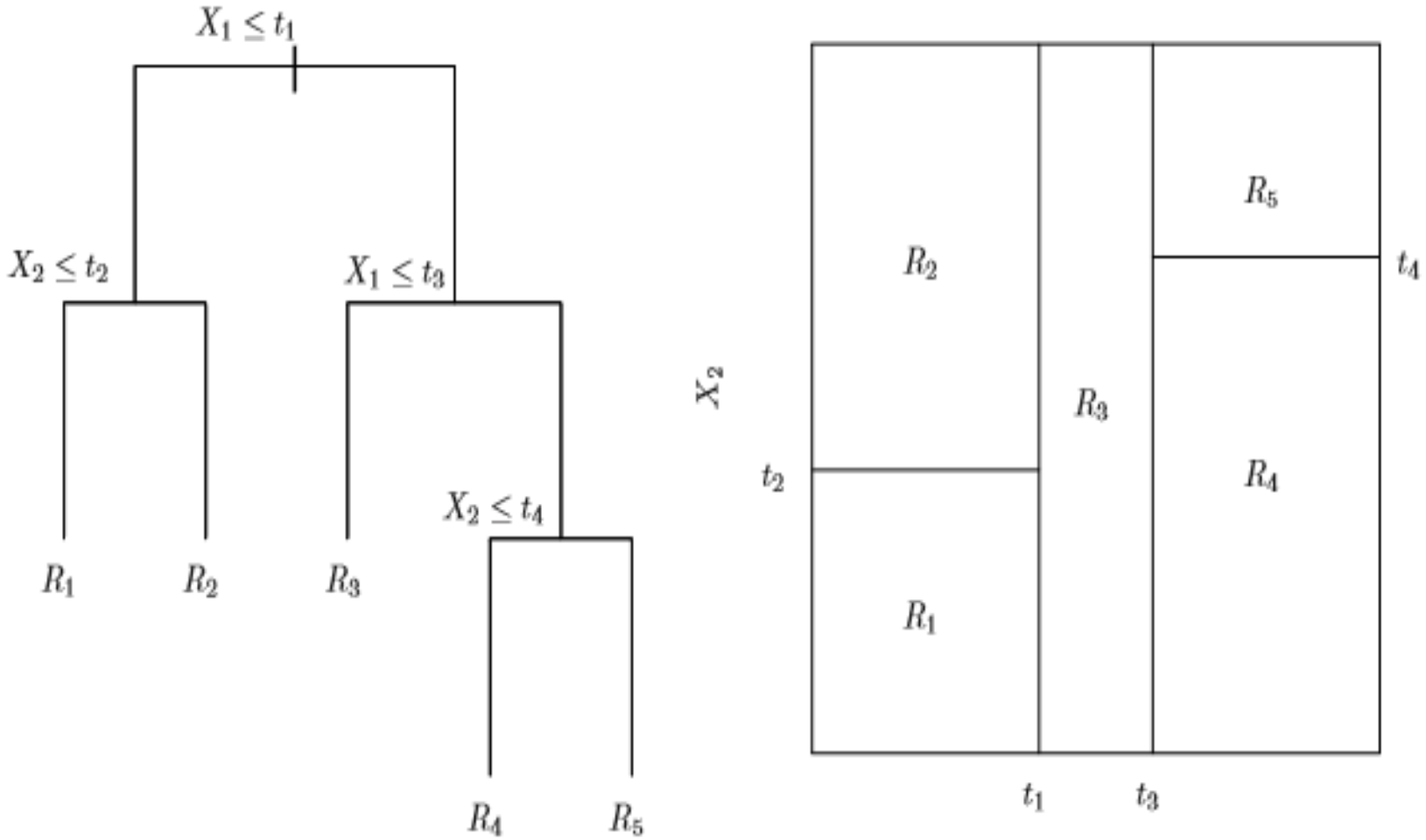
The tree-growing process above is repeated before it satisfies a stopping rule. A tree with J terminal nodes (leaves) minimizes:

$$\text{RSS} = \sum_{j=1}^{J} \sum_{i:\, \boldsymbol{x}_i \in R_j} \left(y_i - \hat{y}_{R_j} \right)^2 .$$

For $\boldsymbol{x}_i$ in R_j, the prediction is the mean response of that region: $\hat{y}_{R_j}$.

Stopping Rule: We stop this process when the reduction of RSS is less than a pre-specified threshold or when the number of observations in every terminal node is less than a given number. For the latter, the node already reaches the minimum size will not be considered for further splitting.

Below is a regression tree with 3 internal nodes and 5 leaves.



Source: Figure 8.3 of JWHT (2021)

- Regression trees are capable of representing complex (nonlinear) relations between the variable of interest y and the predictors $x_1, \ldots, x_p$. They have the advantage that their results are easy to interpret and understand, even for non-professionals.
- The process described above is a top-down, greedy approach for splitting. It is "top-down" since it begins from splitting the entire sample space; it is "greedy" because we consider the best split at each step, rather than adopting a split that may not be the best at this step but will lead to a better tree in a later step.
- Drawbacks
  - Regression tree is not robust, in the sense that a slight change of training data may lead to a very different tree.
  - Regression tree is computationally demanding when many predictors are involved or when a large tree with many leaves is to be grown.

Cost Complexity Pruning

A large tree may over fit training data and perform poorly in testing data. To ensure a tree with proper size, we may first grow a large tree T_0 and then prune it upwards (from the last split) to find a proper sub-tree $T \subset T_0$ by removing one or some internal (non-terminal) nodes. To this end, We search for a sub-tree $T(\alpha)$ that minimizes the following penalized RSS:

$$\sum_{j=1}^{|T|} \sum_{i:\, \boldsymbol{x}_i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha |T|,$$

where $|T|$ is the number of terminal nodes of T, R_j is the hyper-cube corresponding to the terminal node j, and α is the tuning parameter for the trade-off between tree complexity ($|T|$) and data fitness (RSS). This is known as cost complexity pruning.

k-Fold Cross Validation for α

- Divide the sample into k sub-samples. For each $j = 1, \ldots, k$, take the j^{th} sub-sample as the test sample and the remaining data as the training sample.
- Given α, we grow a tree for each training sample and prune to obtain a smaller tree $T_j(\alpha)$. We then evaluate $T_j(\alpha)$ using the j^{th} test sample. This leads to k mean squared prediction errors (MSPEs) $\text{MSPE}_j(\alpha)$, $j = 1, \ldots, k$. The desired α, α^*, minimizes the cross validated MSPE:

$$\text{MSPE}_{\text{CV}}(\alpha) = \frac{1}{k} \sum_{j=1}^{k} \text{MSPE}_j(\alpha).$$

- The final tree is the sub-tree $T(\alpha^*)$, pruned from the large tree T_0 with the tuning parameter α^*, where T_0 is obtained using the whole sample. .

Comparison between Regression and Tree

In analogy with the linear regression model $f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j$, a regression tree can be expressed as the regression model with the indicator function $\mathbf{1}(\boldsymbol{x} \in R_j)$ as its regressors:

$$f(\boldsymbol{x}) = \sum_{j=1}^{J} c_j \mathbf{1}(\boldsymbol{x} \in R_j).$$

Yet, $f(\boldsymbol{x})$ above cannot be estimated by OLS because the hyper-cubes R_j are unknown. Searching among all possible hyper-cubes to find the best possible hyper-cubes R_j and the coefficients c_j is computationally formidable. The tree growing process can be viewed as a computationally feasible (stepwise) method for determining R_j and c_j.

Classification Trees

- A classification tree is a decision tree for predicting the qualitative variable that labels two or more classes (categories).
- Just as regression trees, we use recursive binary splitting to construct classification trees. Yet, classification problems are concerned with correct prediction of an observation belonging to a class, rather than minimization of RSS. As such, different measures are needed for binary splitting in classification trees.
- Ideally, we wish to have a classification tree such that most (or all) observations in a terminal node belongs to one class. That is, we want to make each node as "pure" as possible (variation in each node as small as possible). To this end, we consider the node "impurity" measures discussed below.

Measures for Node Impurity

Suppose that y_i take the values $1, 2, \ldots, M$, corresponding to M different classes. When there are N_j observations in the hyper-cube R_j, define

$$\hat{p}_{jm} = \frac{1}{N_j} \sum_{i:\, \boldsymbol{x}_i \in R_j} \mathbf{1}(y_i = m),$$

the sample proportion of training data in R_j that belong to the m th class. Thus, $\sum_{m=1}^{M} \hat{p}_{jm} = 1$. R_j would be "pure" if one of $\hat{p}_{jm}$ is close to one. The class with the largest sample proportion is the most commonly occurring class in R_j:

$$m_j^* = \arg\max_m \hat{p}_{jm};$$

the prediction of R_j, $\hat{y}_{R_j}$, is the most commonly occurring class m_j^*.

- Classification error rate:

$$\frac{1}{N_j} \sum_{i:\, \boldsymbol{x}_i \in R_j} \mathbf{1}(y_i \neq m_j^*) = 1 - \hat{p}_{jm_j^*},$$

  the sample proportion of the observations not belonging to m_j^*.
- Gini Index (the variance of the j th node):

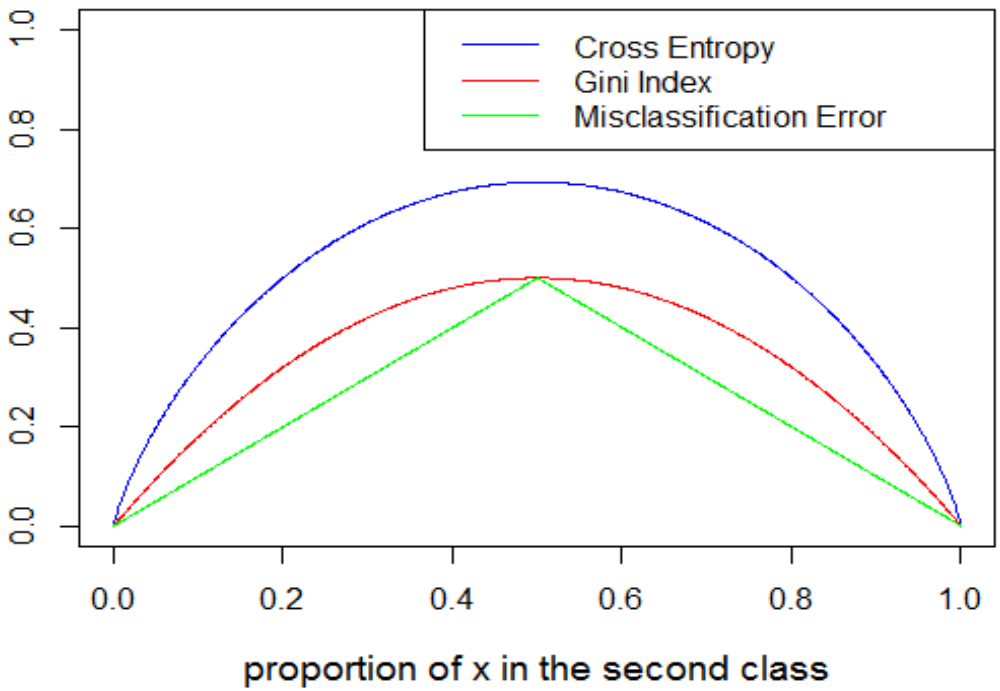
$$\sum_{m=1}^{M} \hat{p}_{jm}(1 - \hat{p}_{jm}),$$

  which would be small when all but one $\hat{p}_{jm}$ are near 0, i.e., R_j is "pure".
- Cross Entropy:

$$-\sum_{m=1}^{M} \hat{p}_{jm} \log(\hat{p}_{jm}),$$

  which would be small when all but one $\hat{p}_{jm}$ are near 0.

For the case $M = 2$, the node impurity measures are: $1 - \max(\hat{p}, 1 - \hat{p})$, $2\hat{p}(1 - \hat{p})$, and $-\hat{p}\log\hat{p} - (1 - \hat{p})\log(1 - \hat{p})$, where $\hat{p}$ is the sample proportion of a class. These measures reach maxima at $\hat{p} = 1/2$ (both classes have the same sample proportion), and they would be smaller when $\hat{p}$ is closer to zero or one.



Recursive Binary Splitting

We conduct recursive binary splitting as before. At each node, we find the cutpoint that minimizes the total node impurity measures below:

$$\sum_{j=1}^{J} \sum_{i:\, \boldsymbol{x}_i \in R_j} (1 - \hat{p}_{jm_j^*}) = \sum_{j=1}^{J} N_j (1 - \hat{p}_{jm_j^*}),$$

$$\sum_{j=1}^{J} \sum_{i:\, \boldsymbol{x}_i \in R_j} \sum_{m=1}^{M} \hat{p}_{jm}(1 - \hat{p}_{jm}) = \sum_{j=1}^{J} N_j \left(\sum_{m=1}^{M} \hat{p}_{jm}(1 - \hat{p}_{jm}) \right),$$

$$-\sum_{j=1}^{J} \sum_{i:\, \boldsymbol{x}_i \in R_j} \sum_{m=1}^{M} \hat{p}_{jm} \log(\hat{p}_{jm}) = -\sum_{j=1}^{J} N_j \left(\sum_{m=1}^{M} \hat{p}_{jm} \log(\hat{p}_{jm}) \right),$$

where J is the number of leaves.

An Example of Splitting

Consider a 2-class classification problem for a sample of 800 observations, with 400 in each class. A node with a and b observations belonging to, respectively, the first and second classes is denoted as $R_j(a, b)$. Suppose now a split yields the nodes $R_1(300, 100)$ and $R_2(100, 300)$. We have: $N_1 = N_2 = 400$, $m(1) =$ first class, $m(2) =$ second class,

$$\hat{p}_{1,m(1)} = \frac{1}{N_1} \sum_{i:\, \boldsymbol{x}_i \in R_1} \mathbf{1}(y_i = m(1)) = \frac{3}{4}, \quad \hat{p}_{1,m(2)} = \frac{1}{4},$$

and similarly, $\hat{p}_{2,m(1)} = 1/4$, $\hat{p}_{2,m(2)} = 3/4$. For this split, the total classification error and Gini index are, respectively,

$$400 \times (1 - 3/4) + 400 \times (1 - 3/4) = 200,$$
$$400 \times 2(3/4)(1/4) + 400 \times 2(3/4)(1/4) = 300.$$

Consider a different split that yields $R_1(200, 0)$ and $R_2(200, 400)$. We then have: $N_1 = 200$, $N_2 = 600$, and

$$\hat{p}_{1,m(1)} = \frac{1}{N_1} \sum_{i:\, \boldsymbol{x}_i \in R_1} \mathbf{1}(y_i = m(1)) = 1, \quad \hat{p}_{1,m(2)} = 0,$$

$$\hat{p}_{2,m(1)} = \frac{1}{N_2} \sum_{i:\, \boldsymbol{x}_i \in R_2} \mathbf{1}(y_i = m(1)) = \frac{1}{3}, \quad \hat{p}_{2,m(2)} = \frac{2}{3}.$$

For this split, the total classification error and Gini index are, respectively,

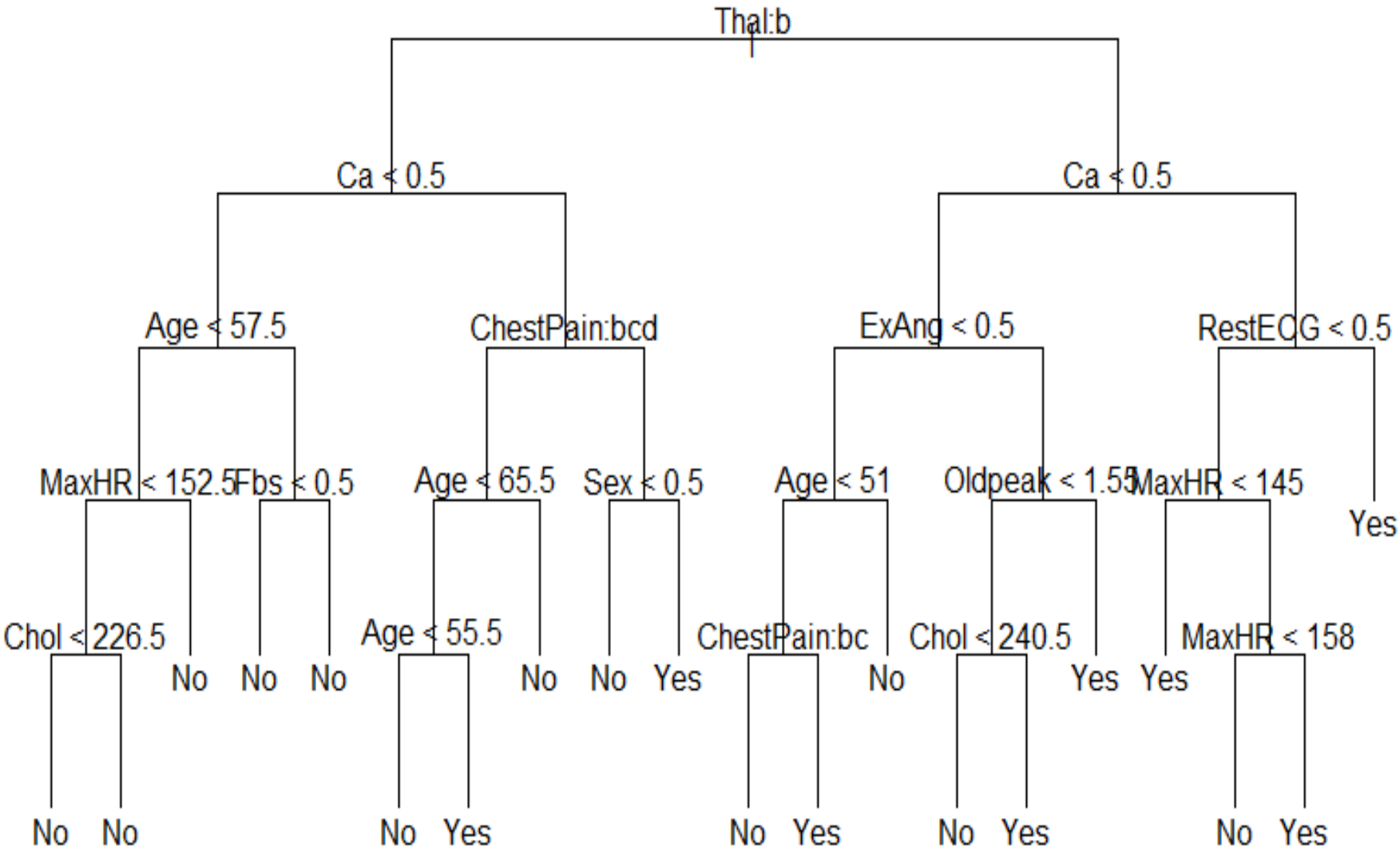
$$200 \times (1 - 1) + 600 \times (1 - 2/3) = 200,$$
$$200 \times 0 + 600 \times 2(1/3)(2/3) = 266.667.$$

Note that, while the classification error fails to distinguish these two splits, the Gini index chooses the second split.

Example (Heart Disease) from Sec. 8.1.2 JWHT (2021)

We want to predict `AHD` (if an individual has heart disease) based on 13 predictors. These include the following qualitative predictors: `Sex`, `Thal` (Thallium stress test), `ChestPain`, and quantitative predictors: `Age`, `Chol` (cholesterol measurement), `MaxHR` (maximum of heart rate), etc. The unpruned classification tree below is grown using recursive binary splitting based on the Gini Index.

Note: As can be seen in the next page, a split may yield two terminal nodes with the same predicted values. Such splits do not make a difference in prediction but improve node purity. The prediction of a purer hyper-cube usually has better accuracy.



Note that this tree is computed using the data of JWHT (2021) but different from that in p. 337 of JWHT (2021).

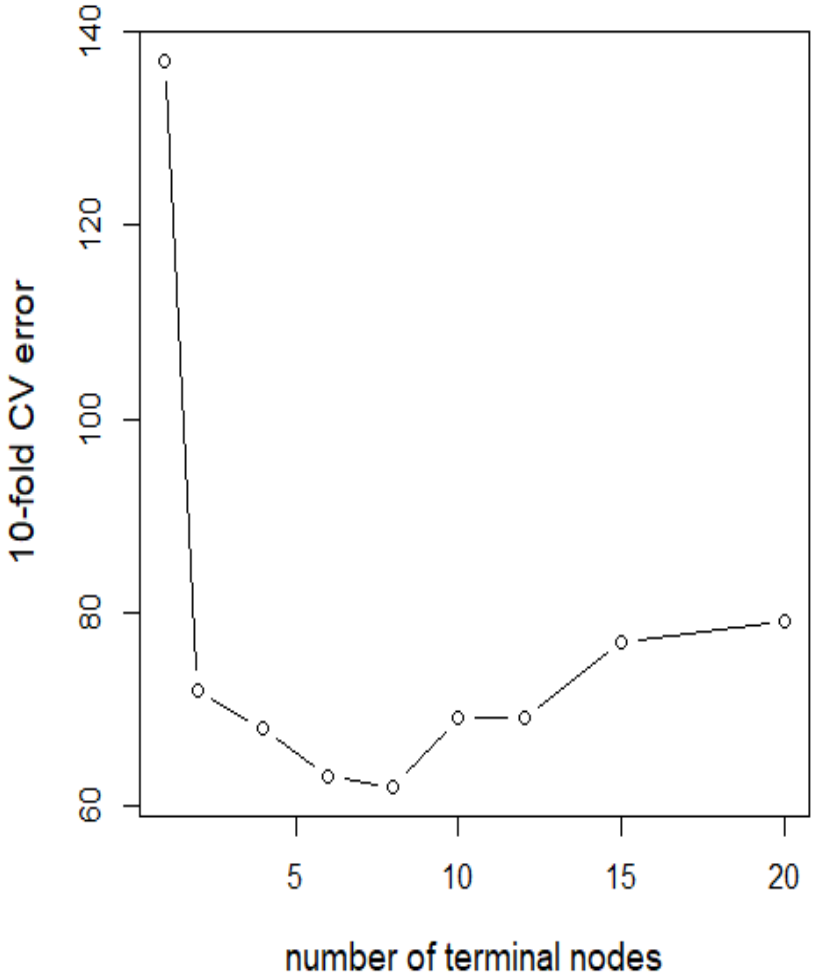
Tree Pruning

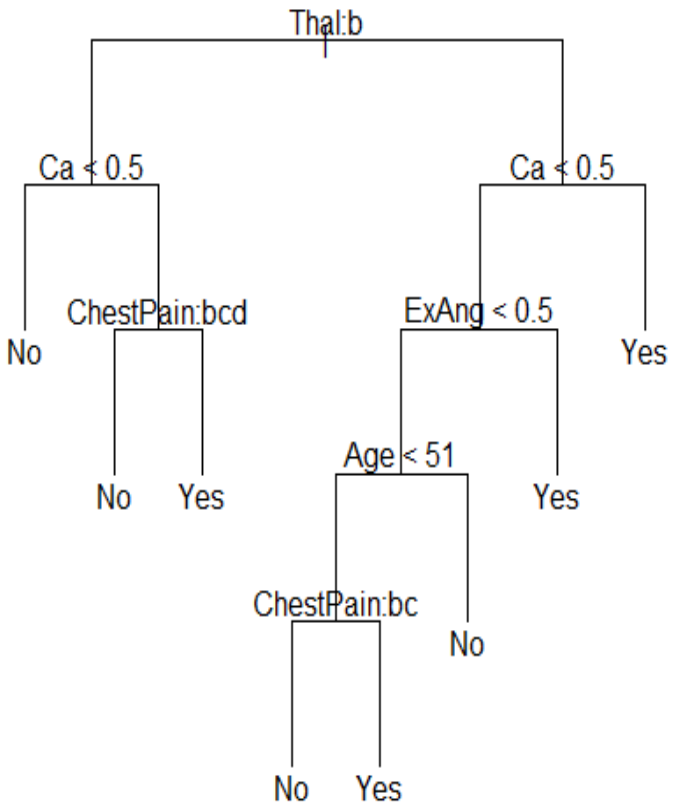
The idea of pruning a classification tree is the same as that of pruning a regression tree. Specifically, for a large grown tree T_0, we search for a sub-tree $T \subset T_0$ that minimizes

$$\text{The total measure of impurity} + \alpha|T|,$$

where the measure of impurity may be the classification error rate, Gini index, or cross entropy. The classification error rate is more commonly used for pruning a classification tree. We can determine a proper α using k-fold CV, based on the criterion of the total classification error, instead of MSPE, of test data.

Below is the 10-fold CV classification error rate for different numbers of terminal nodes (left) and the tree with 8 terminal nodes (right).





.

Bagging

Idea: It is well known that averaging helps reduce variance. As the prediction of a tree typically has a large variance, an effective approach to reducing its variance is bagging (bootstrap aggregating). This method constructs many trees from bootstrapped samples and averages their resulting predictions.

- Bootstrap the training sample, $\mathcal{S} = \{(y_i, x_{i1}, \ldots, x_{ip}), i = 1, \ldots, n\}$, B times to obtain B bootstrapped samples:

$$\mathcal{S}^*(b) = \{(y_i^*(b), x_{i1}^*(b), \ldots, x_{ip}^*(b)), i = 1, \ldots, n\}, \quad b = 1, \ldots, B.$$

- Grow a large tree (without pruning) from each $\mathcal{S}^*(b)$ and denote the prediction of the resulting tree based on the predictor $\boldsymbol{x}$ as $\hat{f}^*(\boldsymbol{x}; b)$, $b = 1, \ldots, B$.

- Bagged prediction
  - Regression trees: Bagged prediction is the average of B bootstrapped predictions:

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

    A large B in bagging improves prediction accuracy and does not cause over-fitting.
  - Classification trees: $\hat{f}^*(\boldsymbol{x}; b)$ is the indicator of some class j, $j = 1, \ldots, M$. Instead of using average of predictions, bagging of classification trees uses majority vote. That is, the prediction based on $\boldsymbol{x}$ is determined by the majority of B predicted classes from the bootstrapped trees.
- Drawback: While bagging improves prediction accuracy, it cannot be represented by a single tree. Thus, the bagged predictions are not easy to interpret.

Out-of-Bag (OOB) Error

As far as test MSE is concerned, it can be estimated in a natural and straightforward manner, without using the CV approach. To see this, note that given a training sample with n observations, the probability of an observation not included in a bootstrapped sample in bagging is $(1-1/n)^n$, which is about 36.2% for $n=30$, 36.7% for $n=300$, and 36.8% for $n \to \infty$. Thus, a bootstrapped tree roughly makes use $2/3$ of the original observations; the remaining $1/3$ observations not used for training this tree are referred to as the out-of-bag (OOB) observations. The OOB observations are natural candidates for the test sample. The test MSE can then be computed using the predictions of the OOB observations. For classification problems, we can calculate the test classification error rate using the OOB observations.

Algorithm: Computing the Out-of-Bag (OOB) MSE

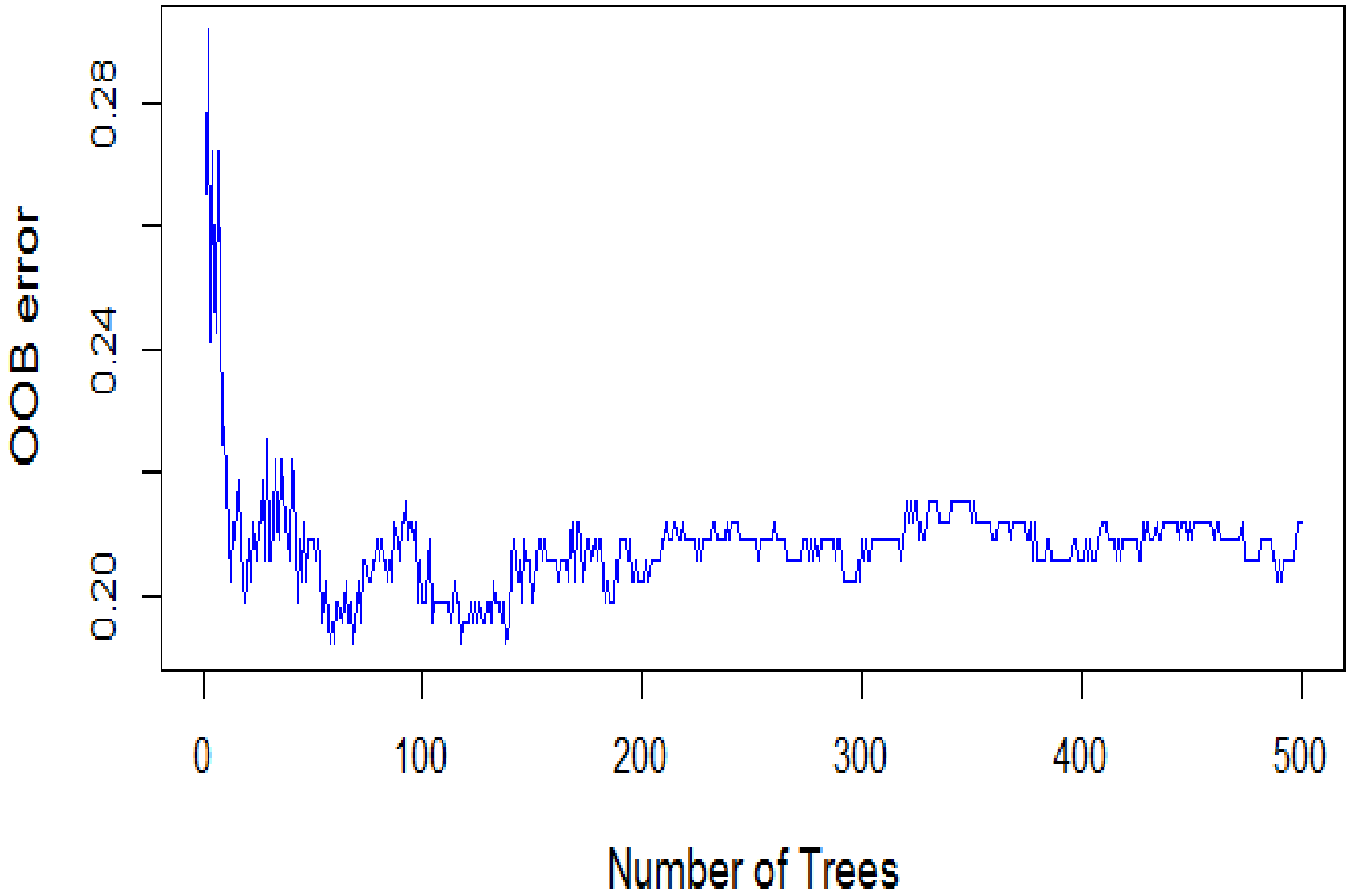
1. For a given observation $\boldsymbol{x}_i$, find the bootstrapped samples $\mathcal{S}^*(b)$ that do not include $\boldsymbol{x}_i$; let there be $B^\dagger$ such samples. For each of these $\mathcal{S}^*(b)$, grow a tree and compute its predictions based on $\boldsymbol{x}_i$: $\hat{f}^*(\boldsymbol{x}_i; b)$.
2. For y_i, the prediction based on the OOB observation $\boldsymbol{x}_i$ is

$$\hat{y}_i^{\text{OOB}} = \frac{1}{B^\dagger} \sum_{b=1}^{B^\dagger} \hat{f}^*(\boldsymbol{x}_i; b).$$

Repeat the previous steps for $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to obtain $\hat{y}_1^{\text{OOB}}, \ldots, \hat{y}_n^{\text{OOB}}$. The test MSE is the average of the OOB MSEs:

$$\frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}_i^{\text{OOB}} - y_i\right)^2.$$

Below we plot the average of the OOB MSEs based on the heart disease (AHD) data. This error drops when the number of trees increases.



Problem with Bagging

In practice, there may exist some "powerful" predictors, in the sense that their predictive ability dominates that of other predictors. These predictors are likely to be selected by top splits in most (or all) bootstrapped trees. As such, the bootstrapped trees may be similar to some extent and generate similar and correlated predictions. As far as variance reduction is concerned, correlations between variables may mitigate the effect of averaging. For example, for $z_1, \ldots, z_n$ identically distributed with variance σ^2 and pairwise correlations $\rho > 0$ (or covariances $\rho\sigma^2$), we have

$$\operatorname{var}(\bar{z}) = \frac{\sigma^2}{n} + \frac{n^2 - n}{n^2}\rho\sigma^2 = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2.$$

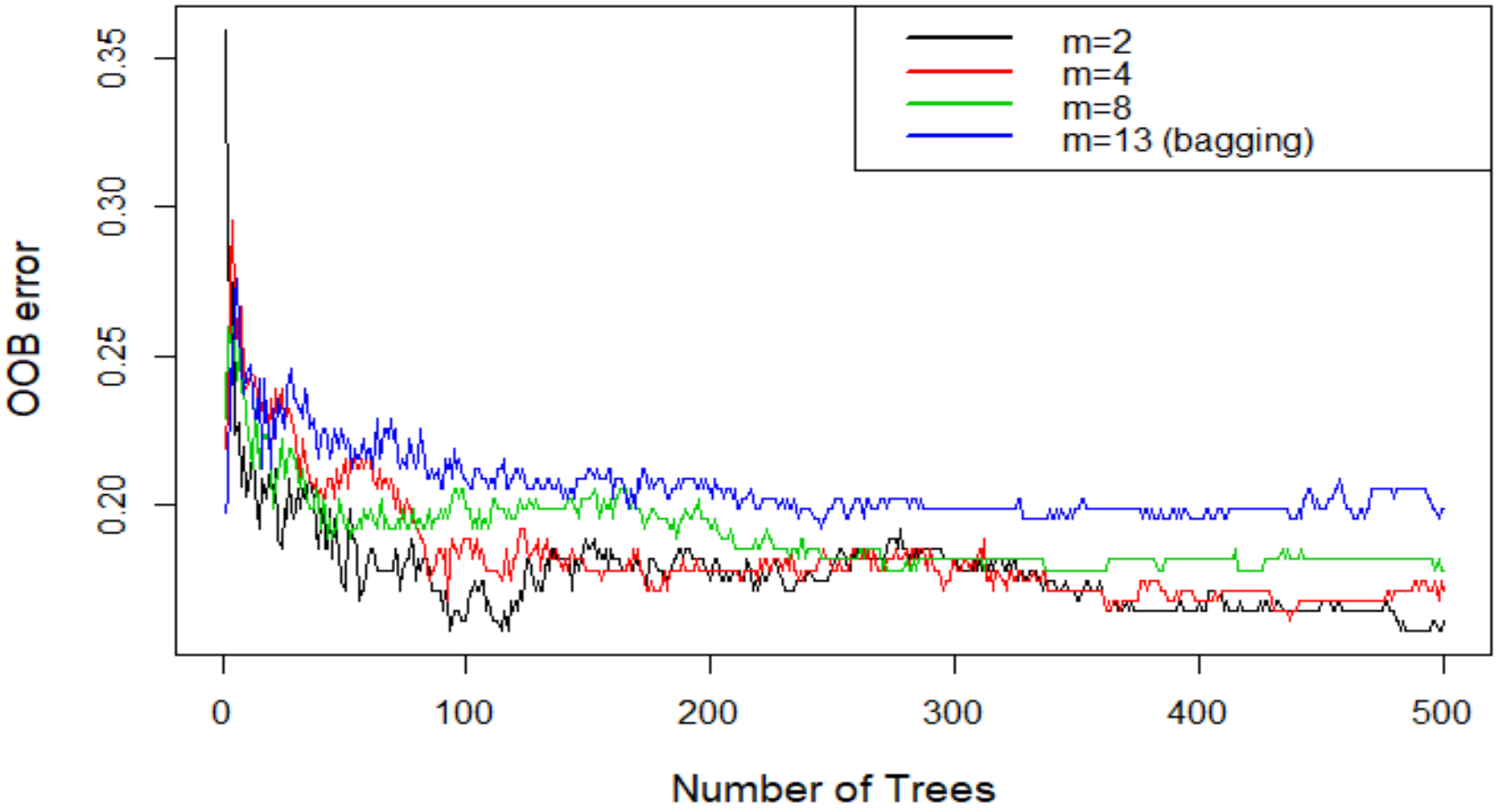
When $\rho = 0$, $\operatorname{var}(\bar{z}) = \sigma^2/n$; when $\rho = 1$, $\operatorname{var}(\bar{z})$ is simply σ^2. This shows that averaging does not reduce much variance when ρ is close to one.

Random Forest

Idea: Random forest introduces a random mechanism to reduce possible correlations between predictions. This is done by using a random subset of predictors for each split, i.e., only $m < p$ predictors, randomly drawn from p predictors, are considered at each split.

- The prediction of a random forest based on $\boldsymbol{x}$ is again the average (or majority) of B bootstrapped predictions, as in bagging.
- There are, on average, $1 - m/p$ of the splits do not involve the "powerful" predictor. For a small m, there would be more splits without this "powerful" predictor, so that other "weak" predictors stand more chance of being selected. Thus, the resulting "randomized" trees would be quite different and have less correlated predictions. It is common to set $m \approx \sqrt{p}$ or $m = p/3$.

Below we plot the average of the OOB MSEs from random forest with different m based on the heart disease (AHD) data. As there are 13 predictors, $m = 13$ corresponds to bagging, and $m = 4 \approx \sqrt{13}$ is a common choice for random forest.



Boosting

Idea: In contrast with bagging and random forest which grow a collection of trees from bootstrapped samples, boosting grows trees sequentially such that each tree captures the error resulted from the tree grown previously. In particular, the boosted tree is the sum of L trees:

$$f_L(\boldsymbol{x}) = \sum_{\ell=1}^{L} T(\boldsymbol{x}; \boldsymbol{\theta}_\ell),$$

where $T(\boldsymbol{x}; \boldsymbol{\theta}_\ell)$ is a tree grown at the step ℓ, obtained by minimizing:

$$\sum_{i=1}^{n} \left[y_i - f_{\ell-1}(\boldsymbol{x}_i) - T(\boldsymbol{x}; \boldsymbol{\theta}_\ell) \right]^2 = \sum_{i=1}^{n} \left[r_{i,\ell-1} - T(\boldsymbol{x}_i, \boldsymbol{\theta}_\ell) \right]^2,$$

where $r_{i,\ell-1} = y_i - f_{\ell-1}(\boldsymbol{x}_i)$ is the error resulted from the previous step.

To see why boosting works, recall that many functions $f(\boldsymbol{x})$ can be well approximated by an expansion that involves a set of basis functions $b(\boldsymbol{x}, \boldsymbol{\gamma}_\ell)$ (e.g., polynomial and trigonometric functions):

$$f_K(\boldsymbol{x}) = \sum_{k=1}^{K} \beta_k b(\boldsymbol{x}, \boldsymbol{\gamma}_k),$$

where β_k are the expansion coefficients and $\boldsymbol{\gamma}_k$ are vectors of parameters. Yet, to find all coefficients and parameters, one needs to minimize

$$\sum_{i=1}^{n} \left(y_i - \sum_{k=1}^{K} \beta_k b(\boldsymbol{x}_i, \boldsymbol{\gamma}_k) \right)^2.$$

This is computationally intensive, especially when the number of basis function, K, and the dimension of $\boldsymbol{x}$ are large.

A more feasible way is to approximate the solution via forward stagewise modeling. That is, at each stage, a new basis function is added to approximate the error resulted from the expansion of lower order. This amounts to minimizing:

$$\sum_{i=1}^{n} \left[y_i - f_{k-1}(\boldsymbol{x}_i) - \beta_k b(\boldsymbol{x}_i, \boldsymbol{\gamma}_k) \right]^2 = \sum_{i=1}^{n} \left[r_{i,k-1} - \beta_k b(\boldsymbol{x}_i, \boldsymbol{\gamma}_k) \right]^2,$$

where $r_{i,k-1} = y_i - f_{k-1}(\boldsymbol{x}_i)$ is the error resulted from the previous expansion $f_{\ell-1}$.

Q: Why does this approach provide an approximation to the desired solution to $\min \sum_{i=1}^{n} \left(y_i - \sum_{k=1}^{K} \beta_k b(\boldsymbol{x}_i, \boldsymbol{\gamma}_k) \right)^2$?

Note that a function $\boldsymbol{f}$ that minimizes the squared error loss, $(\boldsymbol{y}-\boldsymbol{f})'(\boldsymbol{y}-\boldsymbol{f})$, can be found by taking the steepest descent step along the negative gradient direction in the function space:

$$-\nabla_{\boldsymbol{f}}[(\boldsymbol{y}-\boldsymbol{f})'(\boldsymbol{y}-\boldsymbol{f})] = \boldsymbol{y}-\boldsymbol{f}.$$

For $\boldsymbol{f}=\boldsymbol{f}_{k-1}$, the expansion with $k-1$ basis functions, the i^{th} element of $\boldsymbol{y}-\boldsymbol{f}_{k-1}$ is $r_{i,k-1}=y_i-f_{k-1}(\boldsymbol{x}_i)$. Thus, the newly added basis function in effect approximates the steepest descent step from the previous expansion.

In analogy, finding a tree that is as close as possible to $r_{i,\ell-1}$ at each step of the boosting procedure is to approximate the steepest descent step from the tree grown previously. This is also known as Gradient boosting.

Algorithm: Gradient Boosting for Regression Trees

1. Set the initial predictions for y_i as $\hat{f}_0(\boldsymbol{x}_i) = 0$, and the residuals as $r_{i,0} = y_i - \hat{f}_0(\boldsymbol{x}_i)$.
2. For each $\ell = 1, \ldots, L$:
   1. Grow a tree with d splits ($d+1$ terminal nodes) from the modified training data $\{(r_{i,\ell-1}, \boldsymbol{x}_i), i = 1, \ldots, n\}$; denote its predictions as $\hat{\varphi}_\ell(\boldsymbol{x}_i)$.
   2. The predictions of y_i are updated as:

$$\hat{f}_\ell(\boldsymbol{x}_i) = \hat{f}_{\ell-1}(\boldsymbol{x}_i) + \lambda_i \hat{\varphi}_\ell(\boldsymbol{x}_i), \quad i = 1, \ldots, n,$$

   with λ_i a tuning parameter, and the residuals are: $r_{i,\ell} = y_i - \hat{f}_\ell(\boldsymbol{x}_i)$.
3. The final boosted predictions are:

$$\hat{f}_L(\boldsymbol{x}_i) = \sum_{\ell=1}^{L} \lambda_i \hat{\varphi}_\ell(\boldsymbol{x}_i), \quad i = 1, \ldots, n.$$

- Gradient boosting can be understood as a slow learning process. It takes many steps (L) to learn, and at each step, it learns from the residuals of the preceding step using a small tree (small d) and a small step length (λ_i) in the negative gradient direction.
- The step length λ_i, also known as the learning rate, is a small constant. For smaller λ, it takes a larger number of steps, L, to achieve the same degree of approximation accuracy (training MSE). Yet, a larger L requires more computation and leads to over-fitting. HTF (2009, p. 365) suggest setting $\lambda < 0.1$ and determining L by early stopping.
- Stochastic gradient boosting (Friedman, 1999): At each step, grow a tree from a random sub-sample with πn observations, e.g. $\pi = 0.5$ (or a smaller π when n is large). This helps reduce the correlations among the constituent trees and improve computational efficiency.

Tree Size d for Boosting

- When $d = 1$, each tree in the algorithm reduces to a "stump" and depends on only one predictor x_{ij}. Writing $\tilde{\varphi}_\ell(\boldsymbol{x}_i) = \lambda_i \hat{\varphi}_\ell(\boldsymbol{x}_i)$, we have

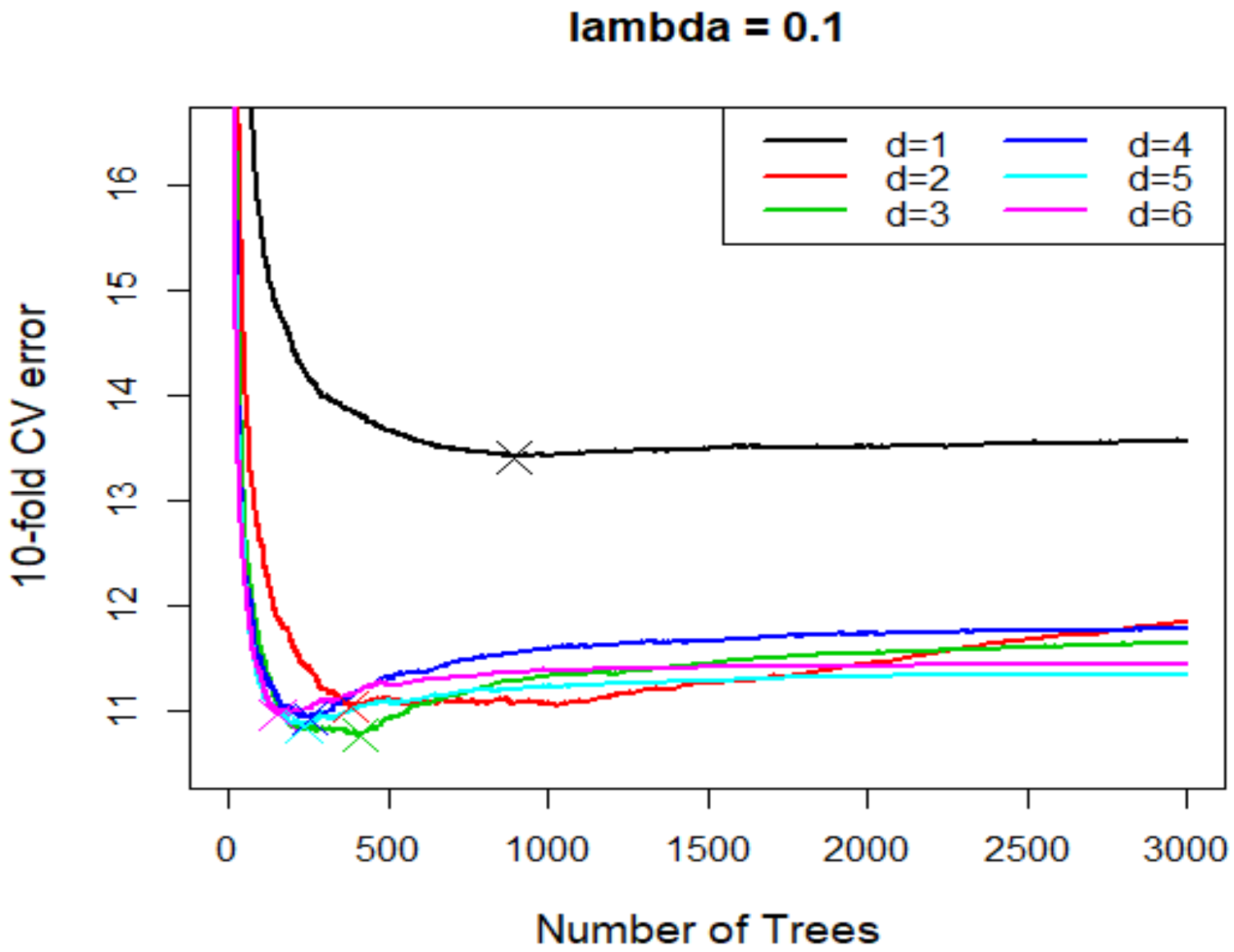
$$\hat{f}_L(\boldsymbol{x}_i) = \sum_{\ell=1}^{L} \tilde{\varphi}_\ell(\boldsymbol{x}_i) = \sum_{j=1}^{p} \sum_{\ell \in G_j} \tilde{\varphi}_\ell(x_{ij}),$$

  where G_j is the group that $\tilde{\varphi}_\ell$ depends only on the j th predictor. The boosted tree is thus an additive function of predictors x_{ij}, $j = 1, \ldots, p$.
- When $d = 2$, the boosted tree is an additive function of predictors x_{ij} and their pairwise interactions $x_{ij}x_{ih}$, $j, h = 1, \ldots, p$. When $d > 2$, the boosted tree is an additive function of up to d-way interactions.
- HTF (2009, p. 363) suggest choosing $3 \leq d \leq 7$. They also find that it is unlikely to require $d > 10$.

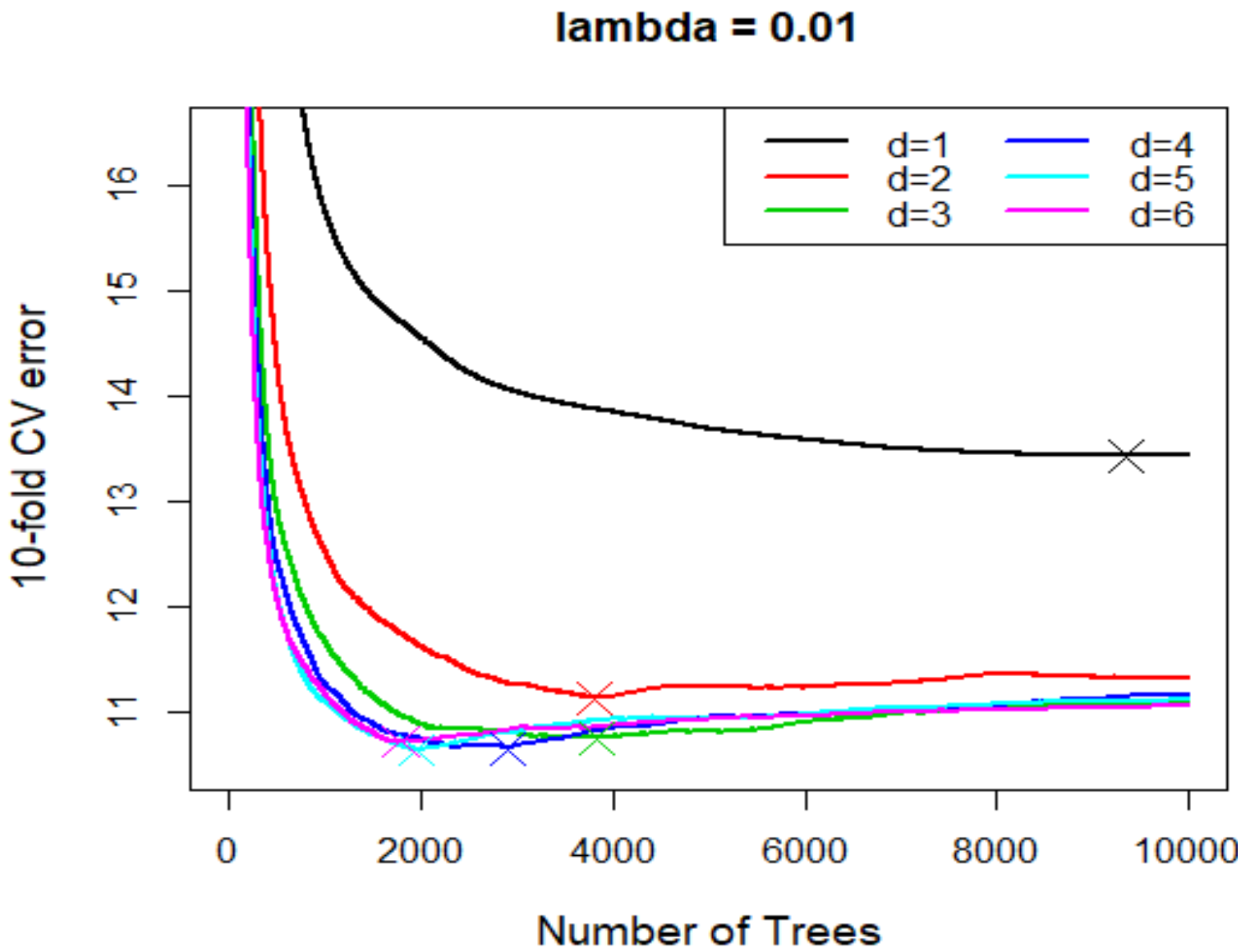
Example: Boston Housing

- We want to predict the median value of owner-occupied homes in Boston (`medv`) using 13 predictors, including `rm` (average number of rooms per dwelling), `age` (proportion of owner-occupied units built prior to 1940), `black` (proportion of blacks by town), `rad` (index of accessibility to radial highways), and so on.
- Here we set $\lambda = 0.1$, 0.01, and 0.001, and evaluate its cross validation error across different number of trees L, and different numbers of splits d for each tree ($d = 1, 2, \ldots, 6$).

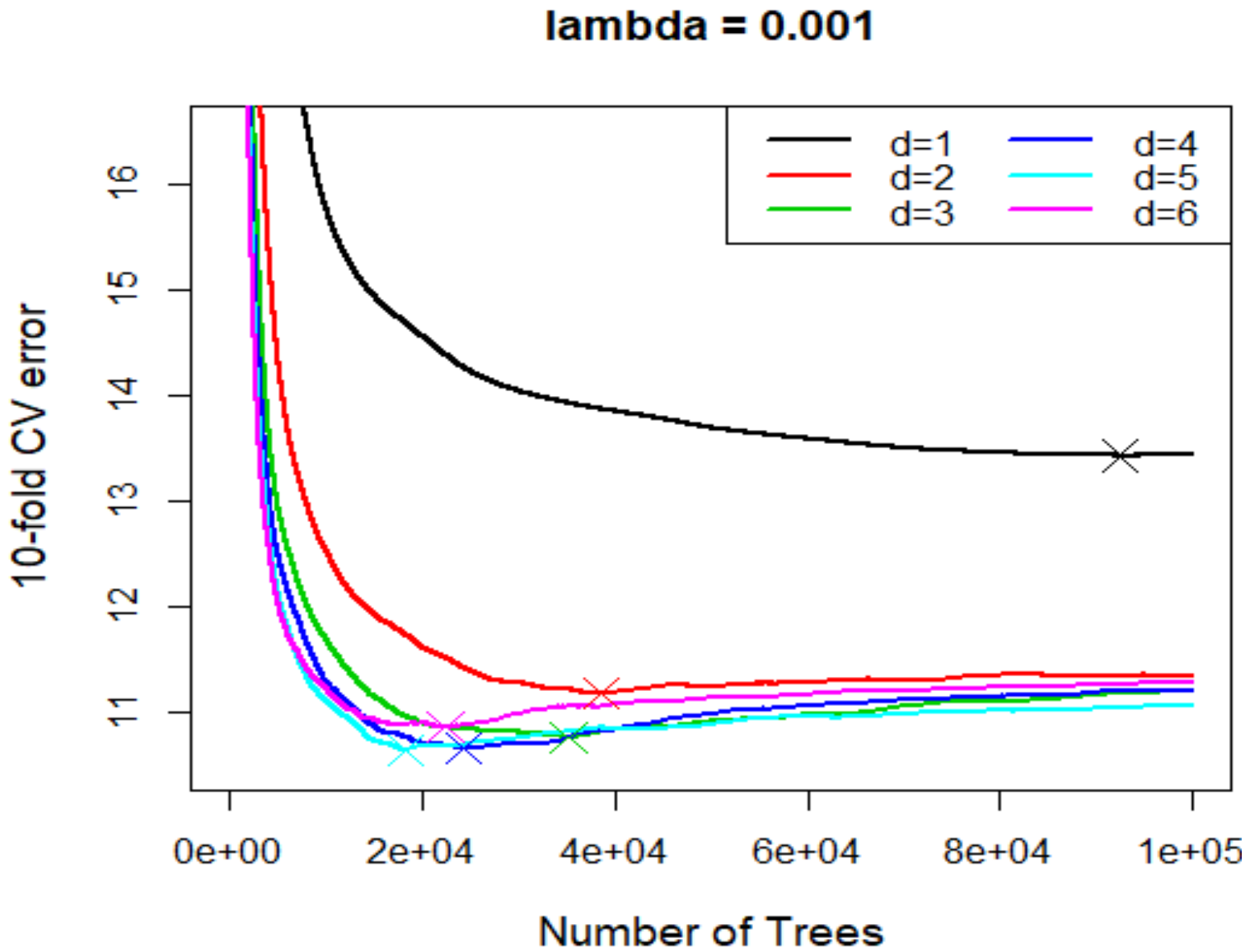
$d = 3$ yields the lowest 10-fold CV error (10.7625) at $L = 411$.



$d = 5$ yields the lowest 10-fold CV error (10.6351) at $L = 1956$.



$d = 5$ yields the lowest 10-fold CV error (10.6380) at $L = 18210$. Note that a small λ does require a large L to achieve good performance.



References and Acknowledgement

References:

1. James, G., D. Witten, T. Hastie, and R. Tibshirani (2021). *An Introduction to Statistical Learning, with Applications in R*, 2nd edition, New York: Springer. (JWHT (2021))
2. Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning*, Second Edition, New York: Springer.

Some of the figures in this presentation are taken from JWHT (2021) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani