MATH 189
Tree-Based Methods

Wenxin Zhou

UC San Diego

Time: 2:00—3:20 & 3:30—4:50pm TueThur

Zoom ID: 985 7463 1067

Outline

- In previous lectures, we introduced least squares method for linear regression.
  - Regression to the mean
  - Simple linear regression
  - Multivariate linear regression

- This week we will introduce some tree-based methods.
  - Decision tree
    - Regression tree
    - Tree pruning
    - Classification tree
  - Bagging
  - Random forests
  - Boosting

Tree-Based Methods

- Tree-based methods are learning tools that involve segmenting the predictor space into a number of simple regions.

- In order to make a prediction for a given observation, 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 or tree-based methods.

- Tree-based methods are simple and useful for interpretation. Decision trees can be applied to both regression and classification problems.

Regression Tree

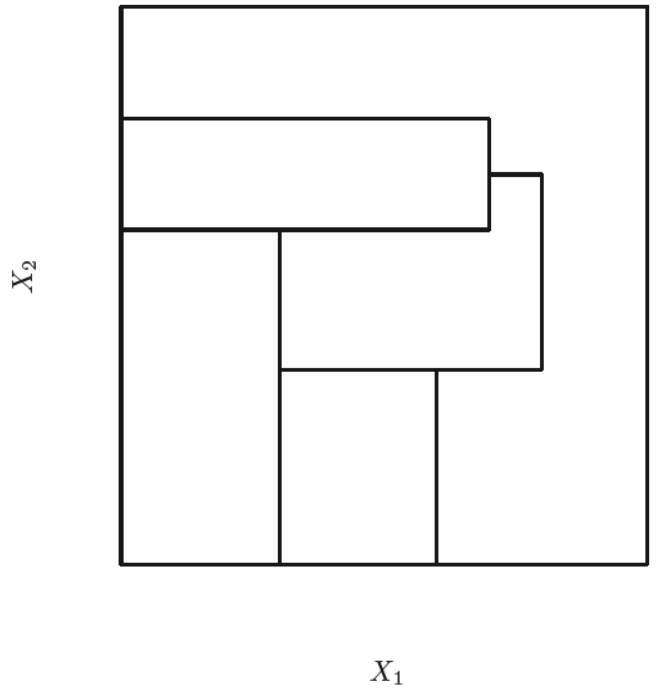
- Consider a multivariate regression problem

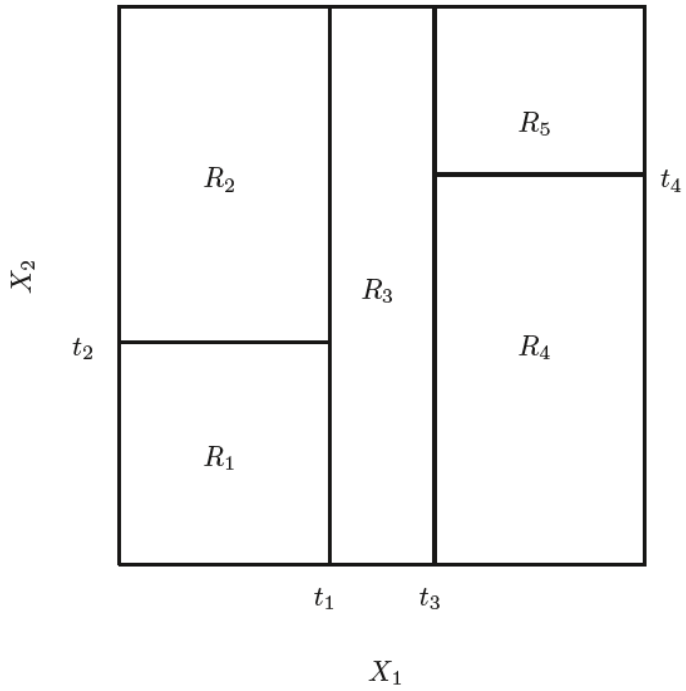
$$Y = f(X_1, X_2, \dots, X_p) + \epsilon.$$

- We now briefly discuss the process of building a regression tree by two steps:
  1. We divide the predictor space—that is, the set of possible values for $X_1, X_2, \dots, X_p$—into J distinct and non-overlapping regions, $R_1, R_2, \dots, R_J$.
  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.
- For instance, suppose that in Step 1 we obtain two regions, R_1 and R_2, and the response mean of the training observations in the R_1 is 10, while the response mean of the training observations in R_2 is 20.
- Then for a given observation $X = x$, if $x \in R_1$ we will predict a value of 10, and if $x \in R_2$ we will predict a value of 20.

Segmenting Predictor Space

- The shape of the tree we grow depends on how we 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 rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.





Tree Growing

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

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

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

- Unfortunately, it is computationally infeasible to consider every possible partition of the predictor space into J boxes.

- Because of this, we take a top-down, greedy approach that is known as recursive binary splitting.

- This approach starts from the top of the tree and then successively splits the predictor space. Each split is indicated via two new branches further down the tree.

Binary Splitting

- We first select a predictor X_j and the cut point s such that we split the predictor space into two regions

$$R_1(j,s) = \left\{X = (X_1, \ldots, X_p)' | X_j < s\right\} \quad \text{and} \quad R_2(j,s) = \left\{X | X_j \geq s\right\},$$

and seek the values of j and s that minimize

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

where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are mean responses for the training sample in $R_1(j,s)$ and $R_2(j,s)$.

- In practice, finding the values of j and s that minimize $\text{RSS}(j,s)$ can be done very quickly, especially when the number of features p is not too large.

Recursive Binary Splitting

- Next, we repeat the process, looking for the best predictor and best cut point in order to split the data further and 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 R_1 and R_2.
- 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 stop when no region contains more than five observations.
- Once the regions $R_1, R_2, \ldots, R_J$ have been created, we predict the response for a given testing/future observation using the response mean in the region where the testing observation belongs.

Example: Baseball Dataset (see Sabermetrics)

- This dataset collected the statistics of 263 players in Major League Baseball from the season 1986.
- The response variable is the annual salary at the opening day of season 1987.

- Here, we use two covariates to illustrate the implementation of regression tree.

- Consider the regression model

$$\log(Salary) = f(Year, Hits) + \epsilon,$$

where Year is the number of years this player has played in the MLB and Hits is the number of hits that he made in the previous year.

Example: Binary Splitting Procedure

- First, we find values of j and s to split the two-dimensional predictor space into two regions

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

which minimize the $\text{RSS}(j,s) = \sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$.

- The variable we find is Year along with the cut point 4.5. In other words, the first split creates two regions

$$R_1 = \{X|Year < 4.5\} \quad \text{and} \quad R_2 = \{X|Year \geq 4.5\}.$$

- In the second step, we can achieve the smallest RSS by further splitting R_2 into two regions with variable Hit and cut point 117.5.

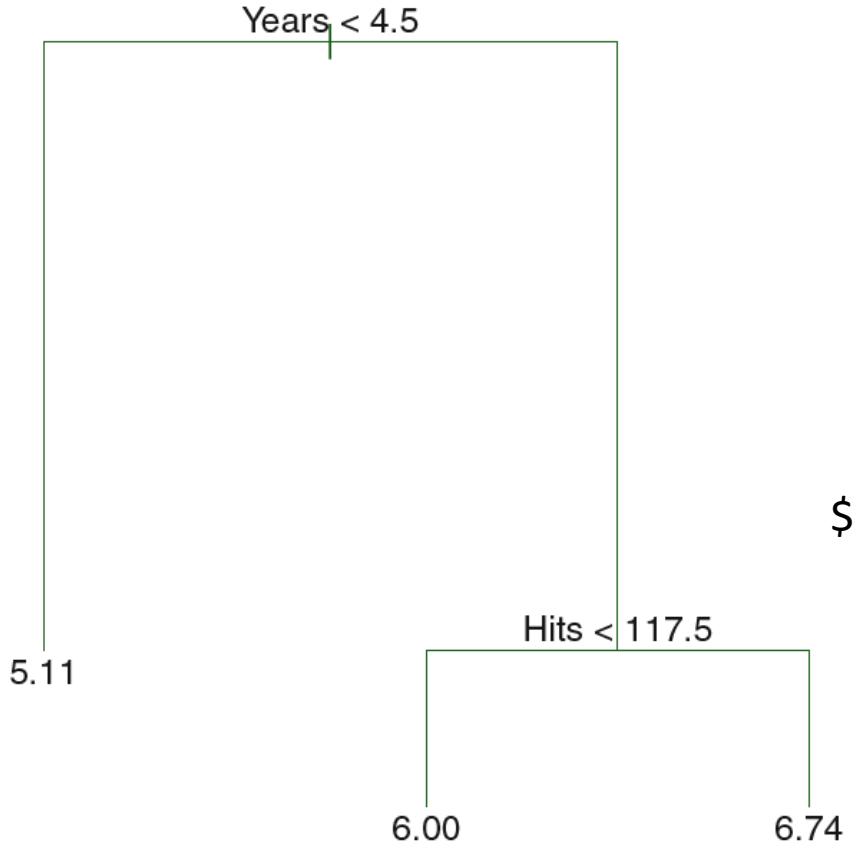
$$R_2 = \{X|Year \geq 4.5 \; and \; Hits < 117.5\} \quad \text{and} \quad R_3 = \{X|Year \geq 4.5 \; and \; Hits \geq 117.5\}.$$

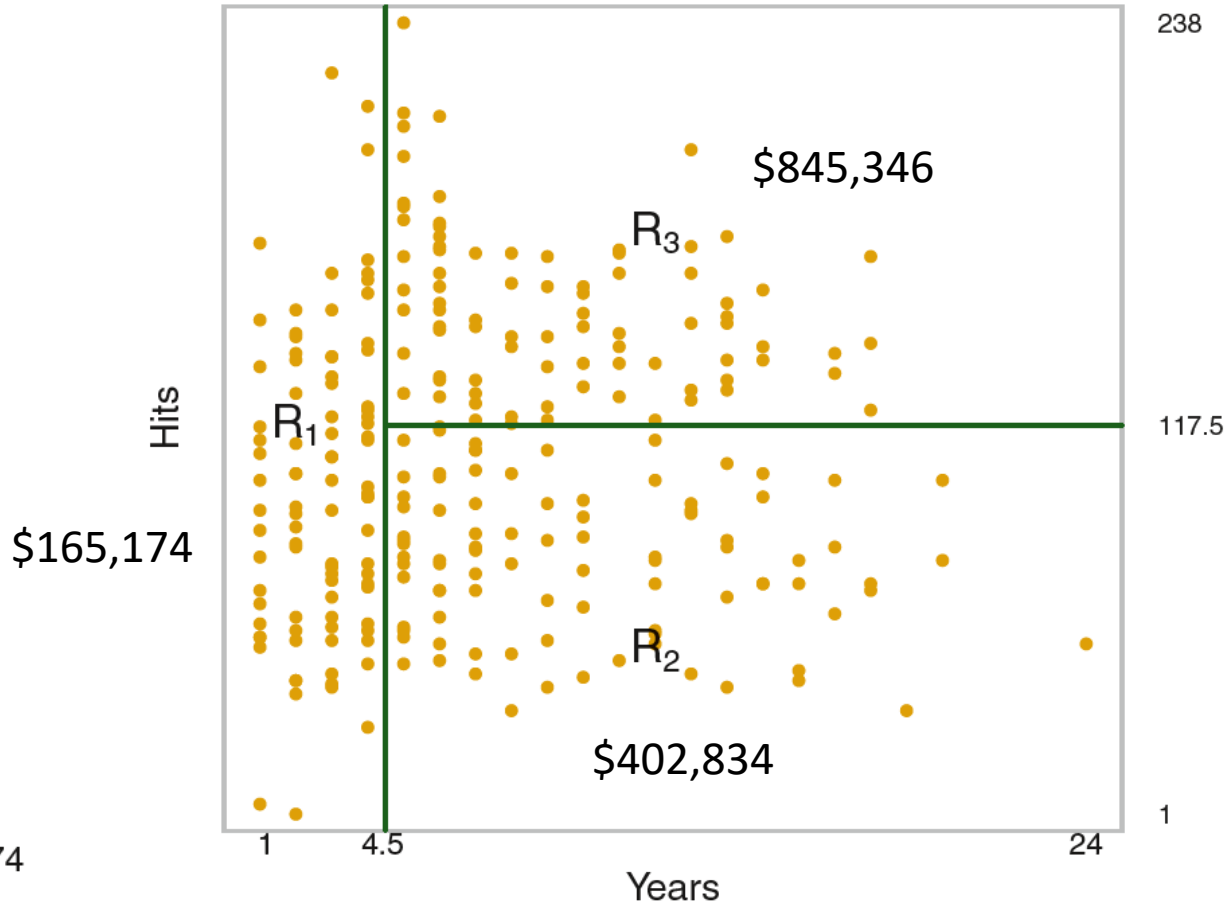
- We stop after step two, and the binary splitting procedure produces 3 regions.

Example: Tree and Space partition

- The binary splitting procedure creates 3 regions

  $R_1 = \{X|Year < 4.5\}$, $R_2 = \{X|Year \geq 4.5 \ \& \ Hits < 117.5\}$ and $R_3 = \{X|Year \geq 4.5 \ \& \ Hits \geq 117.5\}$.





Tree Pruning

- The binary splitting procedure is based on RSS and may suffer from overfitting issue. A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of some bias.

- One possible alternative is to stop tree growing if the decrease of RSS does not exceed a threshold. However, this early stopping rule can be short-sighted since a seemingly worthless early split may be followed by a very good split later on.

- Hence, a better strategy is to grow a very large tree T_0, and then prune it back to obtain a subtree.

- Intuitively, our goal is to select a subtree with the lowest testing error. However, estimating the cross-validation error for every possible subtree would be too expensive, since there is an extremely large number of possible subtrees. Instead, we need a way to select a small set of subtrees for consideration.

Cost Complexity Pruning

- Instead of considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α.

- For each α given, there is a subtree $T \subset T_0$ that minimizes the cost complexity function

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

  Here $|T|$ indicates the number of regions in the tree T, R_m is the m-th region and $\hat{y}_{R_m}$ is the predicted response for m-th region (e.g. sample mean).

- The idea of cost complexity function is similar as information criterion (AIC or BIC). The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
  - When $\alpha = 0$, T equals T_0.
  - When $\alpha \to \infty$, T contains only one region (no splitting).

Cost Complexity Pruning: Algorithm

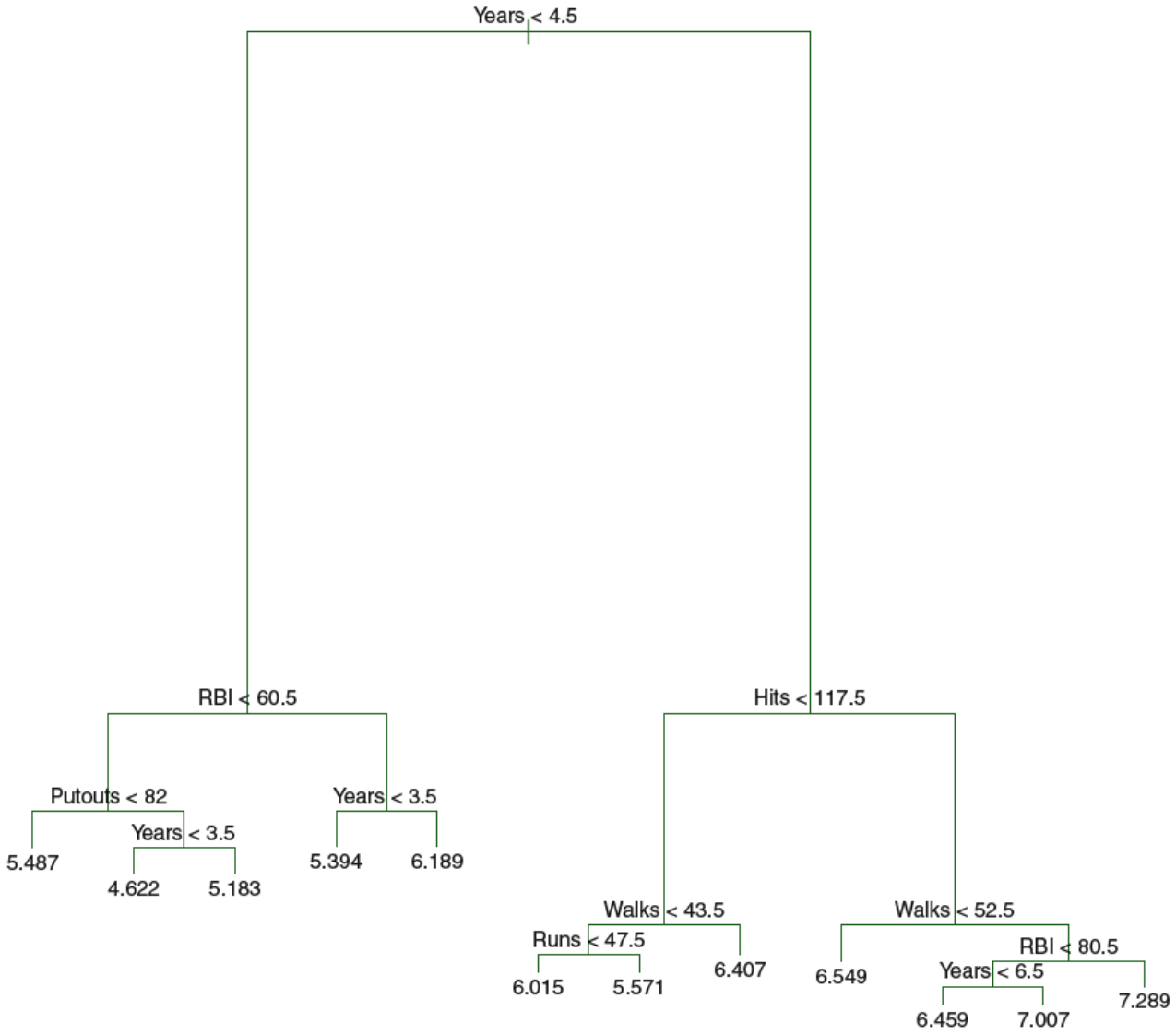
- As we increase α, branches get pruned from the tree in a nested fashion, resulting a sequence of subtrees indexed by α. In practice, we select α using cross-validation (CV).

Algorithm: Building a Regression Tree

1. Use recursive binary splitting to grow a large tree T_0, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to T_0 to obtain a sequence of best subtrees, indexed by α.
3. Use K-fold CV to choose α. For each fold $k = 1, \ldots, K$:
   a) Repeat Steps 1 and 2 on all but the k-th fold of the training data.
   b) Evaluate the prediction error on k-th fold, as a function of α.

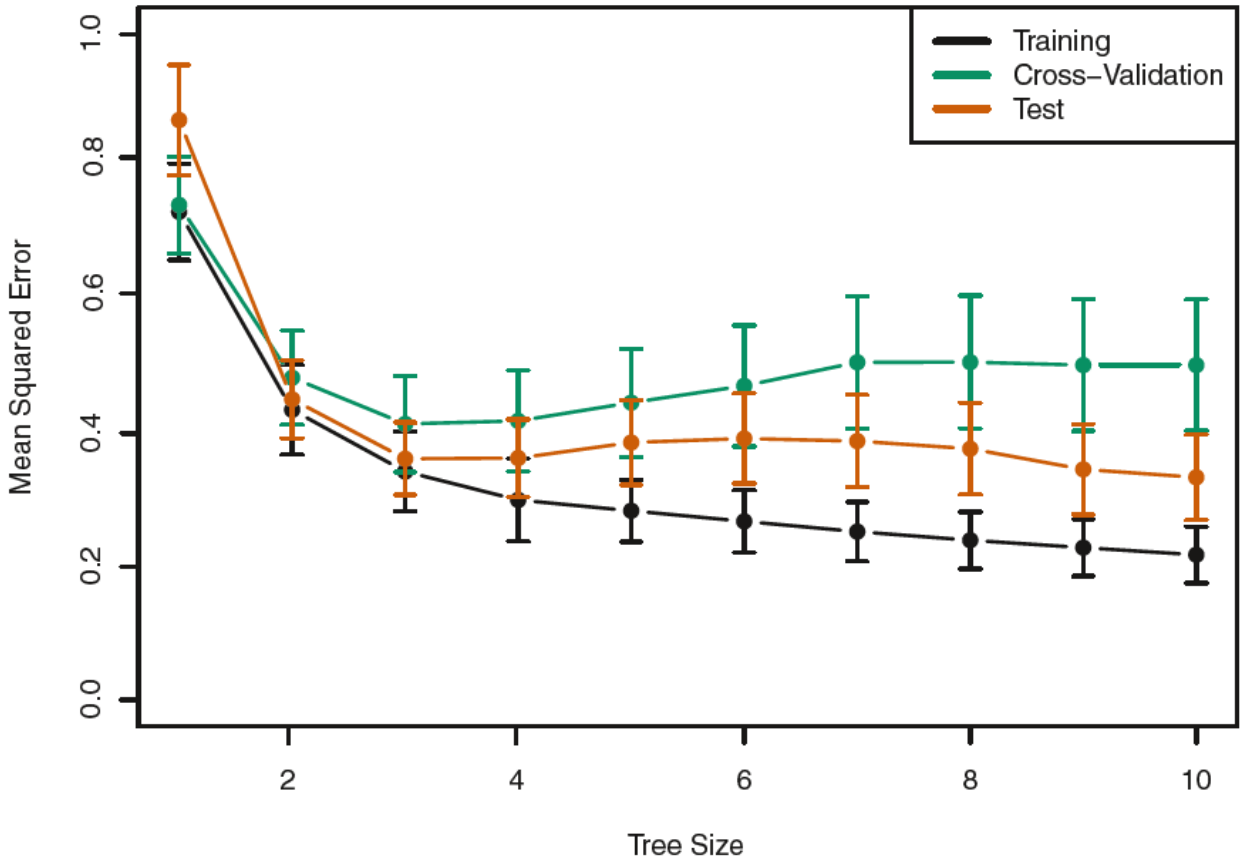
   Average the results over K folds for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Example: Grow a Large Tree for Baseball Dataset



- We randomly divided the dataset in two halves, 132 observations for the training set and 131 observations for the testing set.
- Now, we grow a large regression tree for Baseball dataset using 9 predictors.
- This large un-pruned tree contains 12 regions.

Example: Select Subtree Size With CV



- Now we vary α in cost complexity function

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

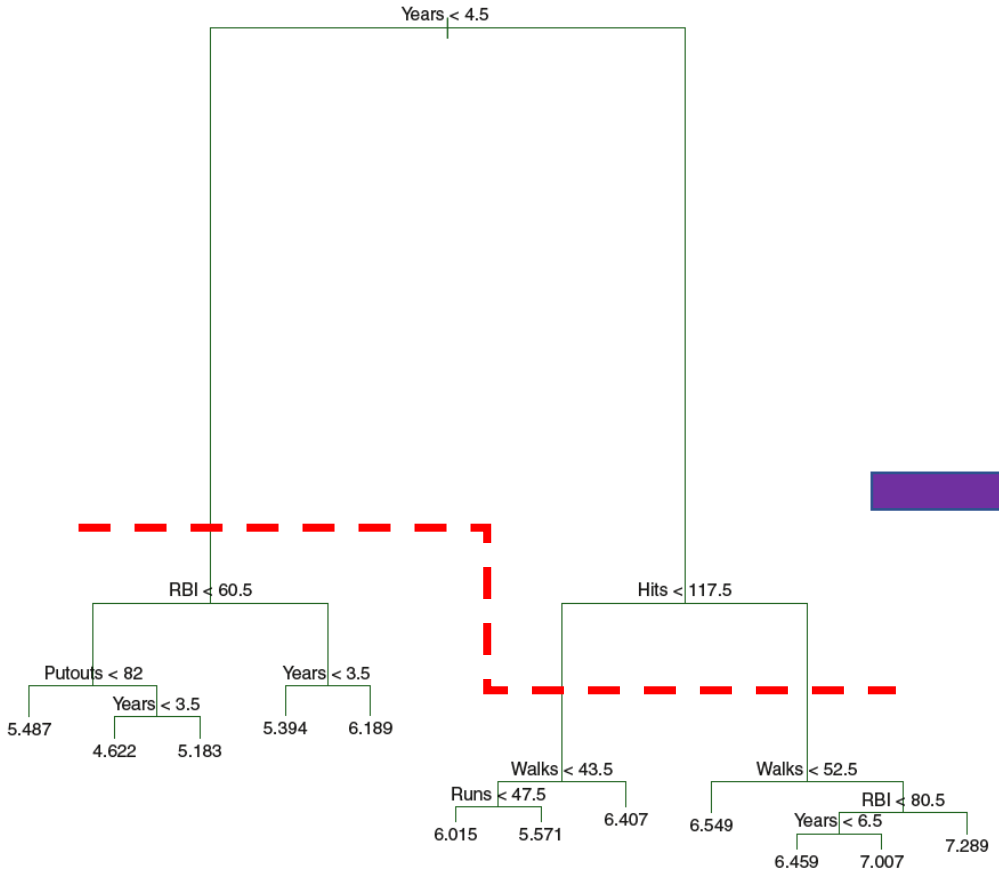
  to create subtrees with different numbers of regions.

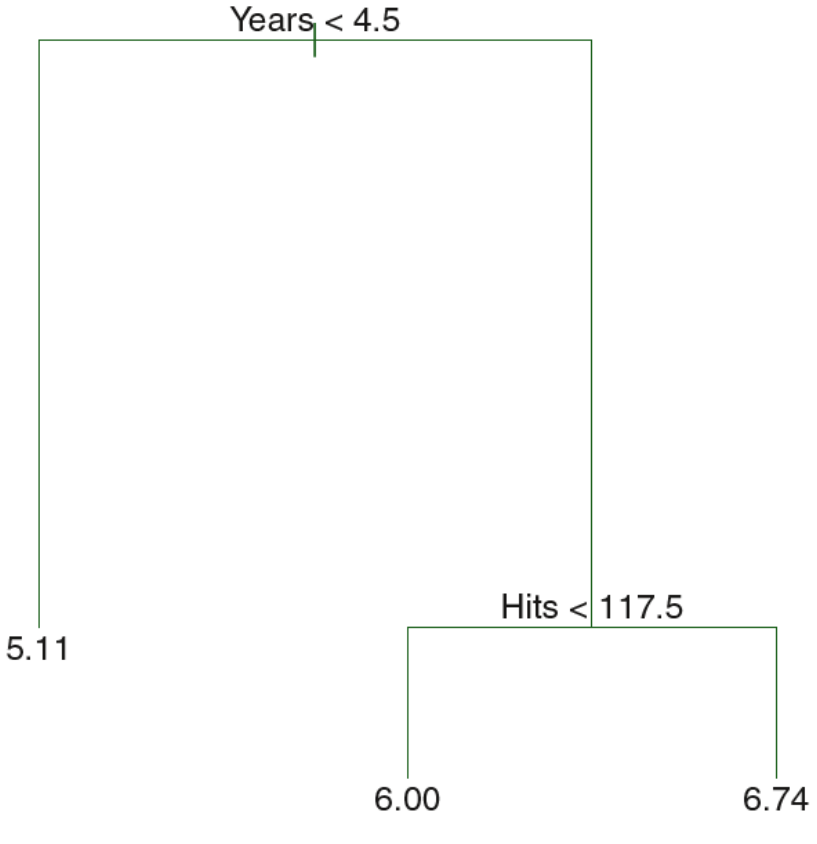
- Finally, we performed six-fold cross validation in order to estimate the cross-validated MSE of the trees as a function of tree size (α).

- The CV indicates a subtree of size 3!

Example: Tree Pruning

- Based on the tree size selected by CV, we prune the large tree into the subtree with 3 regions.
- The pruned tree has almost the same prediction error as the large tree of size 10.





Classification Tree

- A classification tree is very similar to a regression tree, except that it predicts a qualitative response rather than a quantitative one.
- For a classification tree, we classify each observation to the most commonly occurring class of training observations in the region where it belongs.
- The task of growing a classification tree is quite similar to the task of growing a regression tree. Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- To measure the performance of a classification tree, we not only care the class prediction corresponding to a particular region, but also the class proportions among the training observations that fall into that region.

Classification Error Rate and Gini Index

- A natural alternative to RSS is the classification error rate, which is defined as the fraction of the training sample in a region that do not belong to the most common class:

$$\text{Error}(R_m) = 1 - \max_k(\hat{p}_{mk}),$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class.

- However, it turns out that classification error rate is not sufficiently sensitive for tree-growing, and in practice an alternative measure called Gini Index is preferable:

$$\text{Gini}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

- Gini index is a measure of total variance across K classes. Gini index takes a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. Gini index measures the region purity—a small value indicates that a region contains predominantly observations from a single class.

Tree versus Linear Models

- Tree-based methods have a very different flavor from the classic linear regression methods as we discussed before.

- In particular, linear regression assumes a model of the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j\,, \quad X = (X_1, \ldots, X_p)'.$$

- In contrast, regression tree assumes a model of the form

$$f(X) = \sum_{m=1}^{M} c_m \cdot 1_{(X \in R_m)},$$

where $R_1, \ldots, R_M$ represent a partition of feature space (space in which X takes values).

Which model is better?

- It depends on the problem in hand. In general, linear regression has an advantage on prediction performance and regression tree has an advantage of interpretability.

Pros and Cons of Decision Trees

- Decision trees for regression and classification have a number of advantages over the classical approaches.

Advantages of Trees

1. Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
2. Some people believe that decision trees more closely mirror human decision-making process.
3. Trees can be displayed graphically, and are easily interpreted even by a non-expert.
4. Trees can easily handle qualitative predictors without the need to create dummy variables

Dis-advantages of Trees

1. Trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.

2. Decision trees suffer from high variances. A small modification of dataset may produce a completely different tree.

Bagging

- To address the disadvantages of decision trees, some methods like bagging, random forests, and boosting use trees as building blocks to construct more powerful prediction models.

- Bagging, or bootstrap aggregation, is a general-purpose procedure for reducing the variance of a statistical learning method and is particularly useful for decision trees.

- In this approach, we generate B different bootstrapped training datasets.

- We then train our method on the b-th bootstrapped training set in order to get $\hat{f}_b(x)$, and finally average all the predictions, to obtain the bagging estimator

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

Some Discussions on Bagging

1. To apply bagging to regression trees, we construct B regression trees using B bootstrapped training sets, and average the resulting predictions.

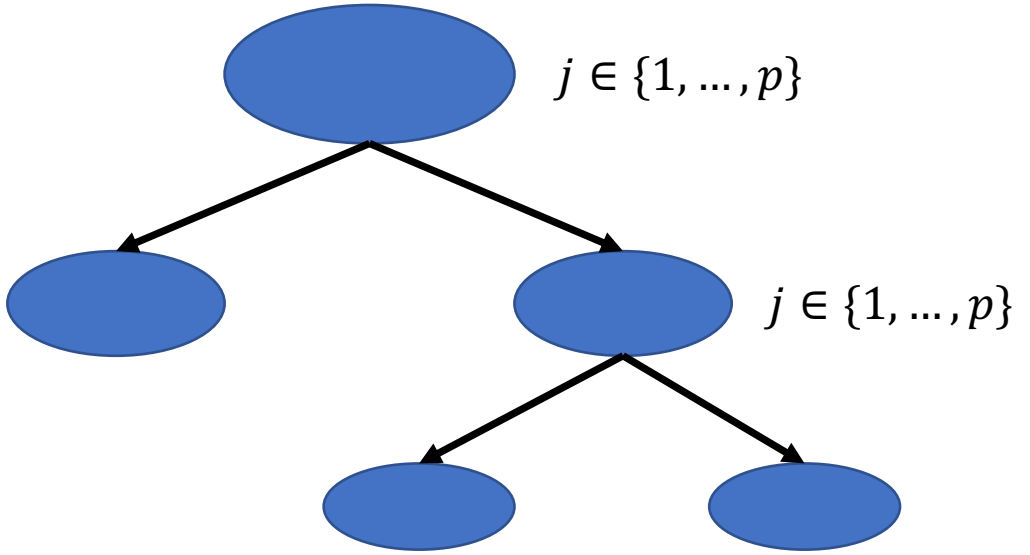
2. These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias. Averaging these B trees reduces the variance to some extent.

3. Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.

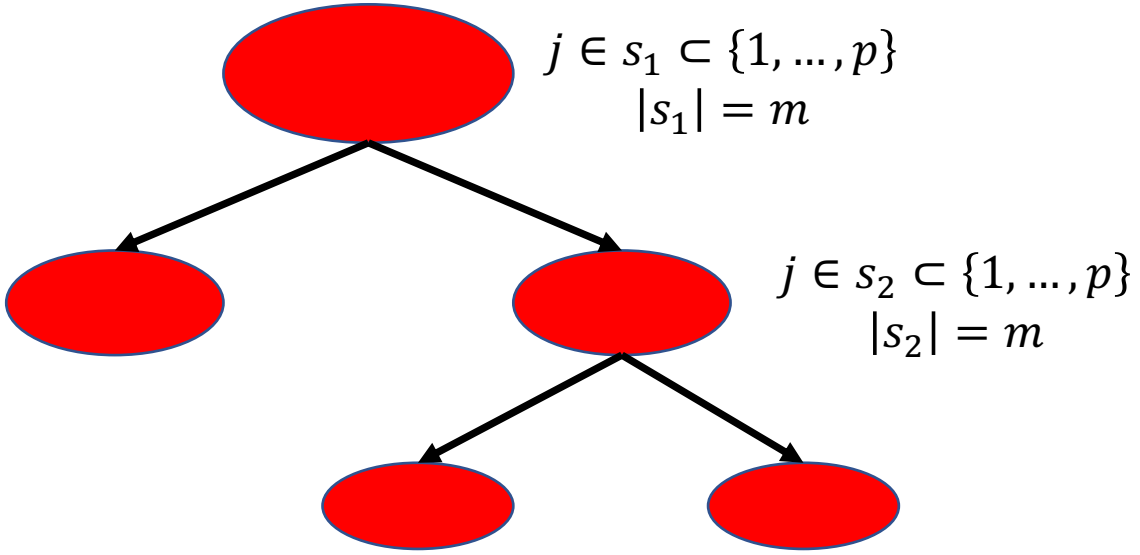
4. The bagging idea can also be implemented to classification trees. One simple approach is to follow majority voting: the overall prediction is the most commonly occurring class among the B predictions.

Random Forests

- Random forests provide an improvement over bagged trees by a random small tweak that decorrelates the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates instead of the full set of p predictors.



Bagging



Random Forests

Why Random Forests?

- In other words, in building a random forests, at each split in the tree, the algorithm is restricted to considering only a small subset of available predictors. This may sound crazy, but it has a clever rationale.

- Suppose that there is one very strong predictor in the dataset, along with a number of other moderately strong predictors. Then 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 bagged trees will look quite similar to each other and are highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.

- In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.

Intuition of Random Forests

- Random forests overcome the correlation problem by forcing each split to consider only a subset of the predictors. Therefore, on average $(p - m)/p$ of the splits will not even consider the strongest predictor, and so other predictors will have more of a chance.

- We can think of this process as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable.

- The main difference between bagging and random forests is the choice of predictor subset size m. When $m = p$, the two methods are equivalent.

- Using a small value of m in building random forests will typically be helpful when we have a large number of correlated predictors. In practice, a popular choice is to set $m = \sqrt{p}$.

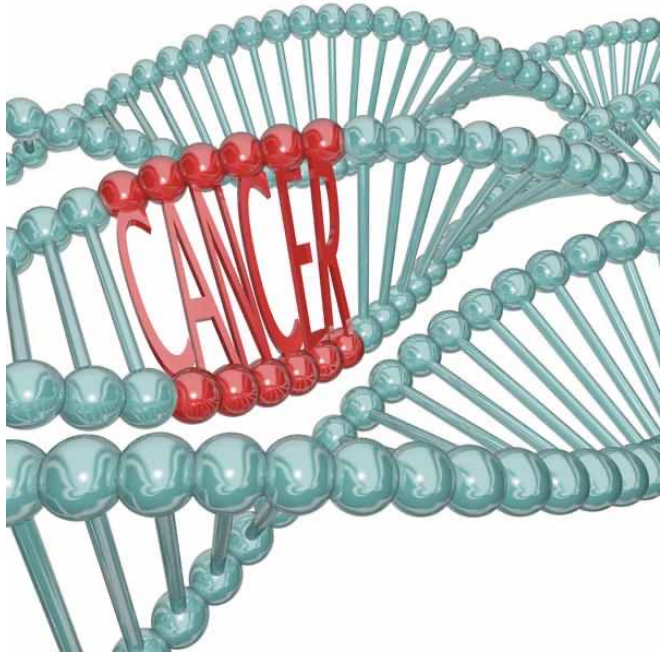
Example: Cancer Type Classification

- We apply bagging and random forests to a high-dimensional biological dataset that consists of expressions of 500 genes measured on tissue samples of 349 patients.
- In this data set, each patient has a qualitative label with 15 different levels: either

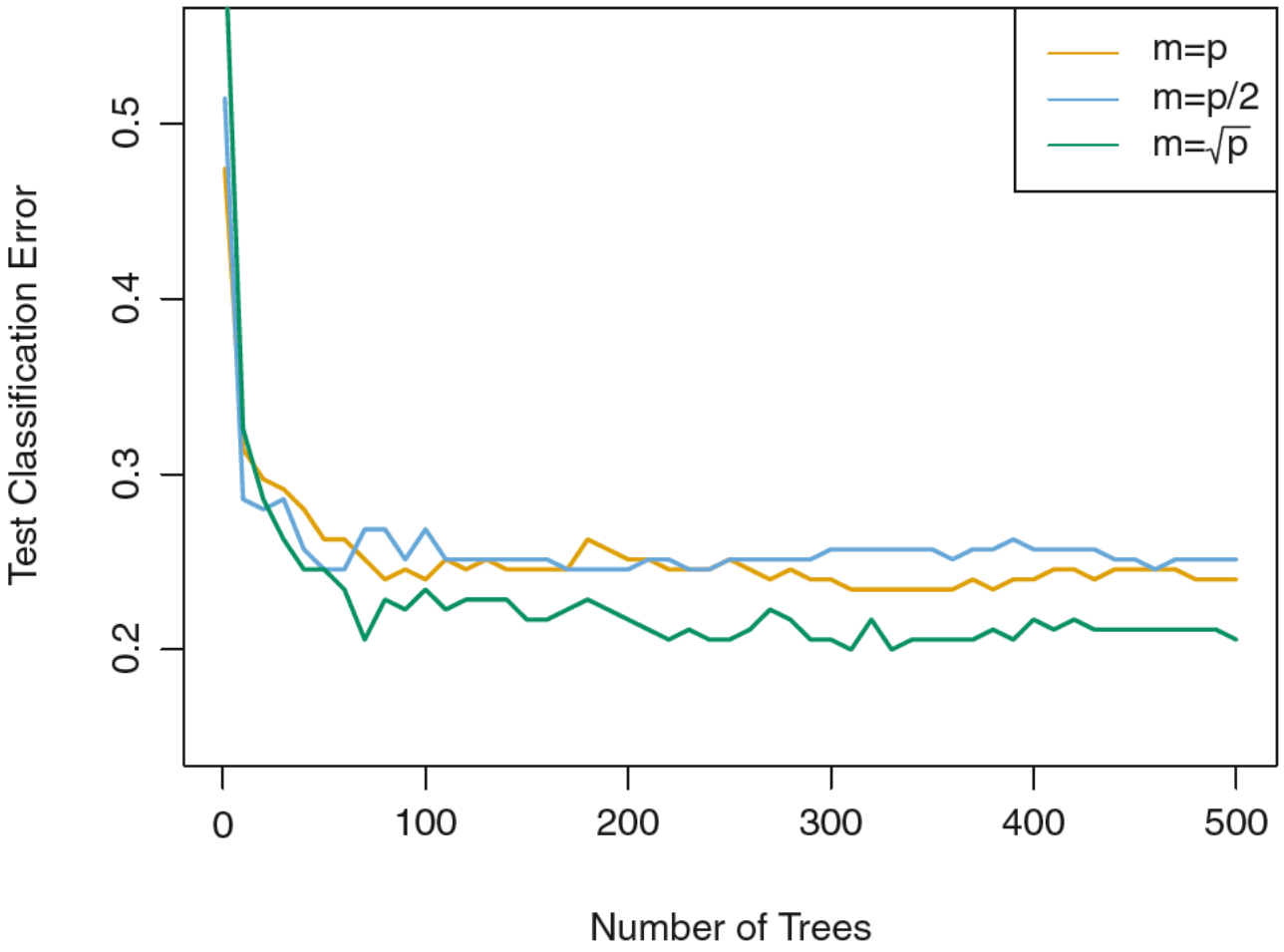
  $y = 0$ stands for normal or

  $y = 1, \dots, 14$ stands for different types of cancer.

- Our goal is to build and compare some tree-based methods to predict cancer types based on the gene expressions.



Example: Random Forest versus Bagging



- We randomly divide the observations into training and test sets, and apply random forests to the training set for three different values of m, the number of splitting variables.
- We see that using 400 trees is sufficient to give good performance, and that the choice $m = \sqrt{p}$ gave a small improvement in test error over bagging ($m = p$) in this example.
- As a benchmark, the error rate of a single tree is 45.7%.

Boosting

- Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original dataset. In other words, each tree is grown using information from previously grown trees.

Algorithm: Boosting for Regression Tree

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i = 1, \dots, n$ in the training set.
2. For $b = 1, \dots, B$, 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:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   c) Update the residuals
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Intuition of 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. That is, we fit a tree using the current residuals, rather than the outcome Y, as the response. 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.

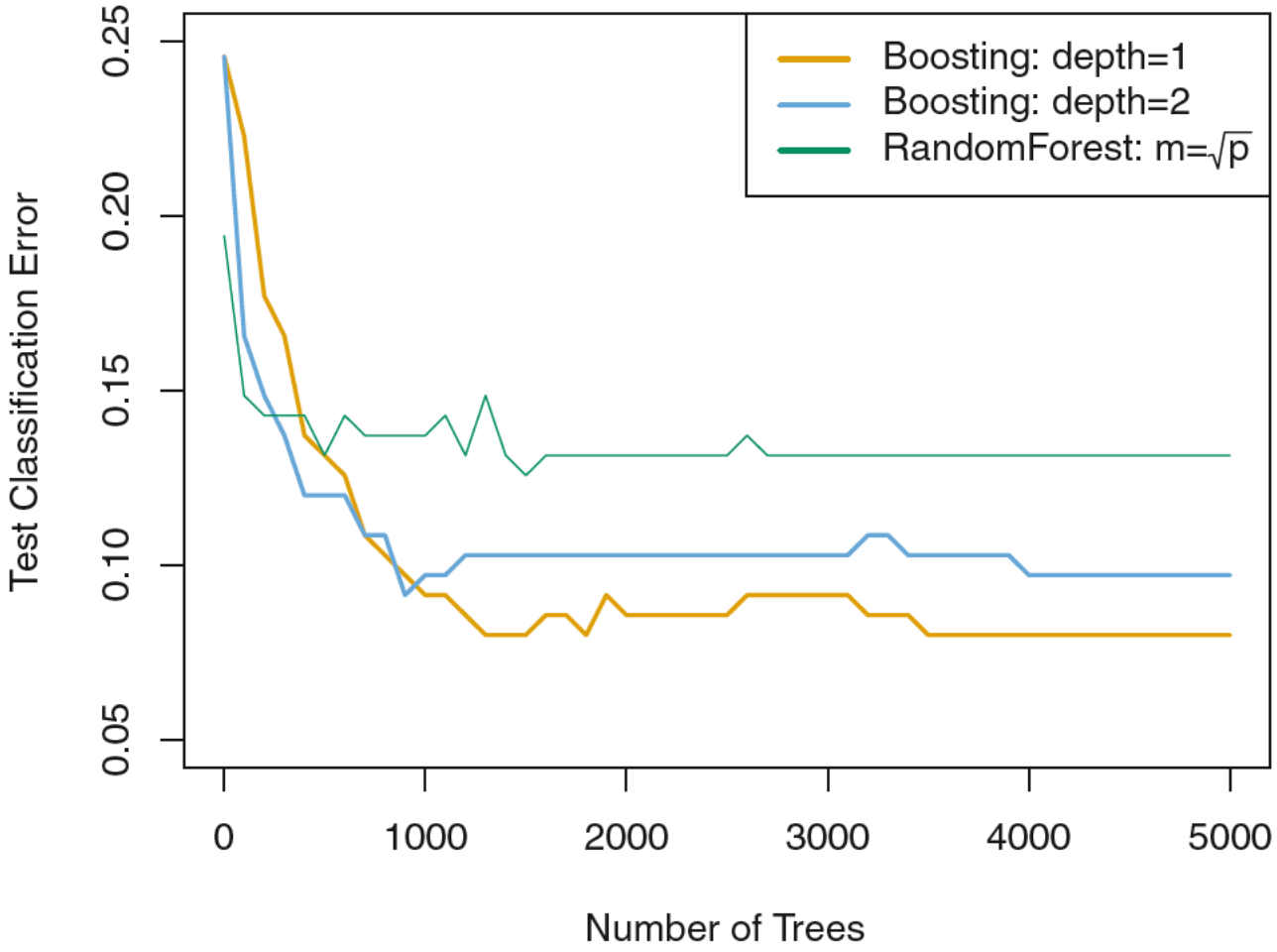
Tuning Parameters in Boosting

1. 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.

2. The shrinkage parameter λ
   - λ is a small positive number that controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice may 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 of splits in each tree d
   - d controls the complexity of the boosted ensemble. Often $d = 1 \text{ or } 2$ works well. In the case $d = 1$, each tree consists only one splitting and boosting 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.

Example: Random Forest versus Boosting



- Now we apply boosting to the cancer classification dataset. We try depth $d = 1$ and 2 respectively. For both boosting models, we set $\lambda = 0.01$.
- Boosting methods require a much larger number of trees to achieve a reasonable testing error rate.
- Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forests, although the standard errors are around 0.02, making none of these differences significant.