



NYC DATA SCIENCE
ACADEMY

Trees, Bagging, Random Forests, & Boosting

Data Science Bootcamp

Outline

- ❖ **Part 1: Decision Trees**
- ❖ **Part 2: Bagging**
- ❖ **Part 3: Random Forests**
- ❖ **Part 4: Boosting**
- ❖ **Part 5: Variable Importance**

PART 1

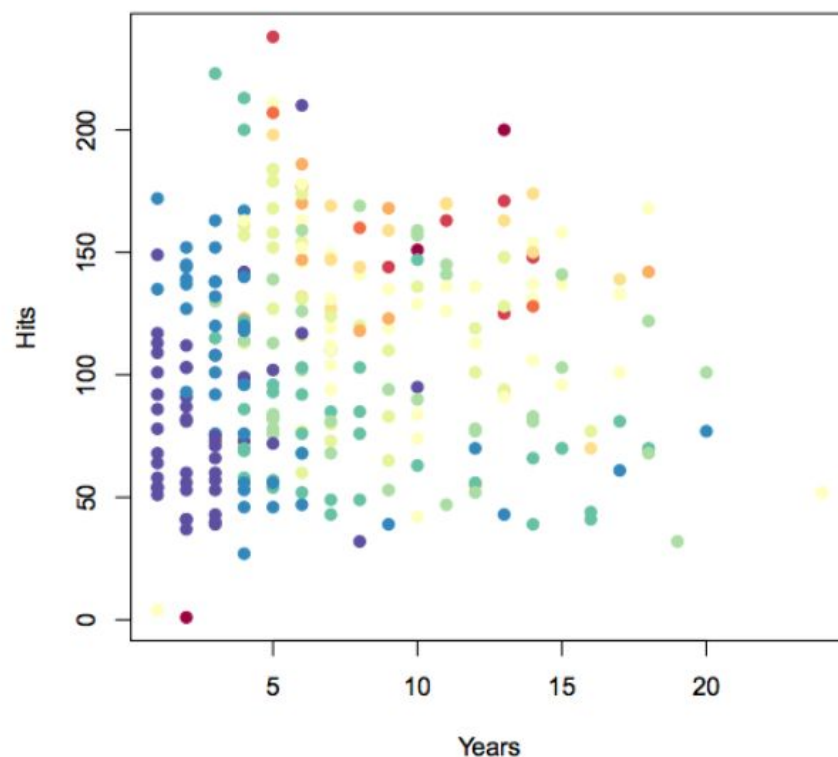
Decision Trees

What are Tree-Based Methods?

- ❖ Tree-based methods are **supervised** learning procedures that aid in both regression and classification settings.
- ❖ The models aim to construct solutions that **stratify the feature space** into relatively easy to describe regions.
- ❖ Essentially, if we can get an idea of the general characteristics of observations that fall within particular regions of space, we can **inform the characteristics** of new observations that fall within the same regions.

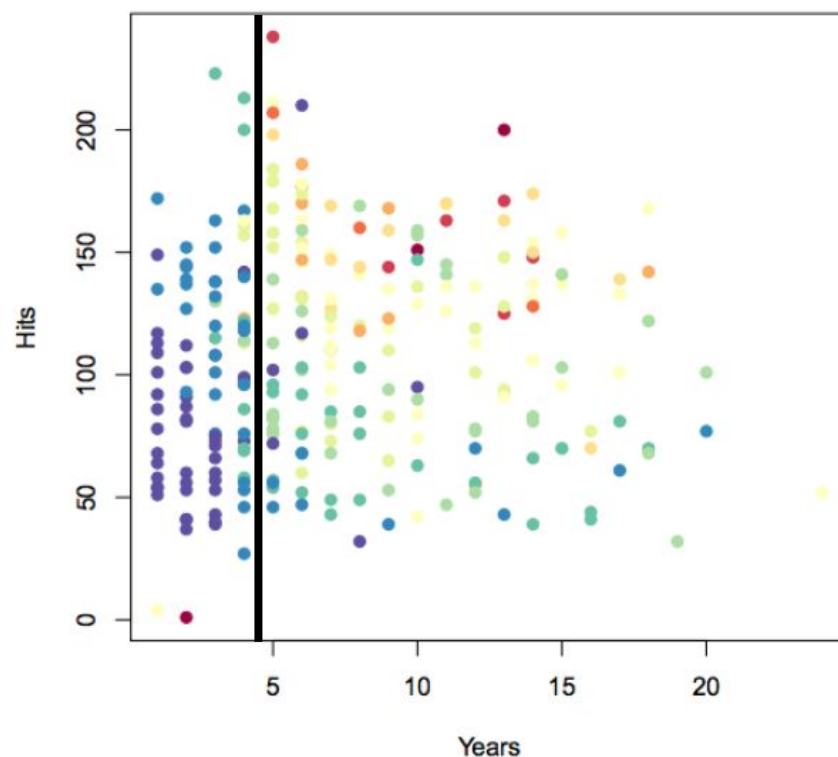
Regression Trees: Visually

- ❖ Let's return to the baseball player salary data. How might we **segment** the following graph into regions?
 - Color corresponds to salary: red/orange = **high**, blue/violet = **low**.



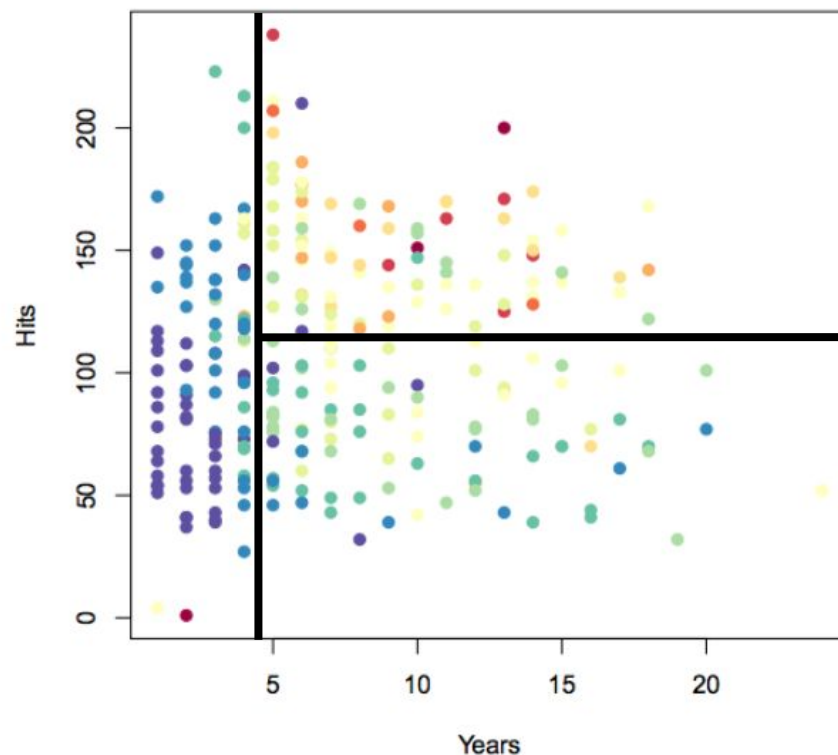
Regression Trees: Visually

- ❖ Let's return to the baseball player salary data. How might we **segment** the following graph into regions?
 - Color corresponds to salary: red/orange = **high**, blue/violet = **low**.



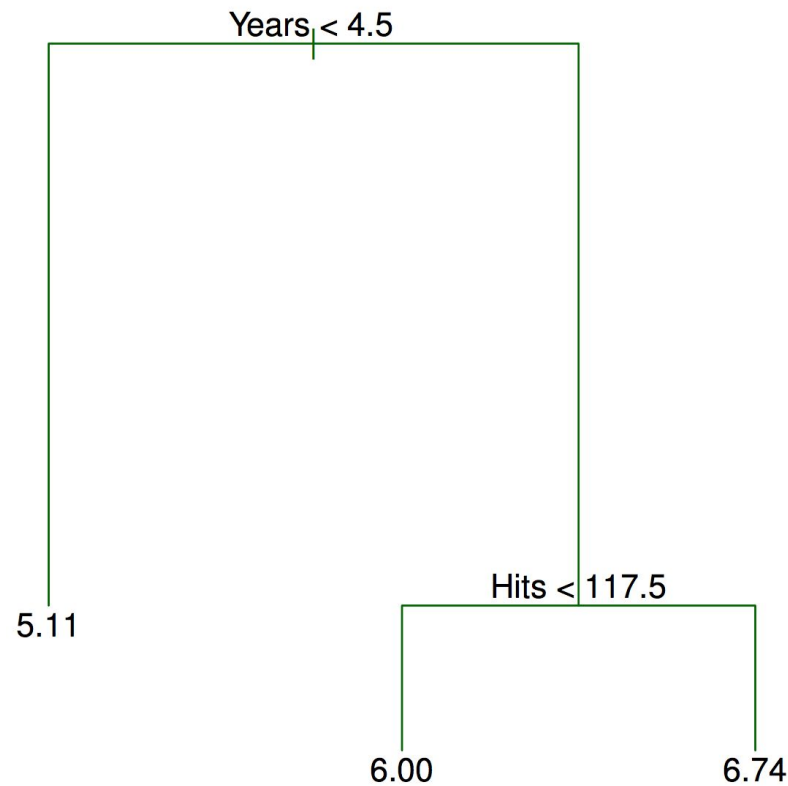
Regression Trees: Visually

- ❖ Let's return to the baseball player salary data. How might we **segment** the following graph into regions?
 - Color corresponds to salary: red/orange = **high**, blue/violet = **low**.



Regression Trees: Visually

- ❖ The **decision tree** that corresponds to cuts similar to what we've created is given below:

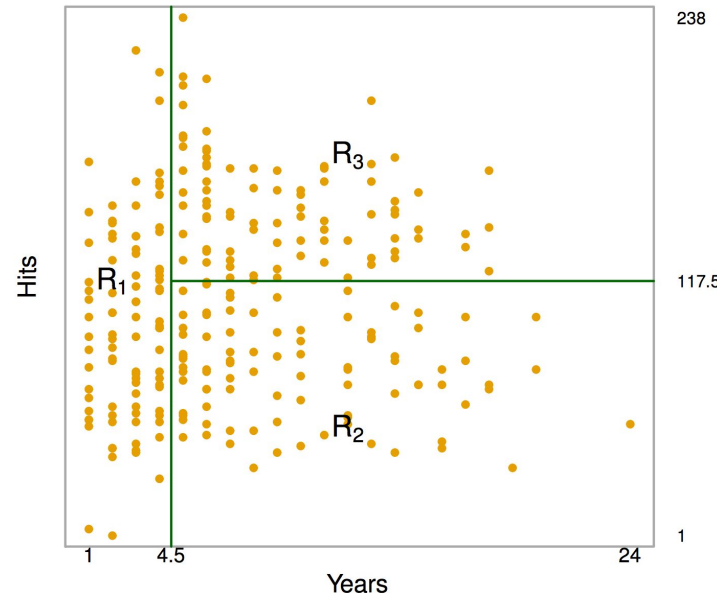


How to Interpret a Decision Tree

- ❖ What is the decision tree actually saying? How do we interpret this tree?
 - We start from the top of the tree and pass a new observation through the various **internal nodes**.
 - At each internal node, we make a **decision** of how to proceed based on the characteristics of the observation at hand.
 - If the listed condition is **satisfied**, we move down the **left** branch.
 - If the listed condition is **not satisfied**, we move down the **right** branch.
 - Continue this process until you reach a **terminal node/leaf** (i.e., a node for which there is no further decision to be made).
 - The number within the terminal node is the **mean response value** for the observations that fell within that region; this is also the prediction for future observations that fall within that region.

How to Interpret a Decision Tree

- ❖ The decision tree ended up **partitioning the feature space** into three regions:
 - R_1 : Where players have less than 4.5 years of experience.
 - R_2 : Where players have greater than 4.5 years of experience and had fewer than 117.5 hits.
 - R_3 : Where players have greater than 4.5 years of experience and had greater than 117.5 hits.



Regression Trees: Mathematically

- ❖ The process of building a regression tree can be simplified into the following two steps:
 1. Segment the predictor space (all possible values of X_1, X_2, \dots, X_p) into J **distinct and non-overlapping regions**. Call these regions R_1, R_2, \dots, R_J .
 2. For every observation that falls into a specific region R_j , predict the **mean of the response values** for the training observations that fell within R_j .

- ❖ This is a bit of an over-simplification:
 - How do we decide where exactly to segment the predictor space?
 - How do we come upon the regions R_1, R_2, \dots, R_J ?

Regression Trees: Mathematically

- ❖ Theoretically, it is possible that regions in the feature space could have any shape; however, if the region splits were to not follow some specific pattern, it would be difficult to represent the resulting model by a decision tree.
 - For ease of interpretation, **rectangular/box-like segments** will allow us to construct a decision tree for our predictive model.
- ❖ The goal now becomes much simpler; find rectangular boxes R_1, R_2, \dots, R_J such that the RSS is minimized, given by:

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

- ❖ We aim to minimize the squared differences of the response as compared to the mean response for the training observations within the j^{th} region.

Regression Trees: Mathematically

- ❖ While the goal itself seems straightforward, it is **computationally infeasible** to consider every possible segmentation of the feature space into J regions.
 - The minimization problem isn't as easily solvable as we have seen with other machine learning methods, especially as the number of regions increases.
- ❖ Tree-based methods provide an approximation by implementing both a top-down and a greedy approach called **recursive binary splitting**:
 - The method is **top-down** because the feature space is split into binary components in a successive fashion, creating new branches of the tree to potentially be split themselves.
 - The method is **greedy** because splits are made at each step of the process based on the best result possible at that given step.
 - The splits are not made based on what might eventually lead to a better segmentation in future steps.

Recursive Binary Splitting: Mathematically

- ❖ The recursive binary splitting process depends on the **greatest reduction in the RSS** based on the predictor X_j and the cutpoint s that end up partitioning the space into the regions:

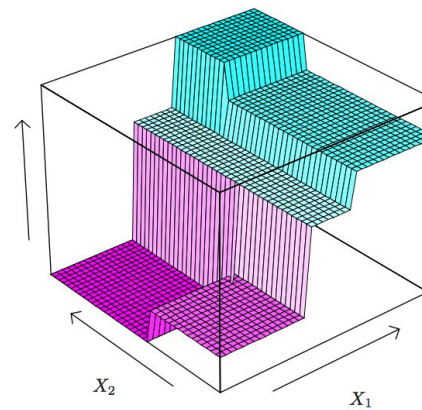
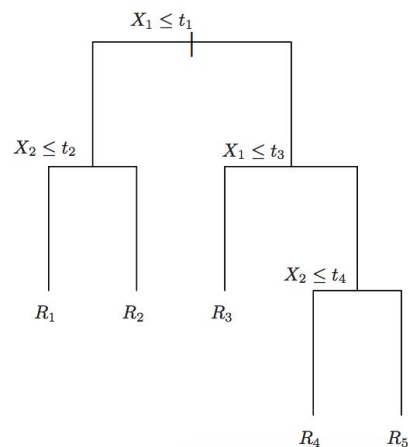
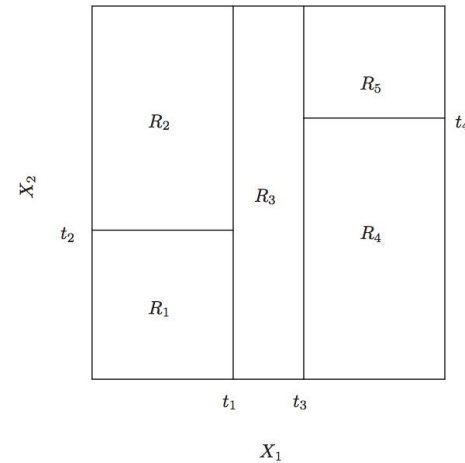
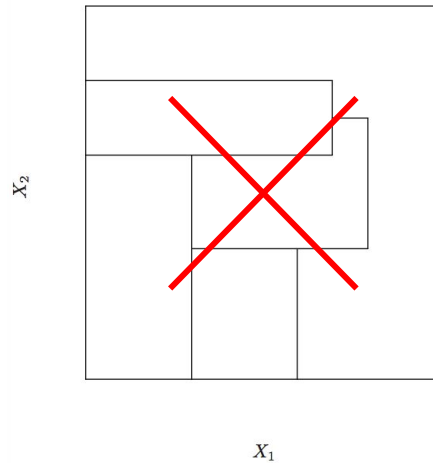
- $R_1(j, s) = \{X \mid X_j < s\}$
- $R_2(j, s) = \{X \mid X_j \geq s\}$

- ❖ In other words, the splitting process seeks the values of j and s that will end up **minimizing** the following:

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

- ❖ This process is repeated by considering each of the newly created regions as the new overall feature spaces to segment.

Recursive Binary Splitting: Visually



When do we Stop Splitting?

- ❖ The recursive binary splitting process as described above is likely to induce **overfitting**, thus leading to **poor predictive performance** on new observations.
 - Consider the extreme case in which each observation has its own terminal node (this model will have high variance). The RSS will be exactly 0 on the training set.
- ❖ What if we try fitting a tree with fewer regions? Theoretically, this should lead to lower variance with a cost of some bias, but ultimately lead to **better prediction**.
 - Grow the tree to a certain extent until the reduction in the RSS at a split doesn't surpass a certain threshold.
 - **Problem**: Although a split might not be incredibly valuable in reducing the RSS early on in a tree, it might lead to a future split that does reduce the RSS to a large extent. What is one possible **solution**?

Tree Pruning

- ❖ We've seen that deciding on the number of splits **prior** to building a tree isn't the best strategy.
 - What if we first built a tree that is very large and then **pruned** it back in order to obtain a suitable **subtree**?
- ❖ The best subtree will be the one that yields the **lowest test error** rate.
 - Given a subtree, we can estimate the test error by implementing the **cross-validation** process, but this is too cumbersome because the large number of possible subtrees. **We still need a better process!**
- ❖ In practice, rather than checking every single possible subtree, the process of **cost complexity pruning** (i.e., **weakest link pruning**) allows us to select a smaller set of subtrees for consideration.

Cost Complexity Tree Pruning

- ❖ Consider a sequence of trees indexed by a non-negative tuning parameter α . For each value of α there corresponds a subtree T such that the following is criterion minimized:

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

- ❖ Where:
 - $|T|$ indicates the total number of **terminal nodes** of subtree T .
 - R_m is the **subset region** of the feature space corresponding to the m^{th} terminal node.

Cost Complexity Tree Pruning

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

- ❖ The tuning parameter α helps **balance the tradeoff** between the overall complexity of the tree and its fit to the training data:
 - Small values of α yield trees that are quite **extensive** (have many terminal nodes).
 - Large values of α yield trees that are quite **limited** (have few terminal nodes).
- ❖ This process is very similar to the **shrinkage/regularization method** utilized in ridge and lasso regression!

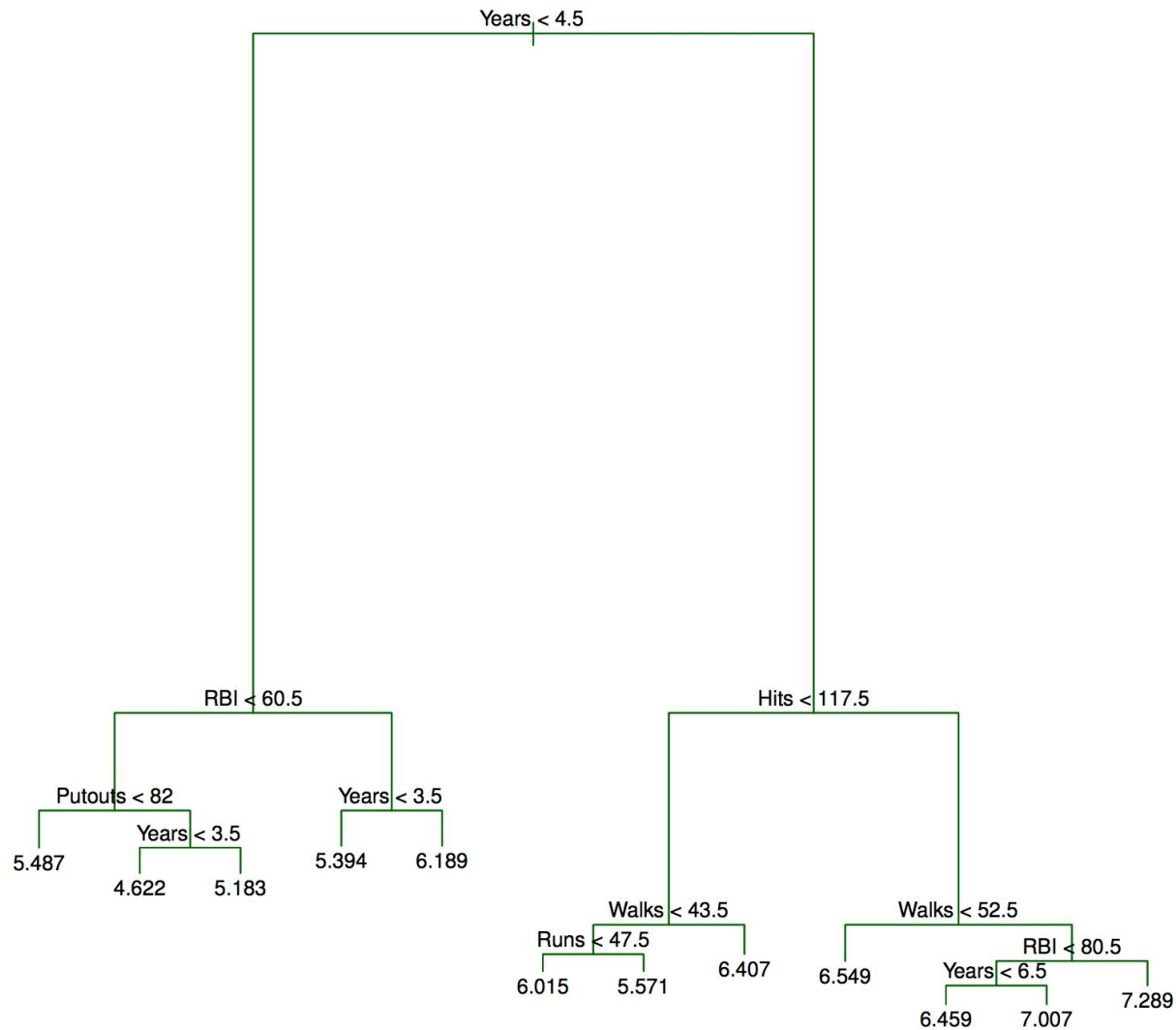
Cost Complexity Tree Pruning

- ❖ It can be shown that as the value of the tuning parameter α increases, branches of the overall tree are pruned in a **nested** manner.
 - Thus, it is possible to obtain a sequence of subtrees as a function of α .
- ❖ As with any other tuning parameter, in order to select the optimal value of α we implement **cross-validation**.
- ❖ Ultimately, the subtree that is used for prediction is built using **all of the available data** with the determined optimal value of α .

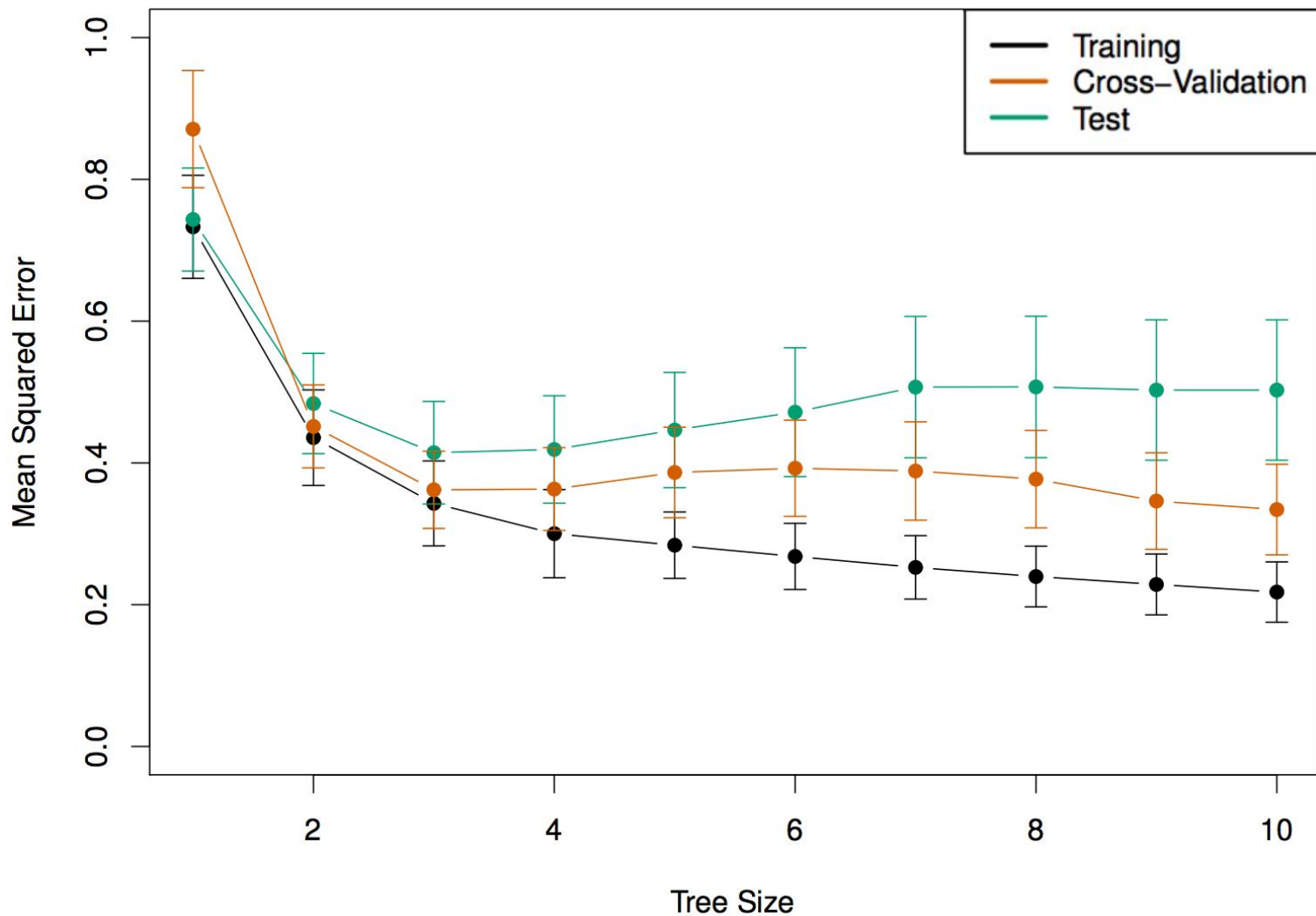
Building a Regression Tree

- ❖ Incorporating these ideas, the algorithm for building a regression tree is as follows:
 1. Use **recursive binary splitting** to build a large tree on the training data; stop before each observation falls into its own leaf (e.g., when each terminal node has fewer than 5 observations, etc.).
 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 the best α .
 - a. For each of the K folds:
 - i. Repeat steps 1 and 2 on all but the k^{th} fold of the training data.
 - ii. Evaluate the mean squared prediction error on the data in the left-out k^{th} fold as a function of α .
 - b. Average the errors for each α ; **select the α** that minimizes this criterion.
 4. Return the subtree of the overall tree from step 2 that corresponds to the best α .

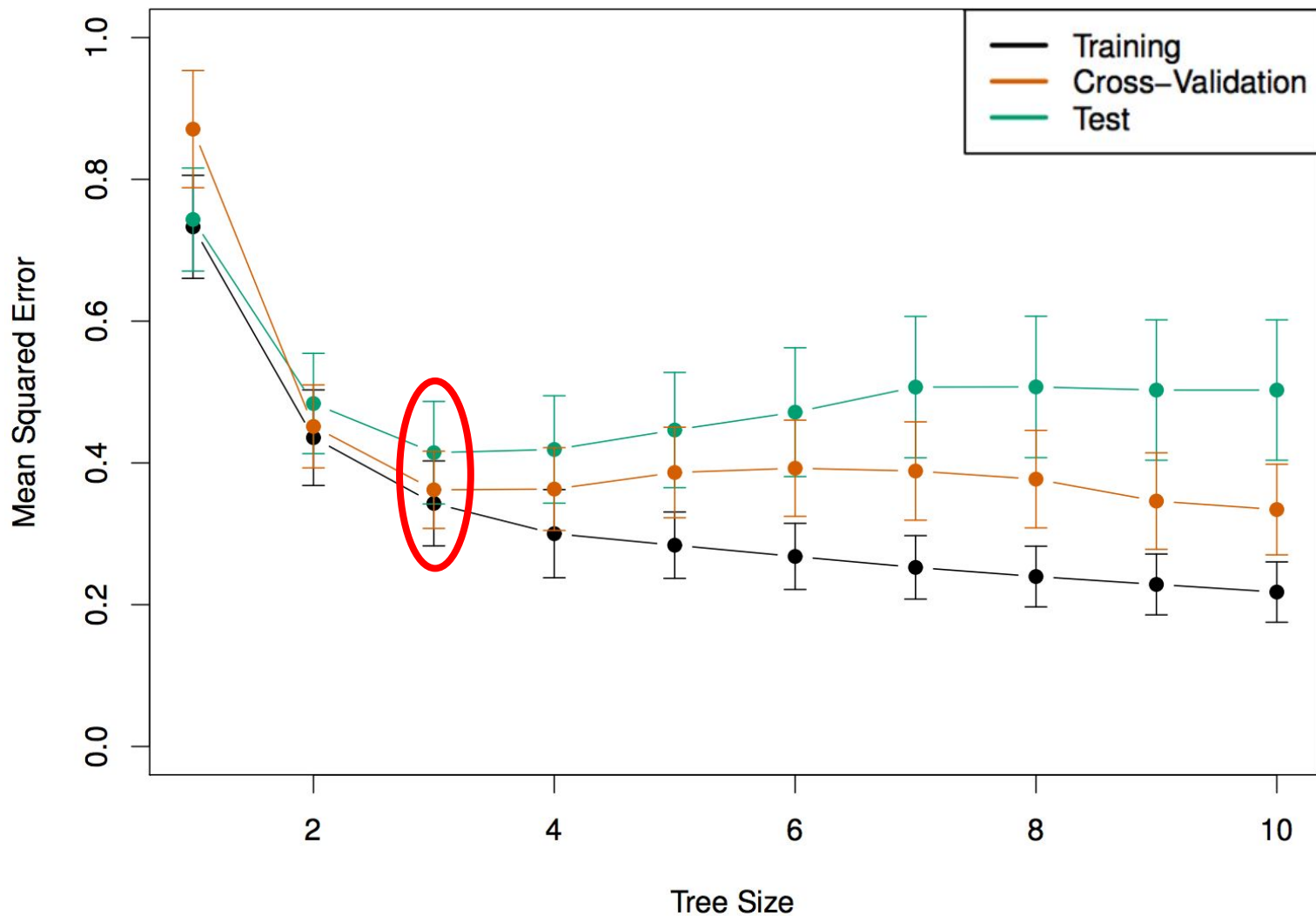
Building a Regression Tree



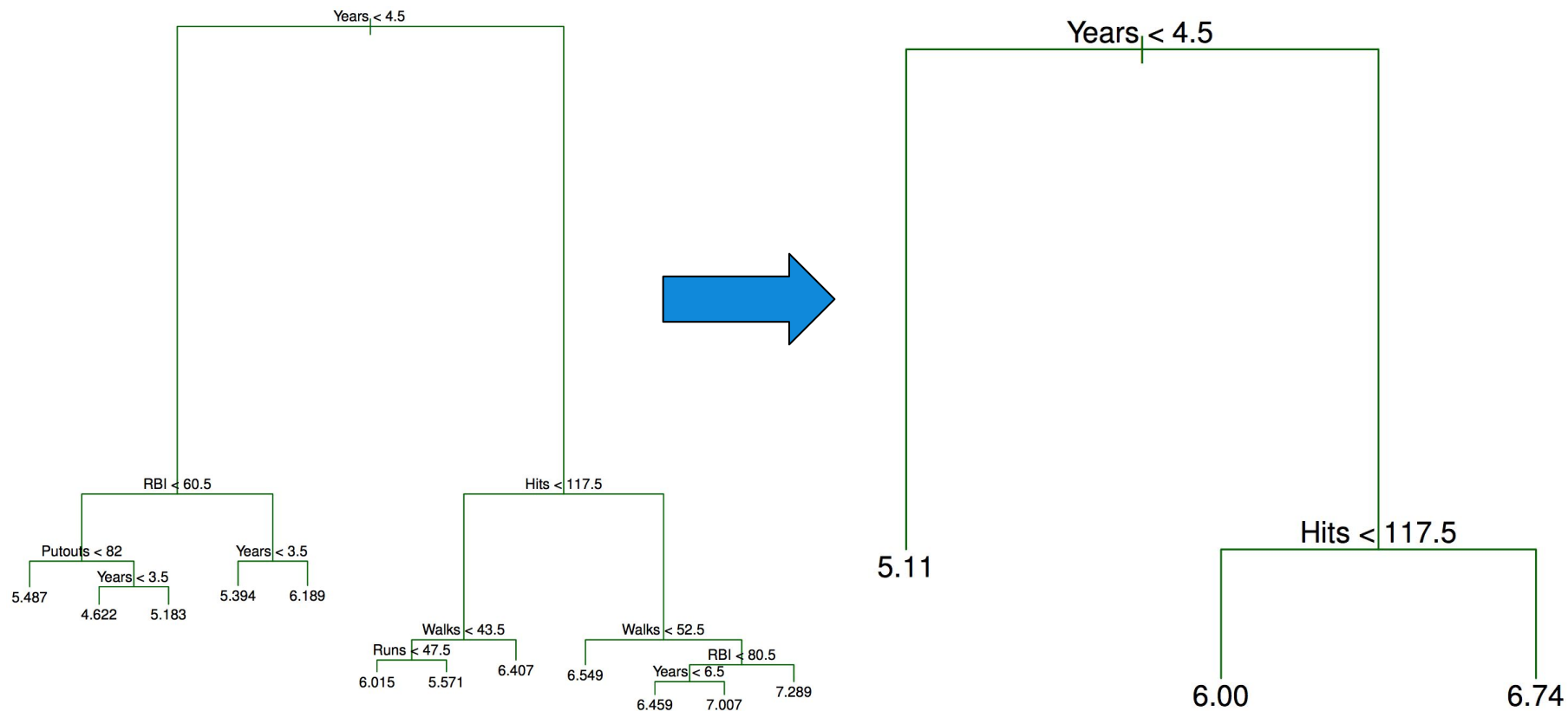
Building a Regression Tree



Building a Regression Tree



Building a Regression Tree



Classification Trees

- ❖ **Classification trees** are very similar to regression trees; they are the analogous output of the aforementioned process when we try to predict a qualitative (categorical) response rather than a quantitative (numerical) response.
- ❖ For each of the subregions created, we predict that an observation belongs to the **most commonly occurring class** of training observations in its associated region.
- ❖ In this setting, we still implement the process of **recursive binary splitting** to create various subregions of our feature space.
 - Unfortunately, the **RSS no longer makes sense** for the categorical setting. What criterion can we use instead?

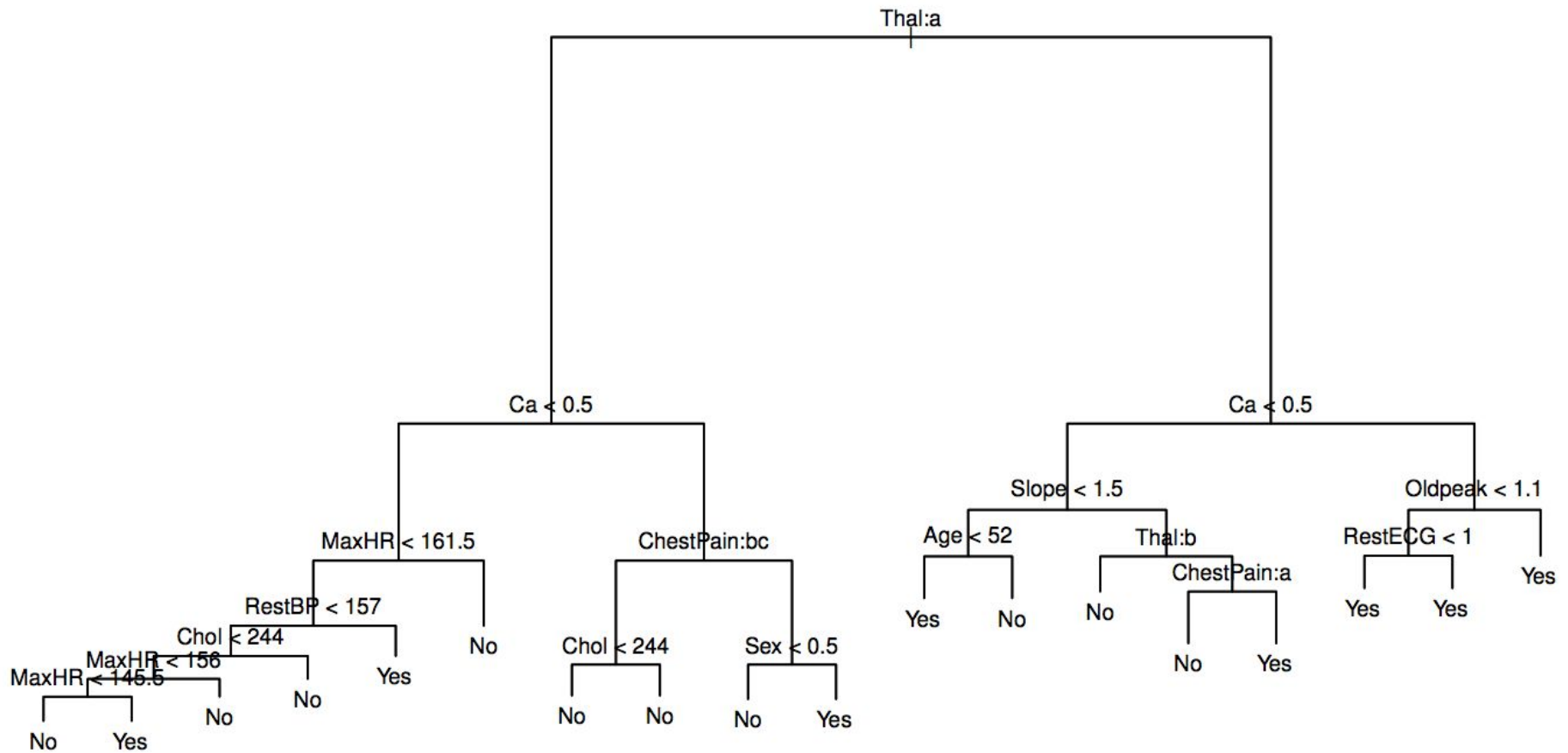
Classification Trees: The Gini Index

- ❖ A simple alternative is the **misclassification rate** (i.e., the fraction of training observations in a region that do not belong to the most common class).
 - It turns out that the misclassification rate can end up not being sufficiently sensitive; it's **too choppy** doesn't lead to a smooth tree building process.
- ❖ The typical criterion used instead is called the **Gini index**, defined as:

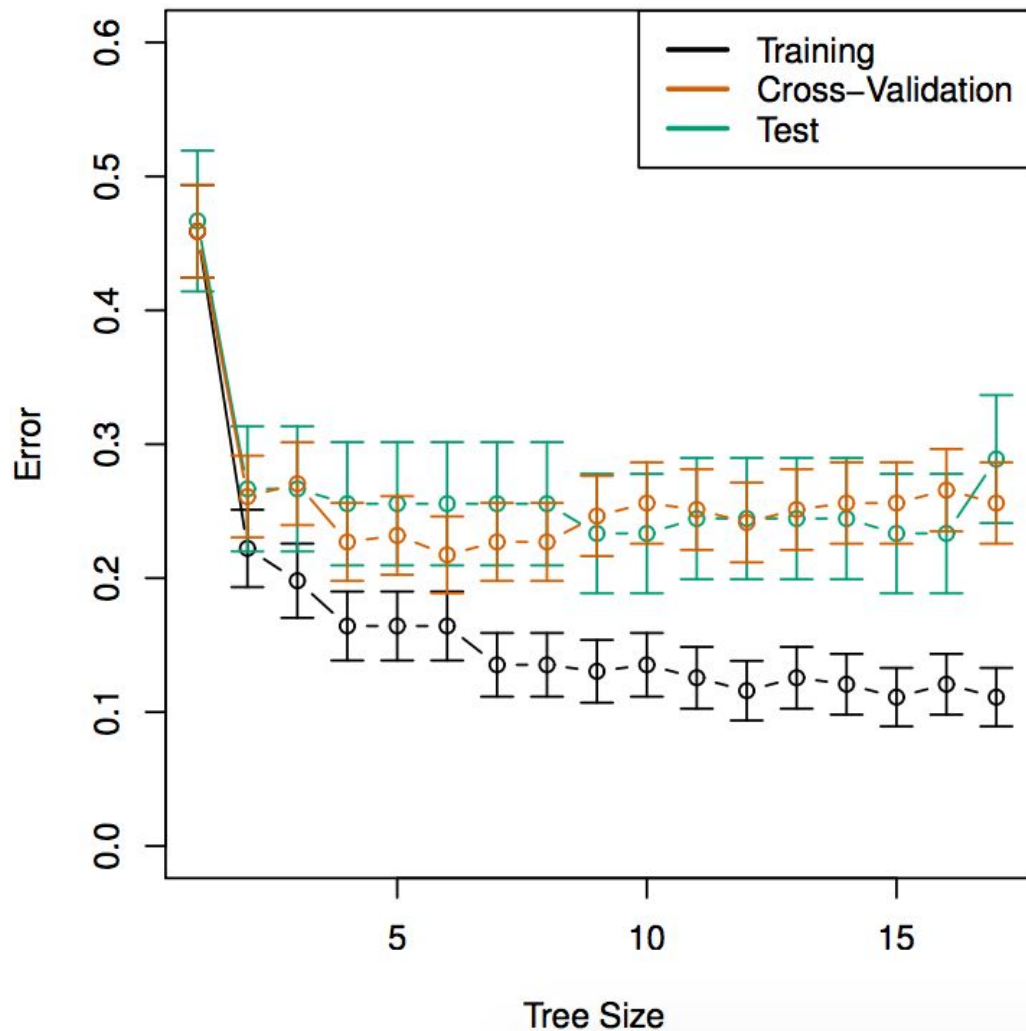
$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- Here, the proportions of interest denote the fraction of training observations in the m^{th} region that are from the k^{th} class.
- The index measures the **total variance** among the K classes; it is often referenced as a measure of terminal node **purity**.

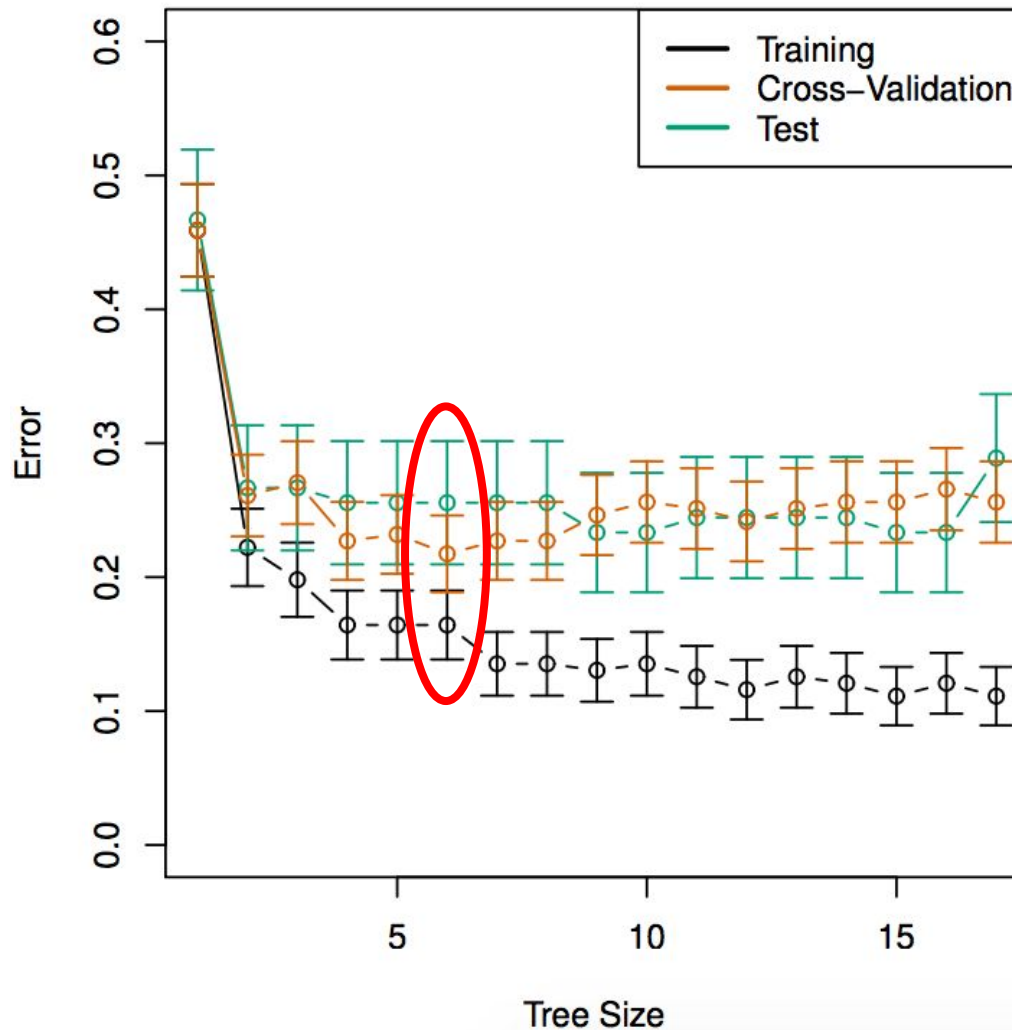
Classification Trees



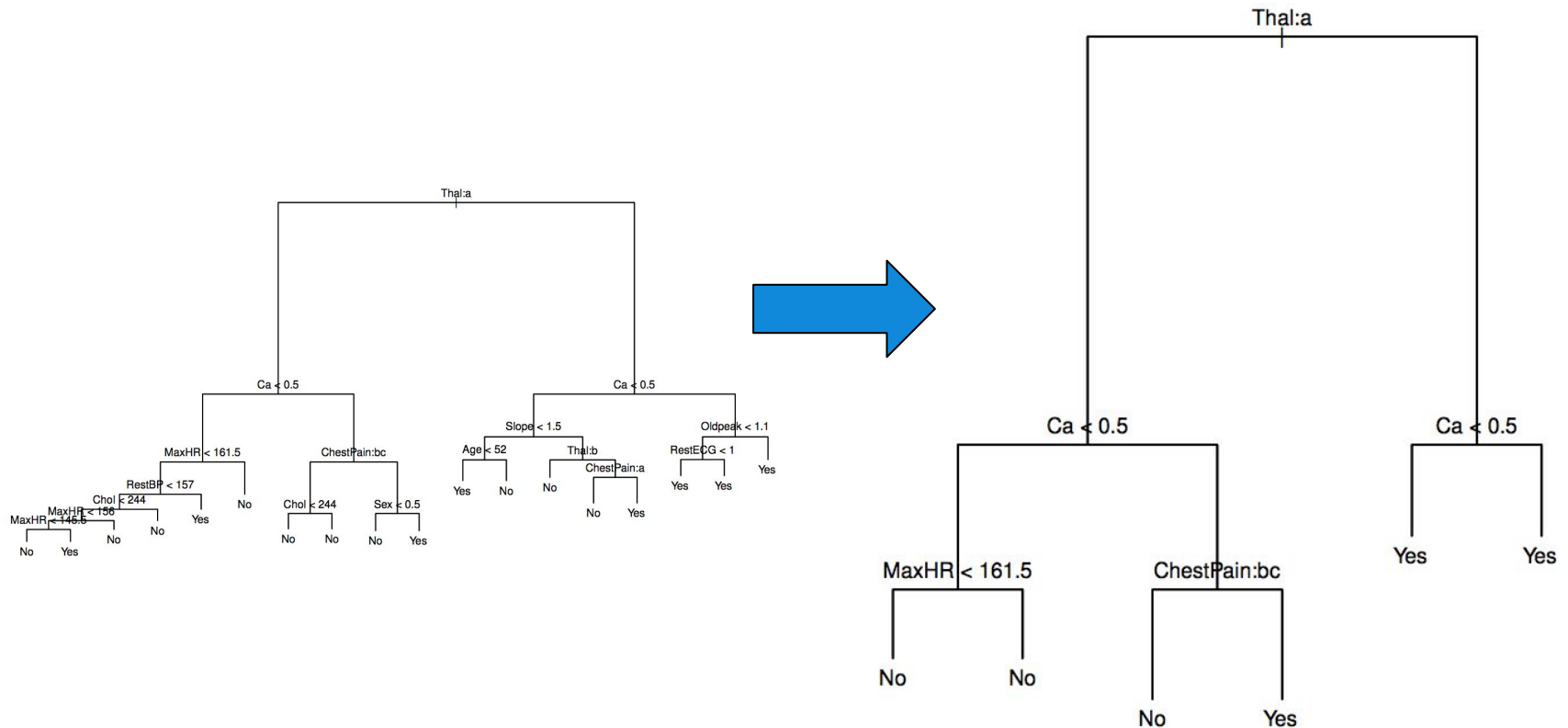
Classification Trees



Classification Trees



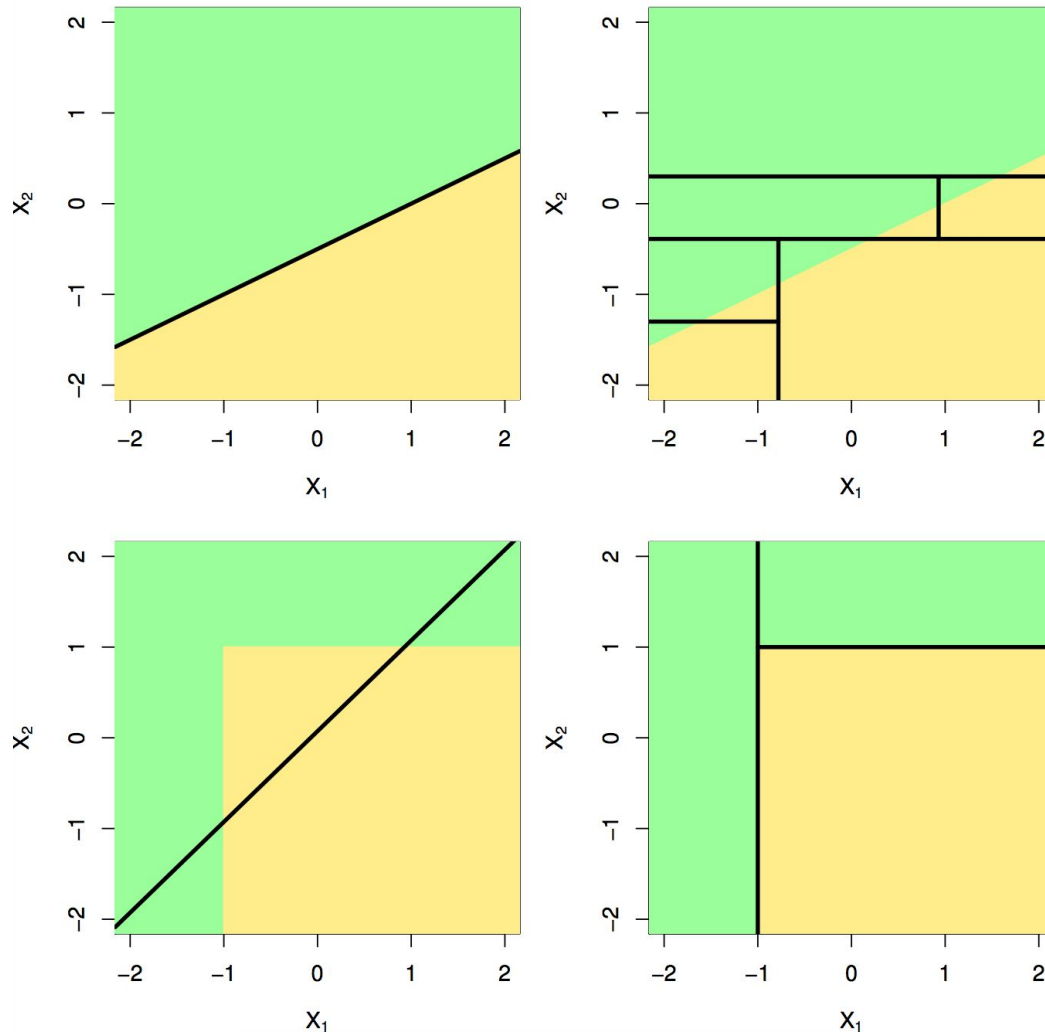
Classification Trees



Classification Trees

- ❖ It is interesting to note that some splits yield terminal nodes that have the **same predicted value**.
 - Why was this split performed at all in the first place?
- ❖ Such duplicate splits are recorded by the classification tree because they lead to **increased node purity**.
 - Having an increased sense of node purity yields an **increased sense of certainty** pertaining to the response value corresponding to each terminal node.

Advantages & Disadvantages of Decision Trees



Advantages & Disadvantages of Decision Trees

- ❖ Without a heavy mathematical background, it is **easy to interpret** a decision tree (especially if it is small); linear regression requires the understanding of an equation.
- ❖ Decision trees can graphically **depict a higher dimensionality** easier than linear regression and still be interpreted by a novice.
- ❖ The process can easily adapt to **qualitative predictors** without the need to create and interpret dummy variables.
- ❖ Decision trees are often believed to reflect a more **“human” decision-making** process as compared to other machine learning methods.

Advantages & Disadvantages of Decision Trees

- ❖ While relatively non-complex among other supervised learning procedures, as a trade-off their **predictive accuracy** tends to be lower and thus not as competitive.
- ❖ What if we could combine the benefits of multiple trees in order to yield an overall prediction? Taking the penalty of **decreased interpretative value**, could this potentially **increase our predictive accuracy**?
 - Bagging
 - Random Forests
 - Boosting

PART 2

Bagging

Variance & Bootstrap Aggregation

- ❖ One of the biggest drawbacks of decision trees is that they **suffer from high variance**:
 - If we randomly split our data into two different parts and fit independent decision trees, the results are likely to be **quite different**.
- ❖ **Bootstrap aggregation** (i.e., **bagging**) is a procedure that aids in the reduction of variance for a statistical learning method; it is frequently used alongside trees.
 - Recall that given a set of n independent observations X_1, X_2, \dots, X_n , each themselves drawn from a distribution with variance σ^2 , the mean of these observations as a group would be given by σ^2/n .
 - Averaging the set of observations **reduces the overall variance**.
- ❖ Why is this **not practical** in the general sense? We typically do not have access to multiple training sets.

Bagging by Bootstrapping

- ❖ What if we could create multiple pseudo-training sets? We can do so by **bootstrapping**:
 - Take **repeated samples** of the same size from the single overall training dataset. Treat these different sets of data as pseudo-training sets.
- ❖ By bootstrapping, we create B different training datasets. The method is trained on the b^{th} bootstrapped training set in order to get predictions for each observation. We can then **average all predictions** (or take the **majority vote**) to obtain the bagged estimate:

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

- ❖ Why is this a clever idea for trees?

Bagging by Bootstrapping

- ❖ Recall that previously we tried to reduce variance by the process of **pruning** our large trees.
 - While pruning reduces the variance of the overall tree model upon repeated builds with different datasets, we **induce bias** because the trees are much simpler.
- ❖ The idea of **bagging** averts the pruning methodology but still gets its benefits:
 - Instead of pruning back our trees, create very large trees in the first place. These large trees will tend to have **low bias**, but **high variance**.
 - Retain the low bias, but get rid of the high variance by **averaging** across many trees.

Out of Bag Error Estimation

- ❖ How do we **estimate the test error** of a bagged model?
- ❖ Recall that in the bagging process, decision trees are fit to bootstrapped subsets of the overall available observations.
 - Observations that are **used to fit the tree** are said to be “**in the bag.**”
 - Observations that are **not used to fit the tree** are said to be “**out of bag.**”
- ❖ We can first predict the response for a given observation using each of the trees in which that observation was out of bag, and then **average the results**.
 - The averaged predictions are used to calculate the **out of bag error estimate**.
 - When the number of bootstrapped samples is large, this is essentially the same as **leave-one-out cross-validation** error for bagging.

PART 3

Random Forests

Benefits of Random Forests

- ❖ It can be shown that the variance of the mean of a sample increases as observations are correlated with one another.
 - In other words, correlated observations are **not as effective** at reducing the uncertainty of the mean as uncorrelated, independent observations.
- ❖ If we could generate trees that are **not correlated** with one another, we could improve upon the bagging procedure.
- ❖ **Random forests** help us by decorrelating our trees.
 - As expected, this results in a **reduction in variance** once we average the trees.

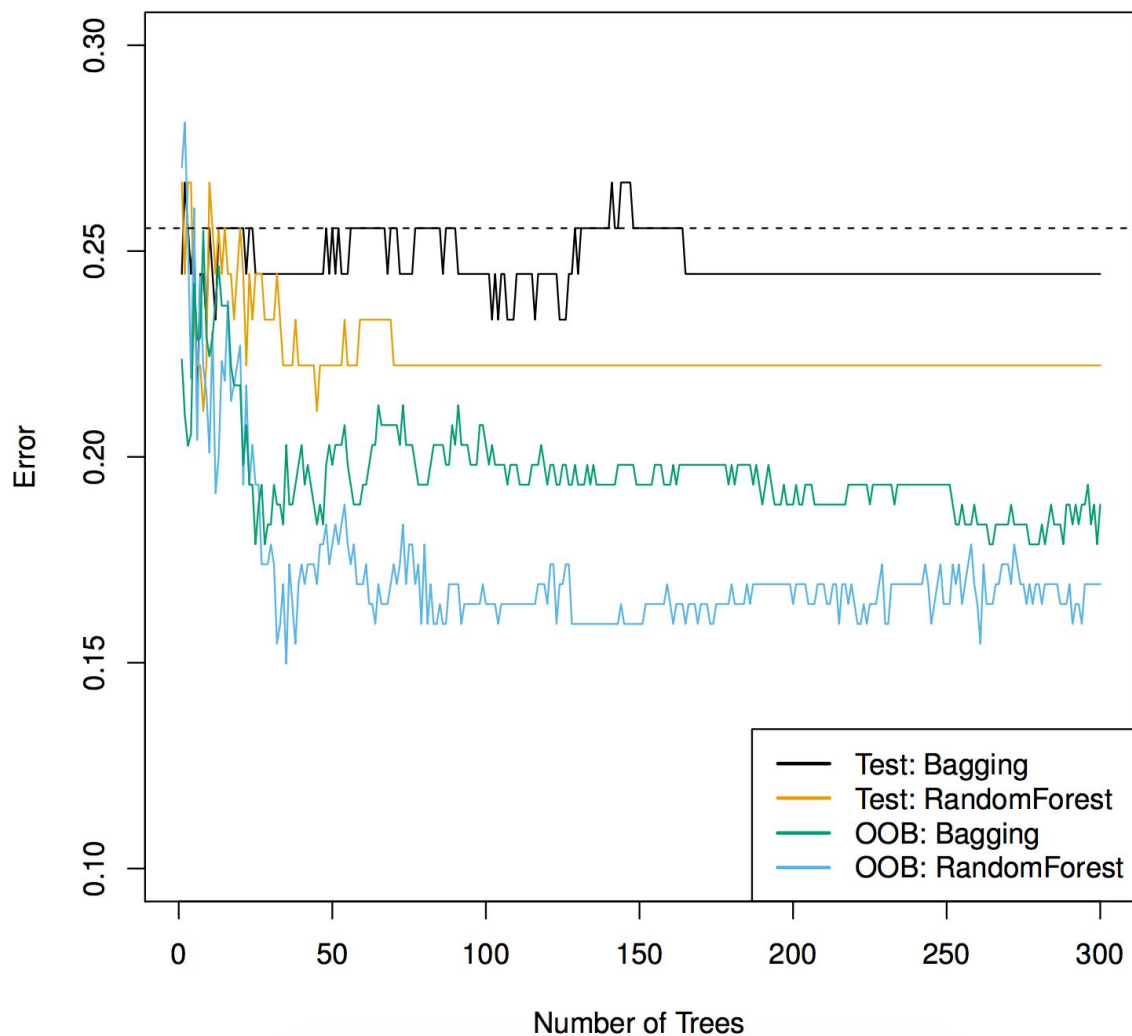
The Random Forest Procedure

- ❖ Similar to bagging, we first build various decision trees on bootstrapped training samples, but we split internal nodes in a special way.
- ❖ Each time a split is considered within the construction of a decision tree, only a **random subset of m** of the overall p predictors are allowed to be candidates.
 - In other words, **only the m predictors** have the possibility to be chosen as the splitting factor.
- ❖ At every split, a **new subset** of predictors is randomly selected.
 - Typically, $m \approx \sqrt{p}$ is a sufficient rule for subset selection.
 - What happens if we choose $m = p$?

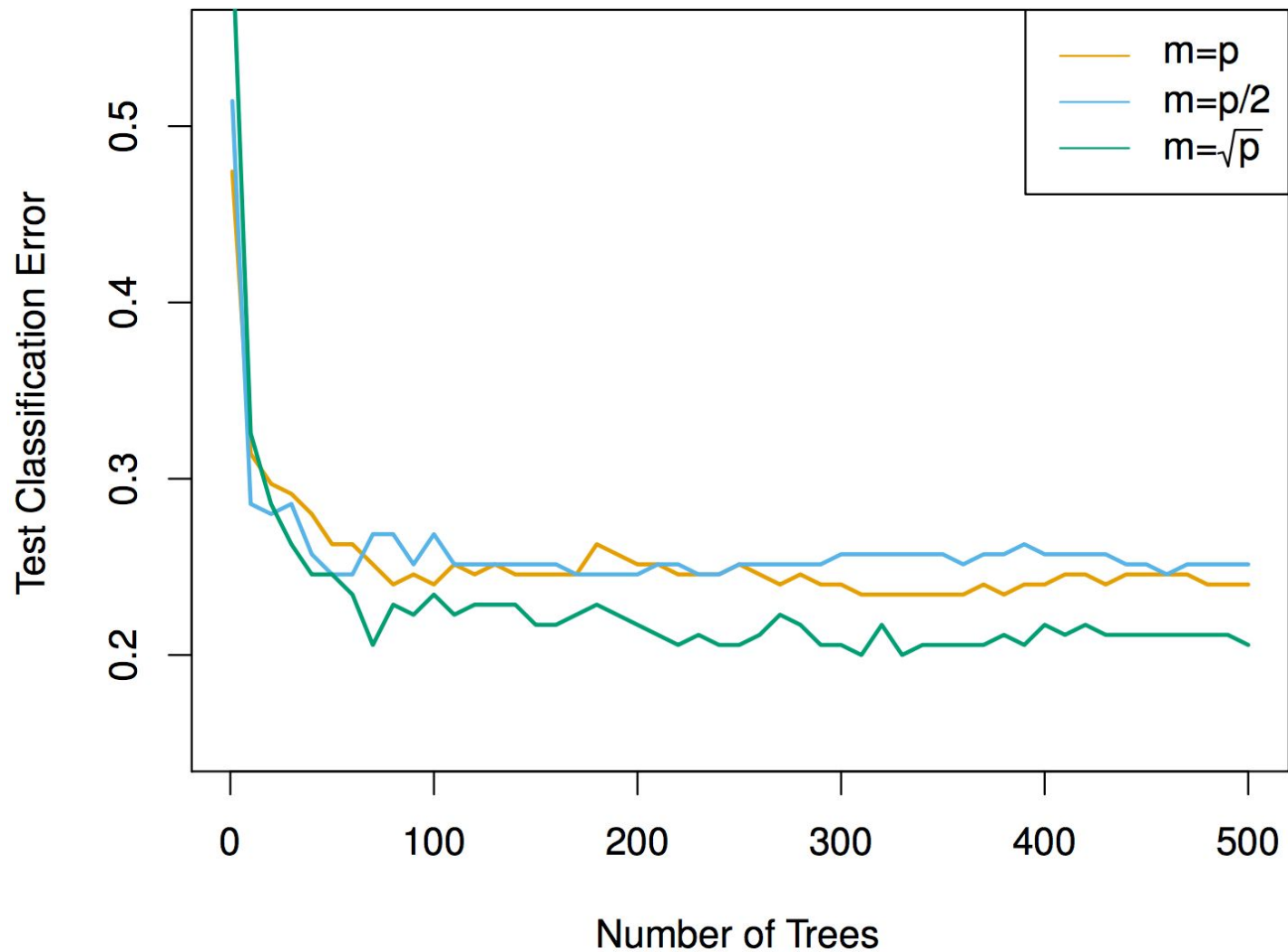
The Random Forest Procedure

- ❖ Although it seems crazy to throw away most predictor variables at each of the splits, why does this end up helping in the long run?
 - The random forest procedure forces the decision tree building process to **use different predictors** to split at different times.
- ❖ Should a good predictor be left out of consideration for some splits, it still has **many chances to be considered** in the construction of other splits.
 - The same idea goes for predictors surfacing in trees as a whole.
- ❖ Even if we used the same training sample, using the random forest procedure would likely yield **different trees**.
- ❖ We **can't overfit** by adding more trees! The variance just ends up decreasing!

Bagging & Random Forests



Bagging & Random Forests



PART 4

Boosting

Boosting Decision Trees

- ❖ Recall that the **bagging** procedure involves:
 - Creating multiple pseudo-training data sets using **bootstrapped sampling**.
 - Fitting separate, **independent decision trees** to each of the bootstrapped training data sets.
 - Combining all of the separate trees by **averaging** in order to create a single predictive model.
- ❖ The **boosting** procedure works in a similar way, except that the decision trees are generated in a **sequential** manner:
 - Each tree is generated **using information from previously grown trees**; the addition of a new tree improves upon the performance of the previous trees.
 - The trees are now **dependent** upon one another.

The Boosting Algorithm

❖ The boosting algorithm is as follows:

1. For each i in the training data, set: $\hat{f}(x) = 0, r_i = y_i$
2. For $b = 1, 2, \dots, B$:
 - a. Fit a tree with d splits ($d + 1$ terminal nodes) to the training data (X, r) .
 - b. Update f by adding in a shrunk 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:

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x)$$

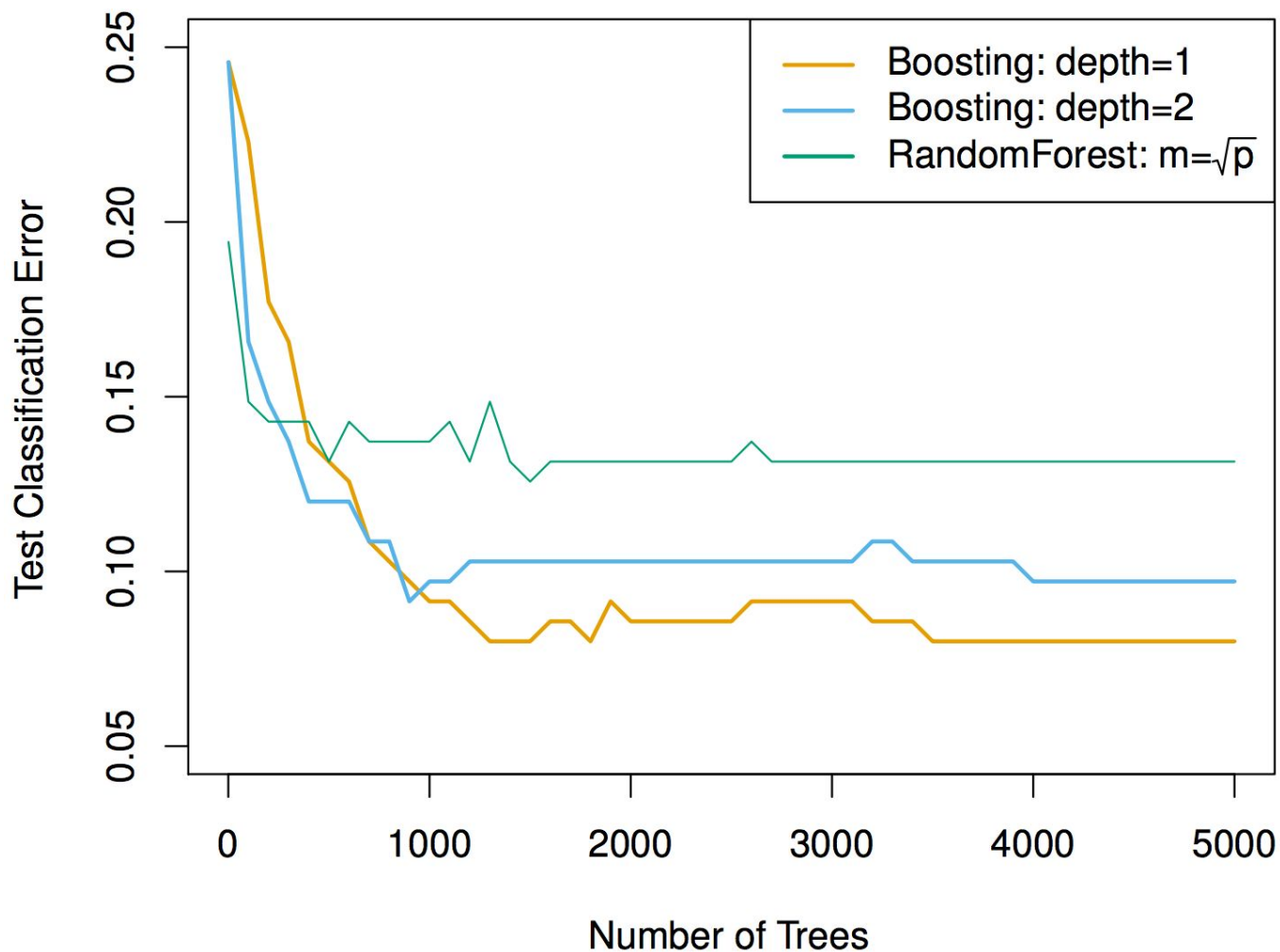
Details of Boosting

- ❖ Whereas creating a single large decision tree can amount to **severe overfitting** to our training data, the boosting approach tends to **slowly learn** our data.
- ❖ Given a current decision tree model, we fit a new decision tree to the **residuals** of the current decision tree.
 - The new decision tree (based on the residuals) is then added to the current decision tree, and the residuals are updated.
- ❖ We limit the number of terminal nodes in order to **sequentially** fit small trees.
 - By fitting small trees to the residuals, we slowly improve the overall model in areas where it does not perform well.
 - The shrinkage parameter λ is taken to be quite small, and slows the process down even further to avoid overfitting.

Tuning Parameters for Boosting

- ❖ Boosting requires three tuning parameters:
 1. The **number of trees B** .
 - a. Unlike in bagging and random forests, boosting can overfit if B is large (although very slowly). Use cross-validation to select B .
 2. The **shrinkage parameter λ** , a small positive number.
 - a. This controls the rate at which boosting learns. Typical values are around 0.01 to 0.001. Note that a very small λ may require using a large value of B .
 3. The **number of splits d** in each tree.
 - a. This controls the complexity of the boosted ensemble. Typically using stumps (single splits where $d = 1$) is sufficient and results in an additive model. The tree depth corresponds to the interaction order of the boosted model since d splits can involve (at most) d distinct variables.

Boosting & Random Forests



PART 5

Variable Importance

Variable Importance

- ❖ For bagged and random forest trees, we can record the total amount that a given criterion is decreased over all splits relevant to a given predictor, averaged over all B trees.
 - For regression trees, we can use the **reduction in the RSS**.
 - For classification trees, we can use the **reduction in the Gini index**.
- ❖ In both the regression and classification setting, we can do this for each predictor of our original dataset.
 - A relatively large value indicates a notable drop in the RSS or Gini index, and thus a **better fit** to the data; corresponding variables are relatively **important predictors**.
- ❖ This allows us to gain a **qualitative** understanding of the variables in our dataset.

PART 5

Review

Review

❖ Part 1: Decision Trees

- What are Tree Based Methods?
- Regression Trees: Visually
- How to Interpret a Decision Tree
- Regression Trees: Mathematically
- Recursive Binary Splitting
 - Mathematically
 - Visually
- When do we Stop Splitting?
- Tree Pruning
- Cost Complexity Tree Pruning
- Building a Regression Tree
- Classification Trees
 - The Gini Index
- Advantages & Disadvantages of Decision Trees

❖ Part 2: Bagging

- Variance & Bootstrap Aggregation
- Bagging by Bootstrapping
- Out of Bag Error Estimation

❖ Part 3: Random Forests

- Benefits of Random Forests
- The Random Forest Procedure
- Bagging & Random Forests

❖ Part 4: Boosting

- Boosting Decision Trees
- The Boosting Algorithm
- Details of Boosting
- Tuning Parameters for Boosting
- Boosting & Random Forests

❖ Part 5: Variable Importance