KAIST

Statistical Machine Learning and Its Applications

Lecture 7: Tree-based Methods

KAIST Mark Mintae Kim

Department of Industrial & Systems Engineering
KAIST

OUTLINE

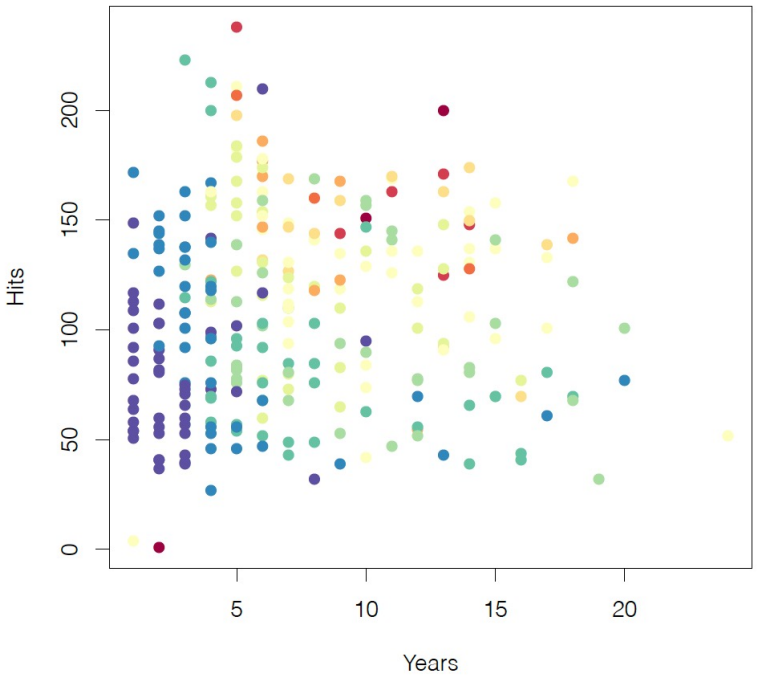
- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees
  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees
- Bagging
- Random Forests
- Boosting

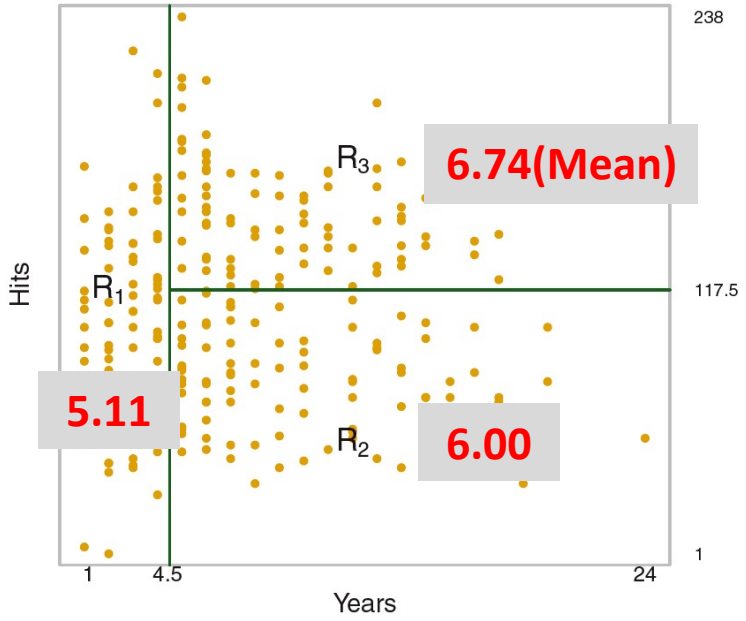
OUTLINE

- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees
  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees
- Bagging
- Random Forests
- Boosting

EXAMPLE: HITTERS DATASET

- Idea: **Segment the predictor space** into a number of simple regions
- **Goal**: Predict a baseball player's *Salary* based on
  - Years (the number of years that the player has played in the major leagues)
  - Hits (the number of hits that the player made in the previous year)
  - Color represents the amount of salary
    - Blue, green → low / Yellow, red → high





Decision-tree

The splitting rules can be summarized in a tree

OVERVIEW

- (+) Tree-based methods are simple and useful for **interpretation**
- (-) However they are typically **not competitive** with the best supervised learning approaches in terms of prediction accuracy

- Hence, we also discuss **bagging**, **random forests**, and **boosting**.
- These methods grow **multiple** trees, and then **combine** them to yield a single consensus prediction

- Combining a large number of trees can often result in dramatic improvements in prediction accuracy,
  - However, at the expense of some loss of interpretability

- Decision trees can be applied to both **regression** and **classification** problems
  - Previous example (Hitters data) was on regression

THE BASICS OF DECISION TREES

- Idea: **Segment the predictor space** into a number of simple regions
- Step 1: Divide the predictor space (i.e. all the possible values for $X_1, X_2, \dots, X_p$) into J distinct regions, $R_1, R_2, \dots, R_J$
- Step 2: Then for every X that falls in a particular region (R_j) we make the same prediction
  - Simply the mean of the response values for the training observation in R_j
- Suppose for example we have two regions R_1 and R_2 with $\hat{y}_{R_1} = 10, \hat{y}_{R_2} = 20$
- Then for any value of X such that $X \in R_1$ we would predict 10 , otherwise if $X \in R_2$ we would predict 20

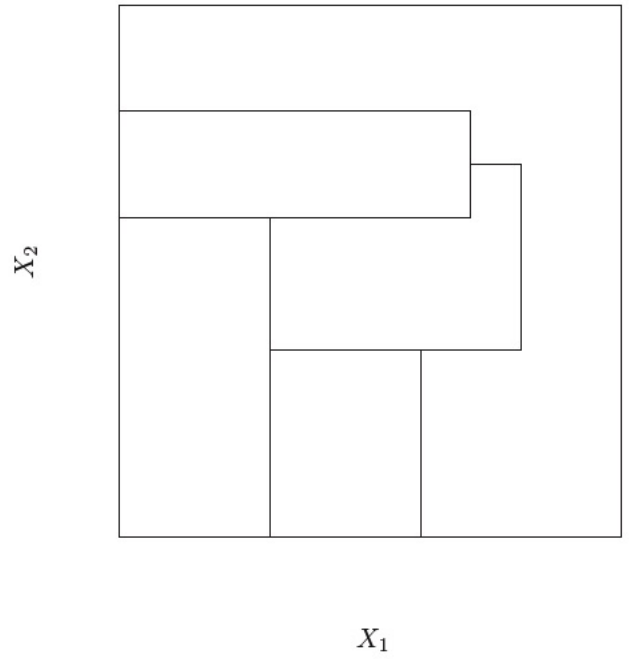
How do we construct the regions $R_1, R_2, \dots, R_J$?

HOW DO WE CONSTRUCT THE REGIONS?

- In theory, the regions could have any shape
- For simplicity and interpretability,
  - Divide the predictor space into high-dimensional rectangles, or boxes
- **Goal:** To find boxes $R_1, R_2, \ldots, R_J$ that minimize the RSS given by

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

  - $\hat{y}_{R_j}$: Mean response for the training observations within R_j



- However, it is **computationally infeasible** to consider every possible partition of the feature space into J boxes.

Recursive binary splits

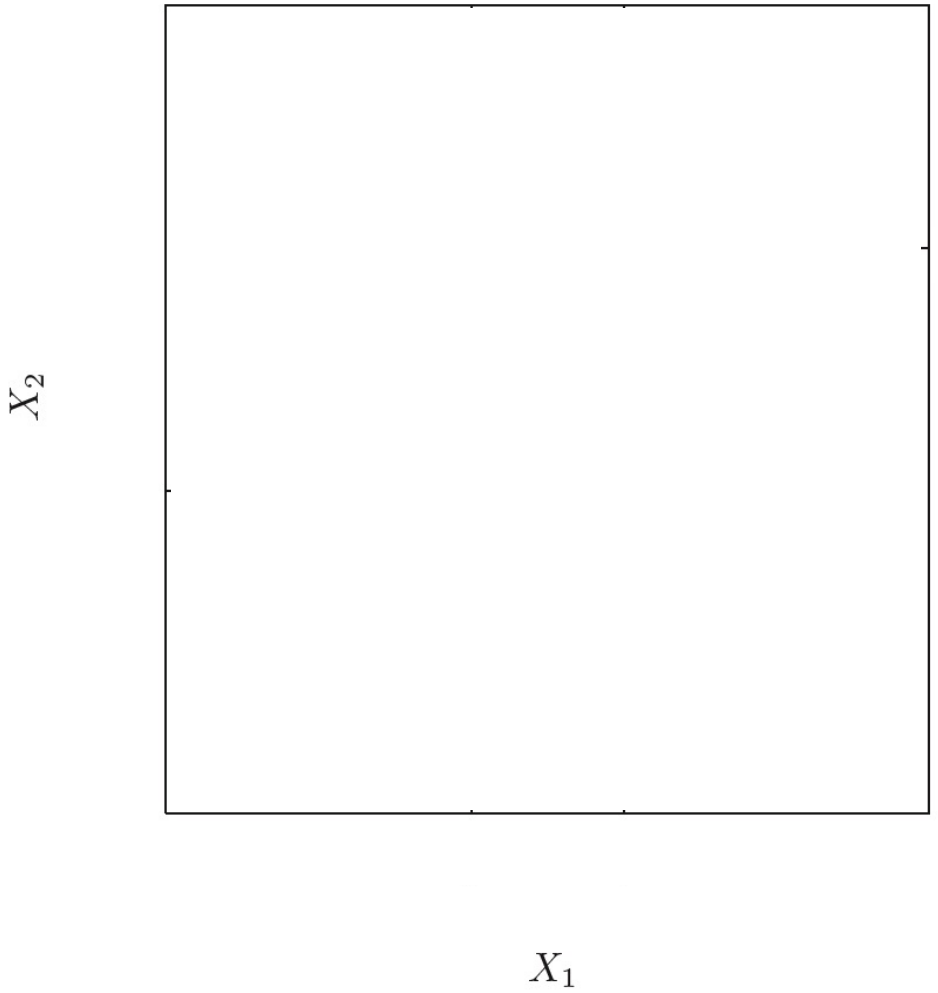
RECURSIVE BINARY SPLITS: OVERVIEW

- A ***top-down***, and ***greedy*** approach

- **Top-down** since it begins at the top of the tree (all observations belong to a single region) and then successively splits the predictor space.
- **Greedy** since at each step of the tree building process, the best split is made at that particular split (rather than looking ahead and picking a split that will lead to a better tree in a future split).
  - Short-sighted

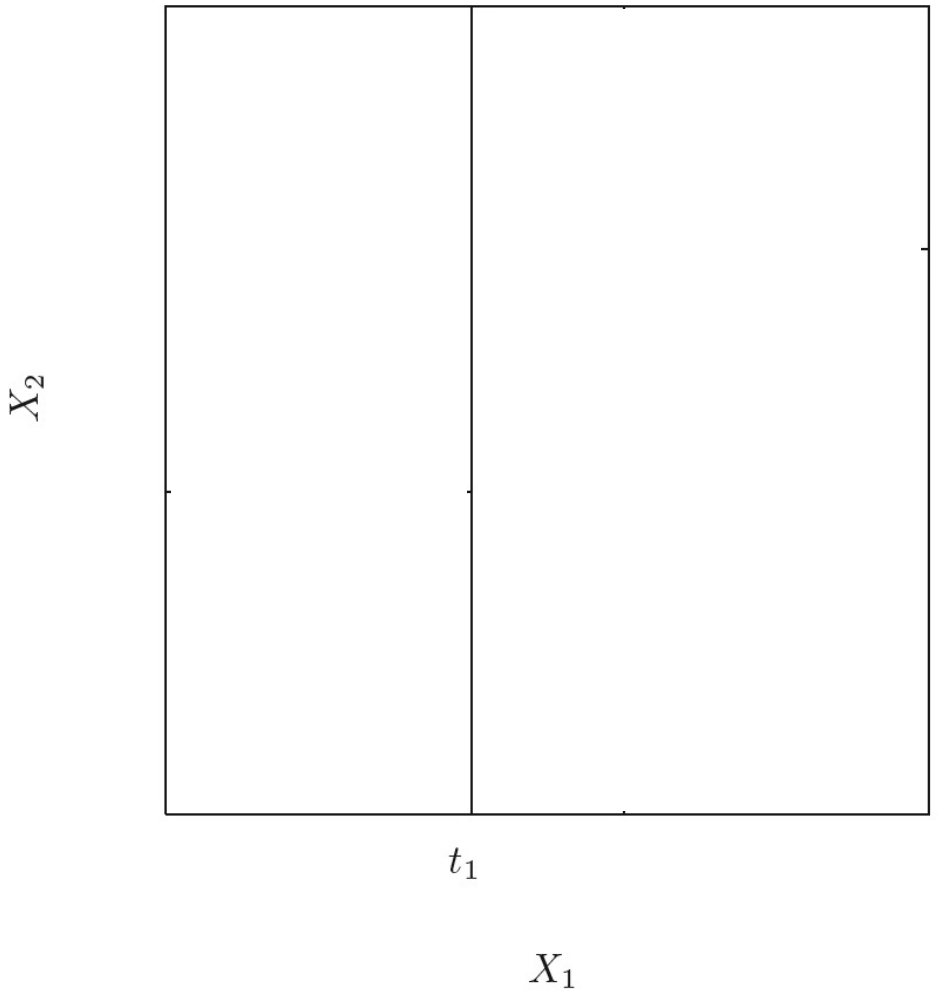
RECURSIVE BINARY SPLITS: EXAMPLES

- Generally we create the partitions by iteratively splitting one of the X variables into two regions



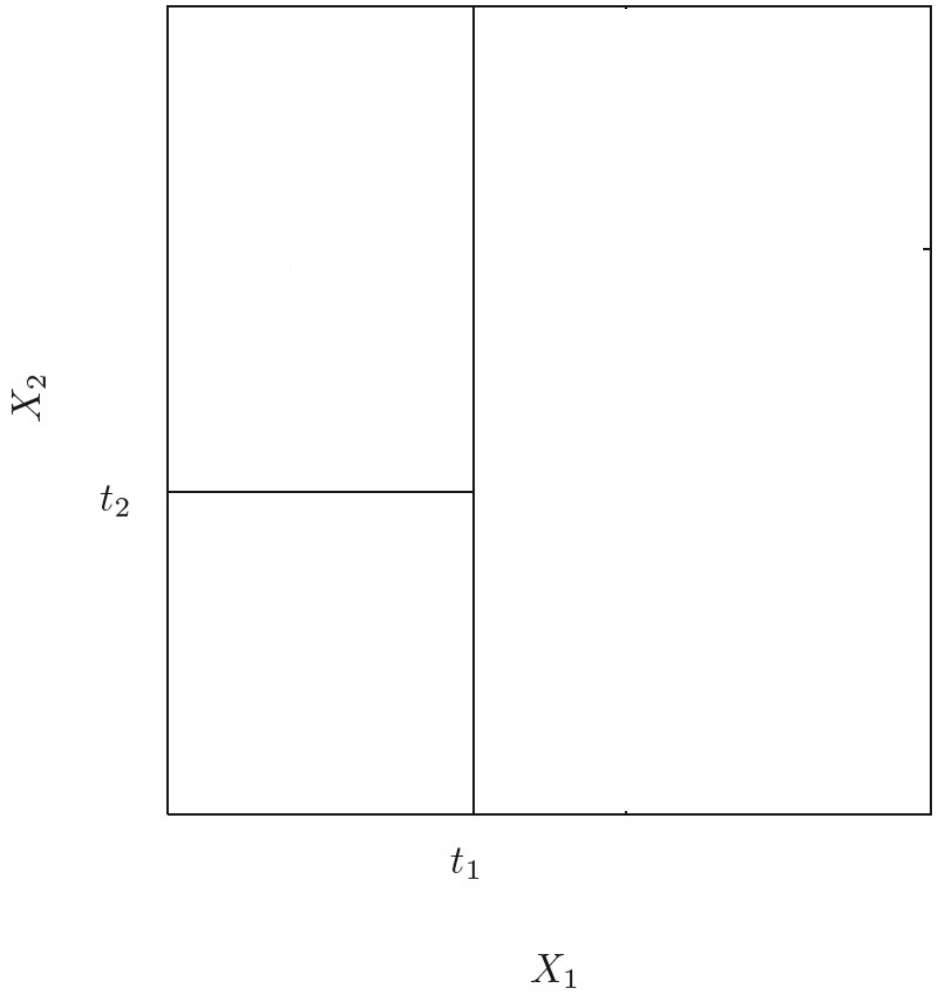
RECURSIVE BINARY SPLITS: EXAMPLES

- First split on $X_1 = t_1$



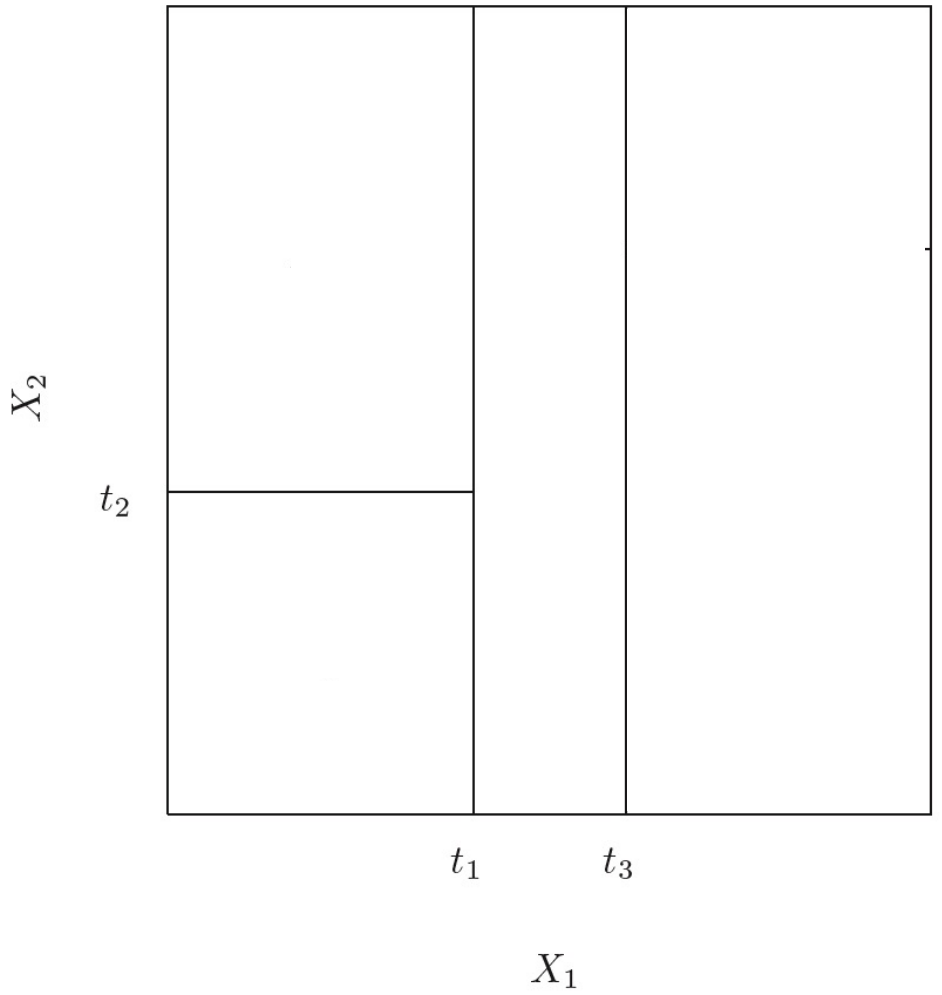
RECURSIVE BINARY SPLITS: EXAMPLES

- First split on $X_1 = t_1$
- If $X_1 < t_1$, split on $X_2 = t_2$



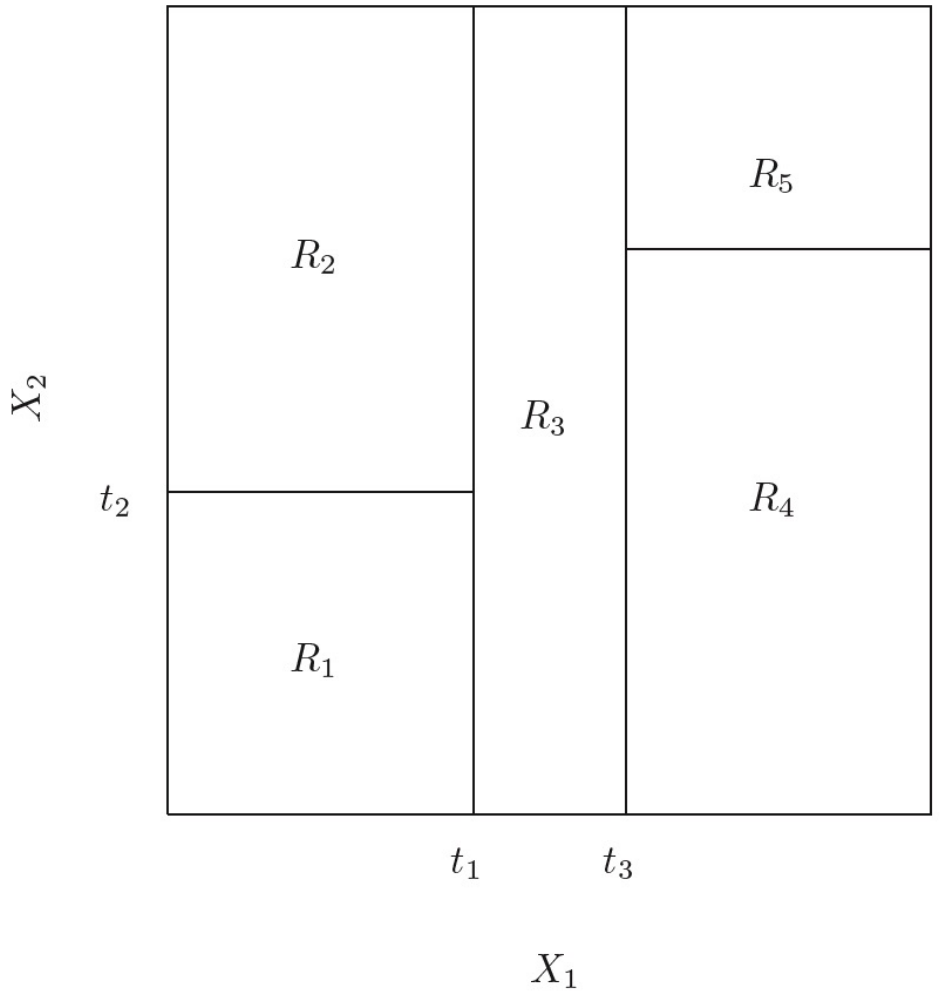
RECURSIVE BINARY SPLITS: EXAMPLES

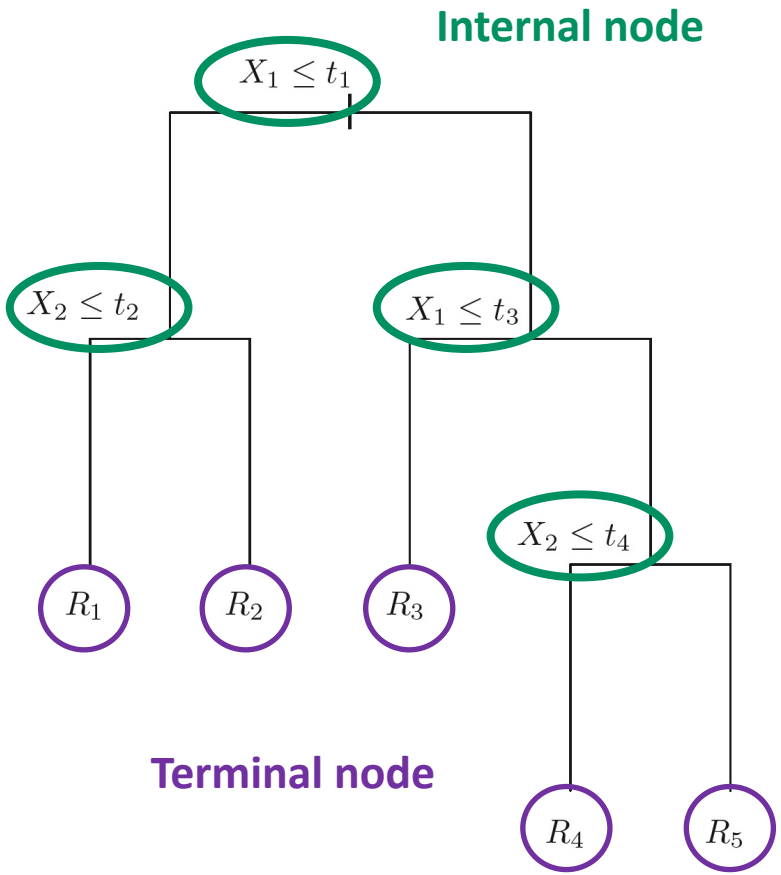
- First split on $X_1 = t_1$
- If $X_1 < t_1$, split on $X_2 = t_2$
- If $X_1 > t_1$, split on $X_1 = t_3$



RECURSIVE BINARY SPLITS: EXAMPLES

- First split on $X_1 = t_1$
- If $X_1 < t_1$, split on $X_2 = t_2$
- If $X_1 > t_1$, split on $X_1 = t_3$
- If $X_1 > t_3$, split on $X_2 = t_4$





RECURSIVE BINARY SPLITS

- For $j = 1, \ldots, p$ and all real value s,
  - Let $R_1(j, s) = \{i \in R: X_j < s\}$ and $R_2(j, s) = \{i \in R: X_j \geq s\}$
  - Let $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ be the mean response of all observations in $R_1(j, s)$ and $R_2(j, s)$, respectively.
- **Goal:** We seek the value of j and s that minimize the following equation

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

- Repeat this process until a stopping criterion is reached
  - e.g. all regions have 5 or fewer points.

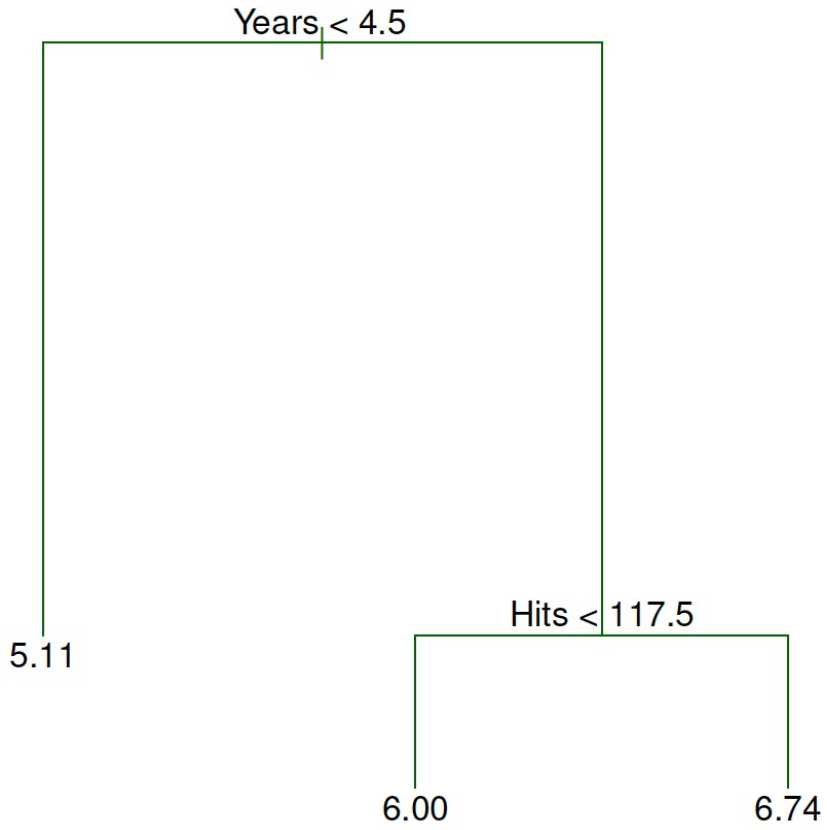
- **Prediction**
  - Once the regions $R_1, R_2, \ldots, R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

EXAMPLE: HITTERS DATASET

- The predicted "Salary" is the number in each terminal node
  - **Mean of the response** for the observations that fall there
- Note that Salary is measured in $1,000s, and log-transformed
- How to interpret?
  - The predicted salary for a player who played in the league for more than 4.5 years and had less than 117.5 hits last year is

$$\$1000 \times e^{6.00} = \$402{,}834$$

  - Years: Most important factor in determining Salary
    - Players with less experience earn lower salaries than more experienced players
  - Players with Years < 5?
    - The number of hits in the previous year seems to play little role in his salary
  - Players with Years>5?
    - The number of hits in the previous year does affect salary
    - Players with more hits last year tend to have higher salaries



Nice interpretation… But Over-simplification of the true relationship between Hits, Years, and Salary

HOW TO BUILD REGRESSION TREES: OVERVIEW

- The process described so far may produce good predictions on the training set, but is likely to **overfit** the data, leading to **poor test set performance**.
  - Too many splits→ Too complex

- A smaller tree with fewer splits (i.e., fewer regions $R_1, \dots, R_j$) might lead to **lower variance** and **better interpretation** at the cost of a little increase in bias.

- One possible alternative is to build the tree only as long as the **decrease in the RSS due to each split exceeds some (high) threshold**

- This results in smaller trees, however, this is problematic since a worthless split early on in the tree might be followed by a very good split later on
  - That is, even if the current split is not effective, the future splits could be effective (Short-sighted, Greedy)

HOW TO BUILD REGRESSION TREES: TREE PRUNING

- **Idea**: Build a very large tree T_0, then cut off ("prune") the branches that are not improving performance to obtain a ***subtree***

- How do we find the best subtree?
  - We want to select a subtree that leads to the **lowest test error rate**
  - Given a subtree, we can estimate the test error rate using cross-validation (CV)
- However, estimating the CV error for every possible subtree would take a long time since there are too many subtrees
- Thus, we need a way to **select a small set of subtrees** for consideration

HOW TO BUILD REGRESSION TREES: TREE PRUNING

- **Cost complexity pruning** (a.k.a. weakest link pruning)
- **Idea**: Rather than looking at all possible subtrees, 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 the following is as small as possible

"Loss + Penalty"
(Ridge, Lasso)

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

Favors trees with fewer terminal nodes

- $|T|$: The number of terminal nodes of the tree T
- R_m: The rectangle or box corresponding to the m-th terminal node
- $\hat{y}_{R_m}$: The predicted response associated with R_m
- α: Controls a trade-off between the subtree's complexity and its fit to the training data (Large α, less complex)
  - If $\alpha = 0$, no penalty. The optimal tree is the original tree T_0
  - If $\alpha = \infty$, no split at all. The predictor is just $\bar{y}$
  - Selected through cross-validation

We increase α from zero to obtain a sequence of subtrees as a function of α

TREE PRUNING: ALGORITHM

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

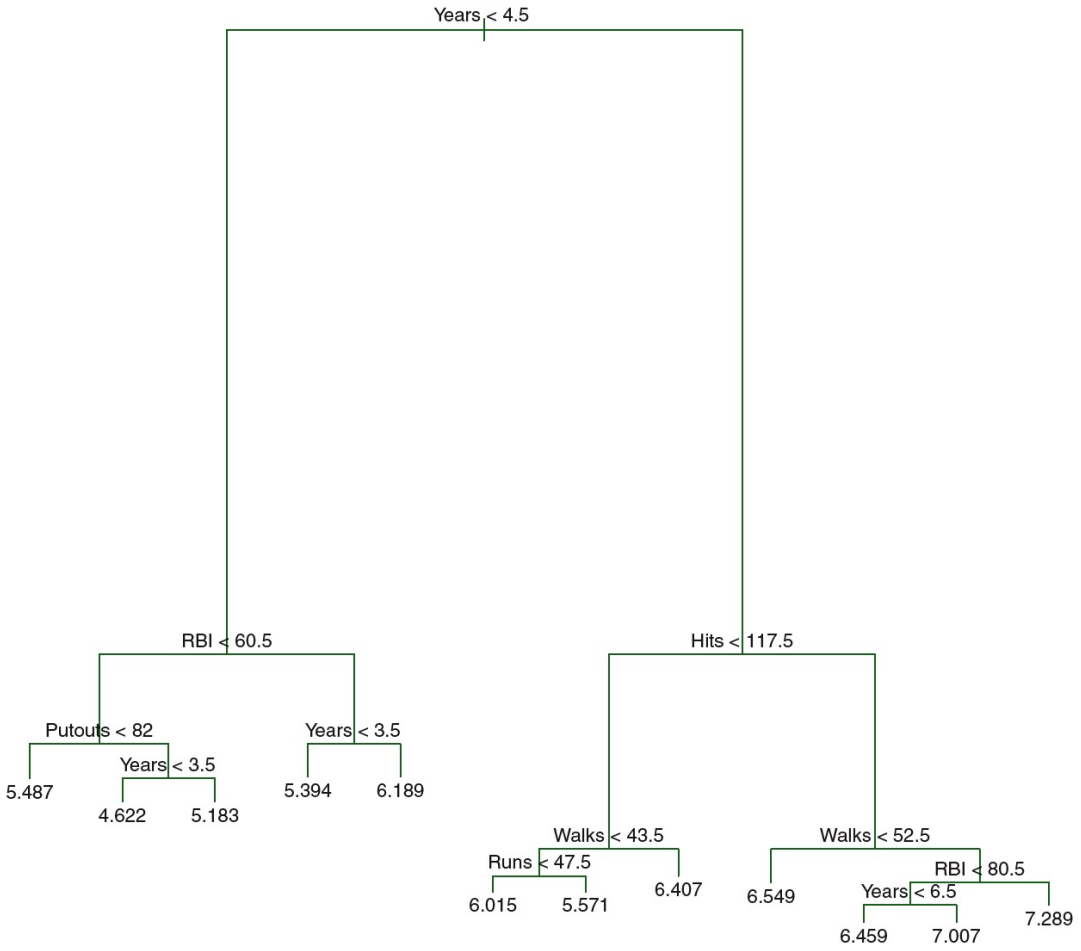
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

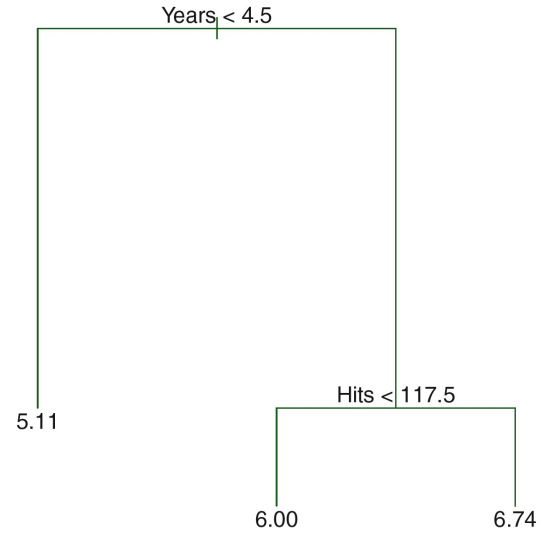
   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

EXAMPLE: HITTERS DATASET

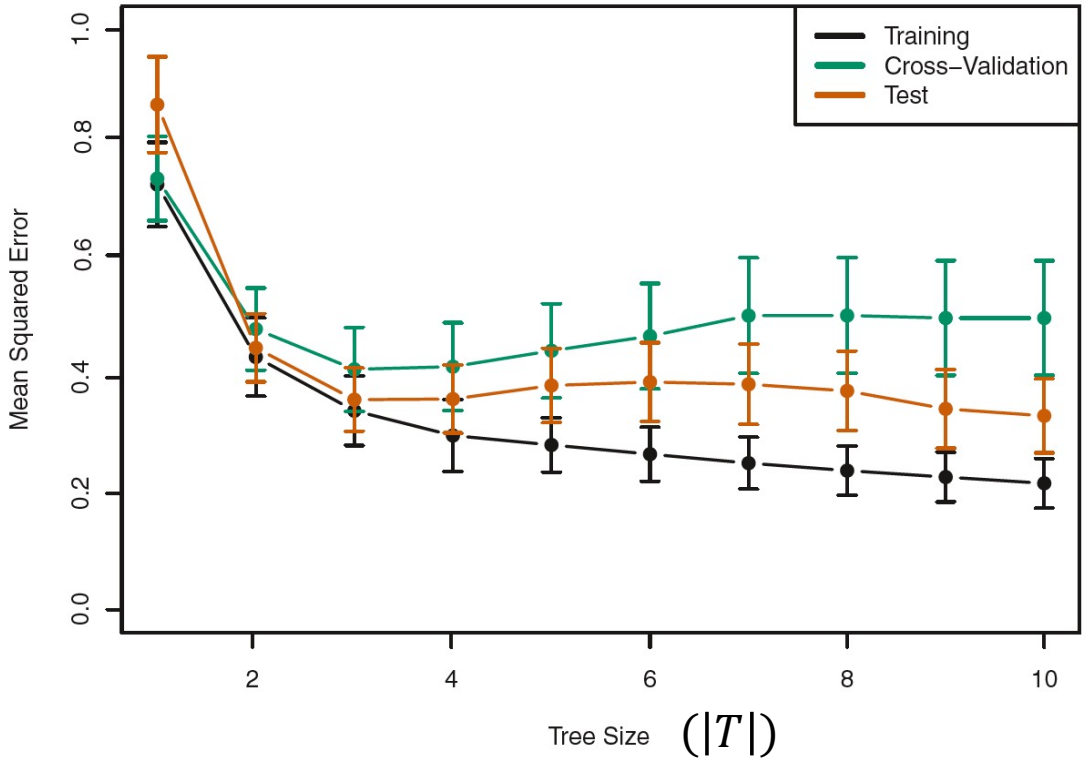


Unpruned tree



Pruned tree containing 3 terminal nodes

Length of a branch represents the amount reduced MSE



OUTLINE

- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees
  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees
- Bagging
- Random Forests
- Boosting

CLASSIFICATION TREES

- **Recall:** Regression has numerical responses, and classification has qualitative responses
- For regression trees,
  - Prediction: The mean response of the training observations in the region
  - For criterion for making the binary split, we select the split with **the greatest reduction of RSS**.
- For classification trees
  - Prediction: The most commonly occurring class of the training observations in the region
  - For criterion for making the binary split, we follow the same idea but **using a different metric**
    - Classification error rate
    - The Gini Index
    - The cross-entropy

SPLIT CRITERION FOR CLASSIFICATION

- Classification error rate
  - Simply the fraction of the training observations in a region that do not belong to the most common class

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

  - $\hat{p}_{mk}$: The proportion of training observations in the mth region that are from the kth class
  - However, classification error is not sufficiently sensitive for tree-growing, and in practice **two other measures** are preferable

- The Gini index

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

  - A measure of total variance across the K classes (Measures uncertainty)
    - c.f. Variance of Bernoulli random variable
  - G is small if $\hat{p}_{mk}$ is close to zero or one.

- The cross-entropy

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

  - An alternative to the Gini index
  - D is small if mth node is pure
  - Very similar to the Gini index numerically

Both measures **node purity**
- Small value indicates a node contains observation from a single class

SPLIT CRITERION FOR CLASSIFICATION

- If we only are about the classification accuracy of the final pruned tree,
  - Classification error rate is used

- If we care about building the tree (the quality of a particular split),
  - Gini-index and cross-entropy are used

- Toy example

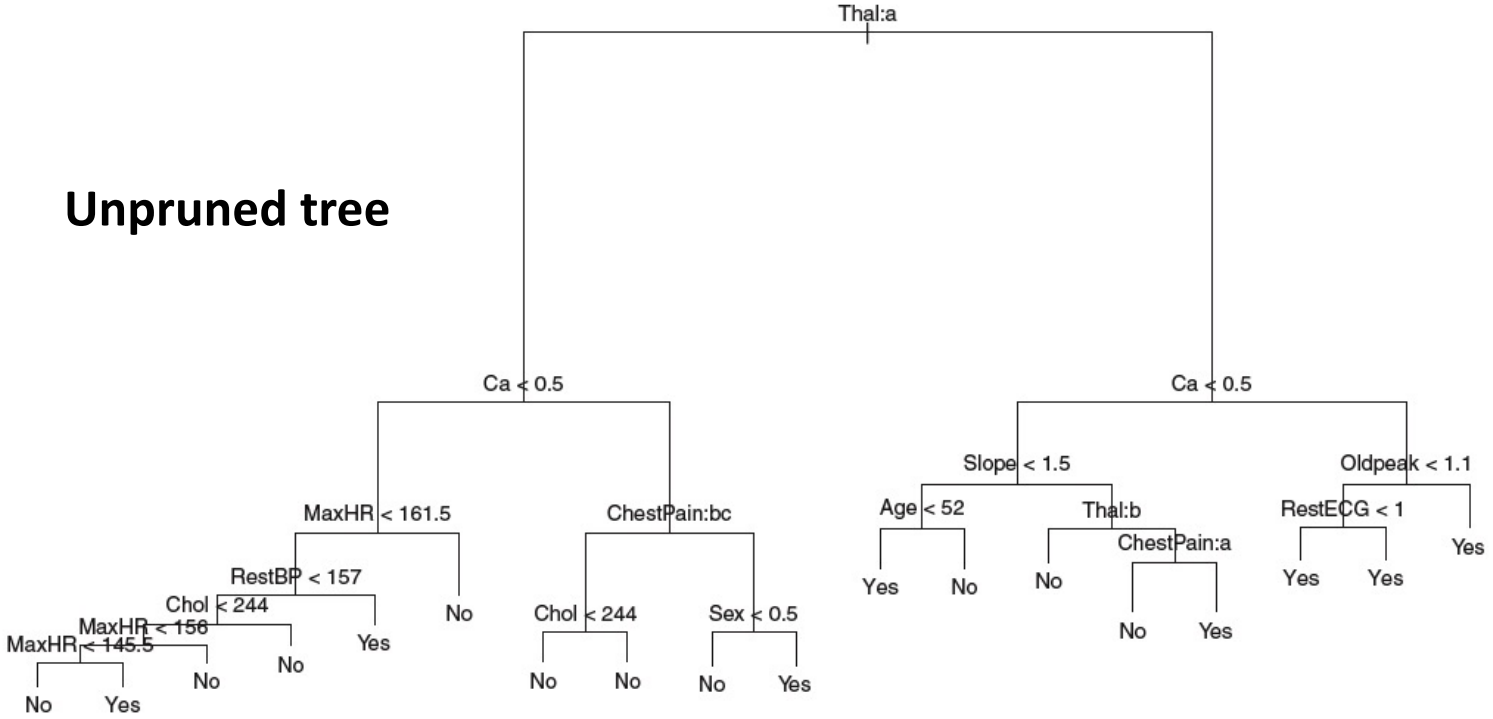
$\hat{p}_1 = [0.5,0.25,0.25] \Rightarrow E = 0.5, G = 0.625, D = 1.0397$

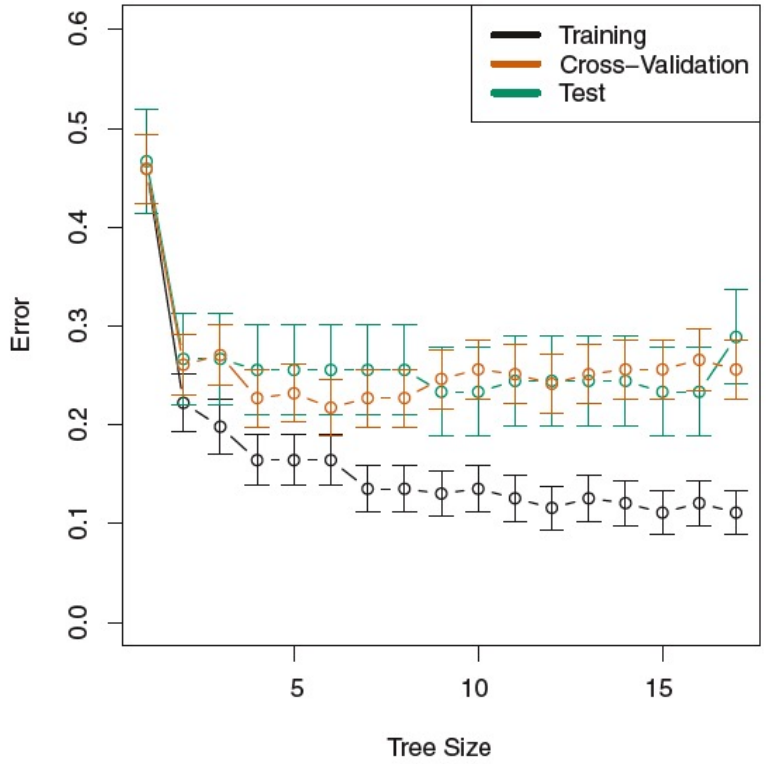
$\hat{p}_2 = [0.5,0.4,0.1] \Rightarrow E = 0.5, G = 0.580, D = 0.9433$

  - E cannot distinguish $\hat{p}_1$ and $\hat{p}_2$, while G and D give more information

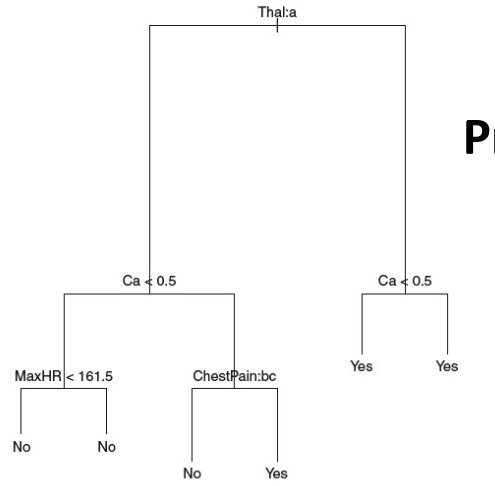
EXAMPLE: HEART DATASET

Unpruned tree





- Observations
  - Use of qualitative predictor variables (No need for dummy variables)
    - "Chestpain:bc": Left-hand branch comes from the 2nd and 3rd values of "Chestpain" variable
  - Some of the splits yield two terminal nodes that have the same predicted value
    - Due to increase in node purity
      - Even though the split "RestECG<1" does not reduce the classification error, it improves the Gini index and the entropy
    - Node purity is important for the confidence of the prediction



Pruned tree (CV)

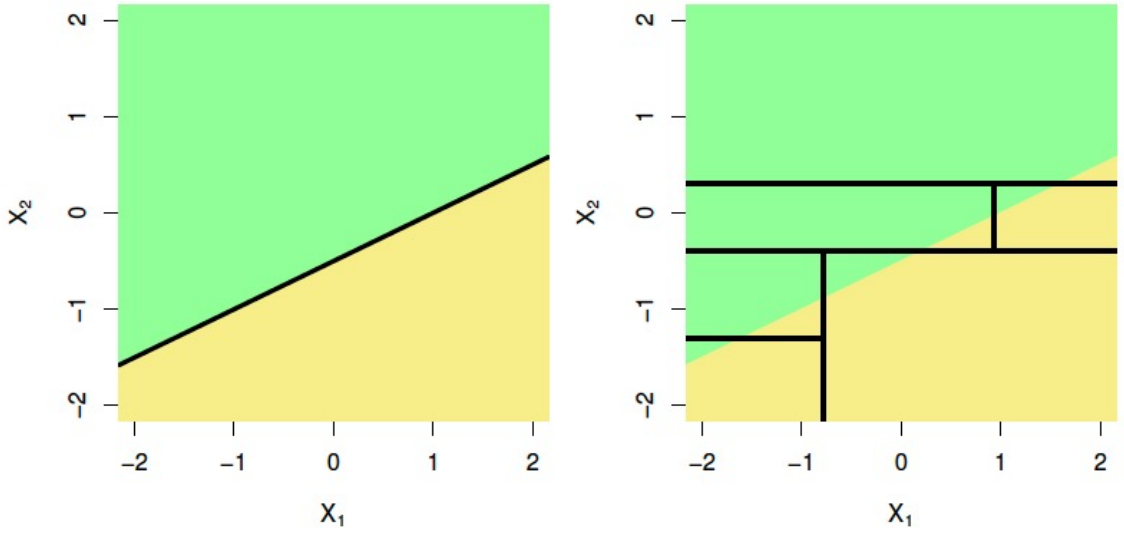
OUTLINE

- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees
  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees
- Bagging
- Random Forests
- Boosting

TREES VS. LINEAR MODELS

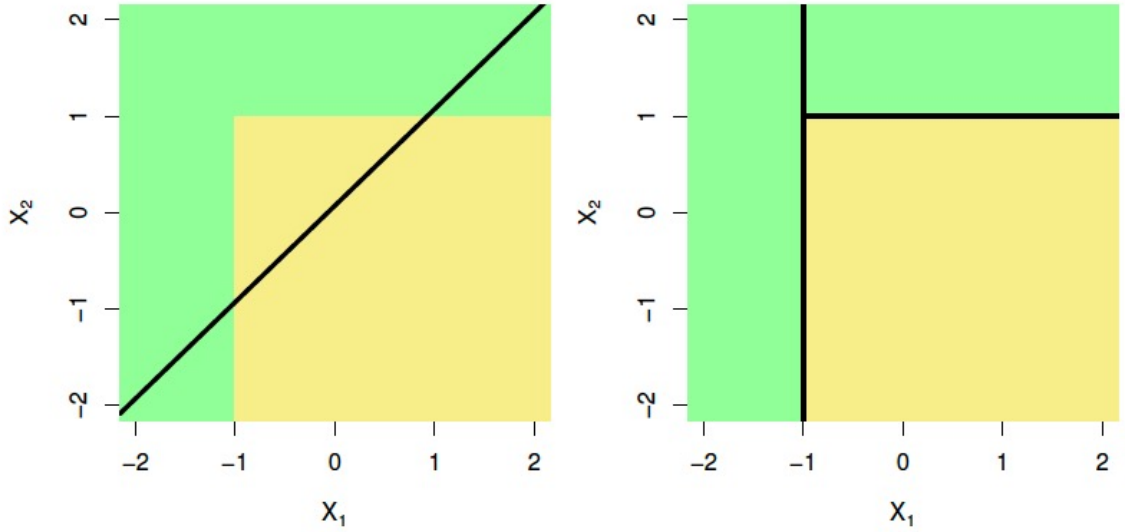
- Q: Which one is better?
- A: Depends on the problem

Decision boundary is linear



Linear model is better!

Decision boundary is non linear



Tree-based method is better!

ADVANTAGES AND DISADVANTAGES OF DECISION TREE

- **Advantages**
  - Easy to explain. In fact, they are even easier to explain than linear regression!
  - Closely mirror human decision-making
  - Can displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
  - Can easily handle qualitative predictors without the need to create dummy variables.

- **Disadvantages**
  - Not accurate enough (vs. other methods we learned so far)
  - Can be very non-robust. A small change in the data can cause a large change in the final estimated tree.
    - High variance

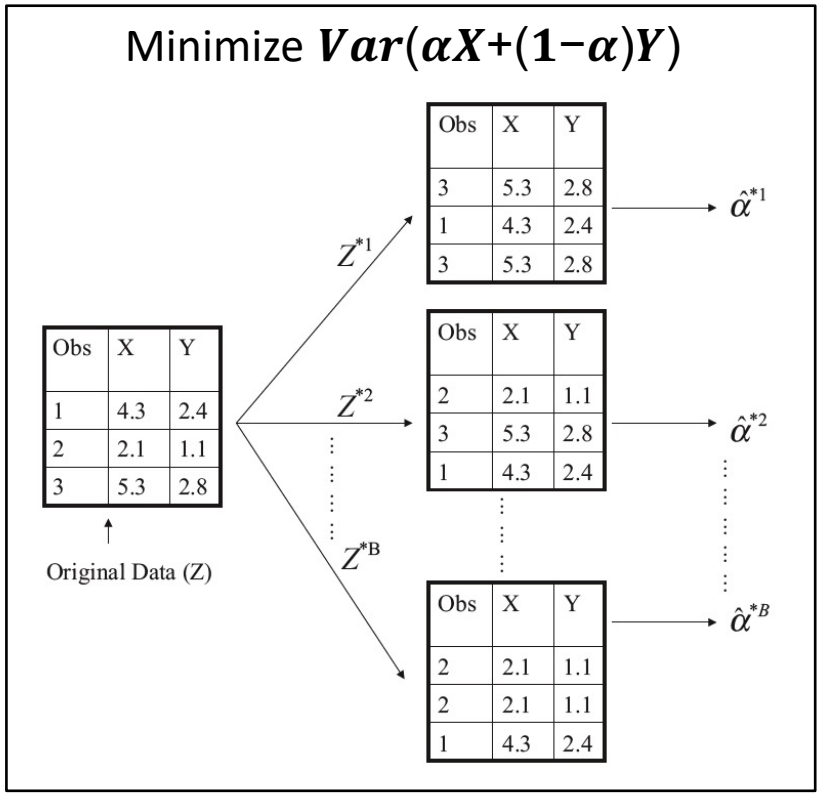
- However, we can improve by aggregating many decision trees, using methods like **bagging**, **random forests**, and **boosting**

OUTLINE

- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees
  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees
- Bagging
- Random Forests
- Boosting

BAGGING: OVERVIEW

- Decision trees suffer from **high variance**!
  - If we randomly split the training data into 2 parts, and fit decision trees on both parts, the results could be quite different
- We would like to have models with **low variance**
  - So as to yield similar results when applied repeatedly **to distinct data sets**
- Solution
  - Bagging: **B**ootstrap + **Agg**regat**ing**
- Recall bootstrap (Lecture 4) was used to compute the standard deviation of a quantity of interest
  - Slide 24 of Lecture 4
- Here, we will see that we can use the bootstrap to improve statistical learning through decision trees



BAGGING: OVERVIEW

- **Recall**: Given a set of n independent observations $X_1, X_2, \ldots, X_n$, each with variance σ^2, the variance of the mean $\bar{X}$ of the observations is given by $\frac{\sigma^2}{n}$

$$Var(\bar{X}) = Var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2} Var\left(\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n} Var(X_i) = \frac{1}{n^2} n\sigma^2 = \frac{\sigma^2}{n} = SE(\bar{X})^2$$

  - Refer to Slide 13 of Lecture 2

- This means that **averaging a set of observations reduces variance**

- Of course, this is not practical because we generally do not have access to multiple training sets.

- This is why we use **bootstrap**
  - Note that bagging is a general learning framework not only limited to decision tree

BAGGING

- Step 1: Generate B different bootstrapped training datasets
- Step 2: Train decision trees $\hat{f}^1(x), \hat{f}^2(x), \ldots, \hat{f}^B(x)$ using B separate bootstrapped training datasets
- Step 3: Average all the predictions, to obtain

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

- These trees are grown deep, and are **not pruned**
- Hence each individual tree has high variance, but low bias.
- This is why bagging is especially useful for decision trees
  - Bagging reduces variance
  - Bagging is useful for methods with high variance and low bias
- For classification, **majority vote** is applied
  - Take the class that occurs most commonly
- In practice, we combine thousands of trees

OUT-OF-BAG ERROR ESTIMATION

- **Problem:** Performing cross validation for all B bootstrapped training observation is time-consuming
- We can instead rely on **out-of-bag error estimation**
- Recall that the key to bagging is that trees are repeatedly fit to **bootstrapped** subsets of the observations.
- One can show that on average, each bagged tree makes use of around **two-thirds of the observations**
  - **Proof** (Remember that bootstrapped is sampling with replacement)
    - P(**first** bootstrap observation is not the jth observation) $= \frac{n-1}{n}$
    - P(**second** bootstrap observation is not the jth observation) $= \frac{n-1}{n}$
    - P(Having the jth observation in a bootstrap observation) $= 1 - \left(\frac{n-1}{n}\right)^n$
    - $\lim_{n\to\infty} 1 - \left(\frac{n-1}{n}\right)^n \Leftrightarrow 1 - \lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = 1 - e^{-1} \approx 1 - 0.36788 = 0.63212.$
- **OOB observation**: The remaining one-third of the observations not used to fit a given bagged tree

OUT-OF-BAG ERROR ESTIMATION

- How can we use OOB observations for prediction?
  - We can predict the response for the ith observation using each of the trees in which that observation was OOB.
  - This will yield around B/3 predictions for the ith observation.
  - Averaging or majority voting the response leads to a single OOB prediction for the ith observation

- An OOB prediction can be obtained in this way for each of the n observations
- The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation.

$$\text{OOB MSE} = \sum_{i=1}^{n} \left(y_i - \hat{f}^*_{-i}(x_i)\right)^2 \qquad \text{OOB classification error} = \sum_{i=1}^{n} I\left(y_i \notin \hat{f}^*_{-i}(x_i)\right)$$

- OOB estimate is essentially the LOOCV error for bagging, if B is large

VARIABLE IMPORTANCE MEASURES

- Bagging improves **prediction accuracy** at the expense of **interpretability**
  - No longer possible to represent the resulting statistical learning procedure using a single tree
  - No longer clear which variables are most important to the procedure

- However, we can obtain an **overall summary of the importance of each predictor**
  - For bagging regression trees, we can use the RSS
    - Record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees
  - For bagging classification trees, we can use the Gini index
    - Add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees
  - A large value indicates an important predictor

EXAMPLE: HEART DATASET

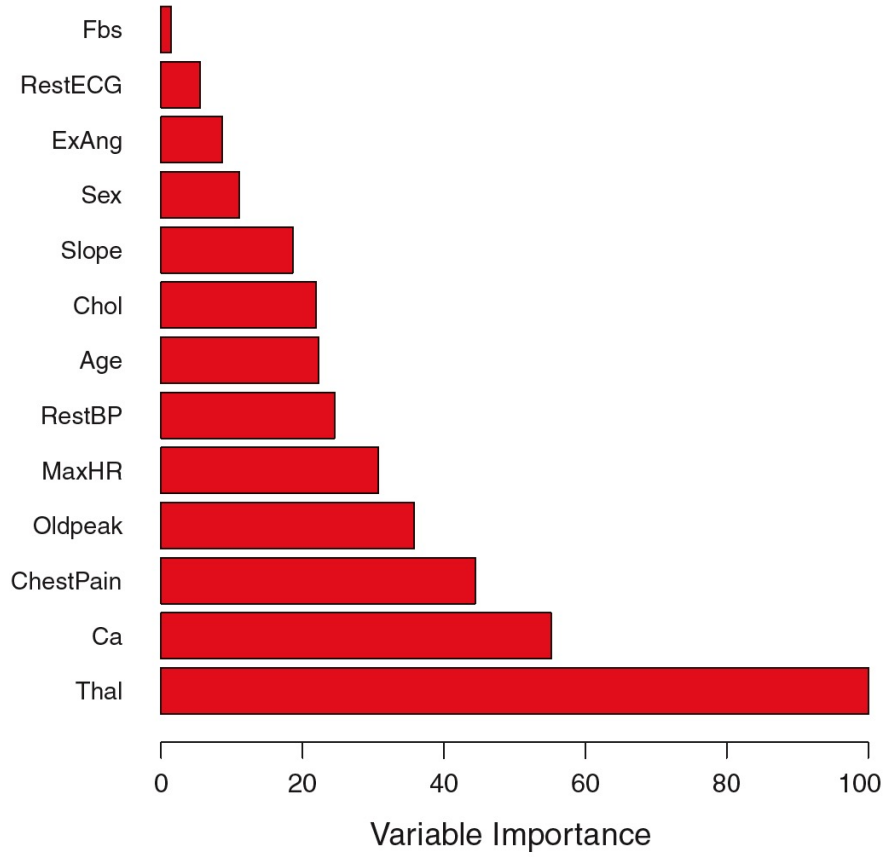


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

- The mean decrease in Gini index for each variable, relative to the largest.
- The variables with the largest mean decrease in Gini index are “Thal”, “Ca”, and “ChestPain”

OUTLINE

- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees
  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees
- Bagging
- Random Forests
- Boosting

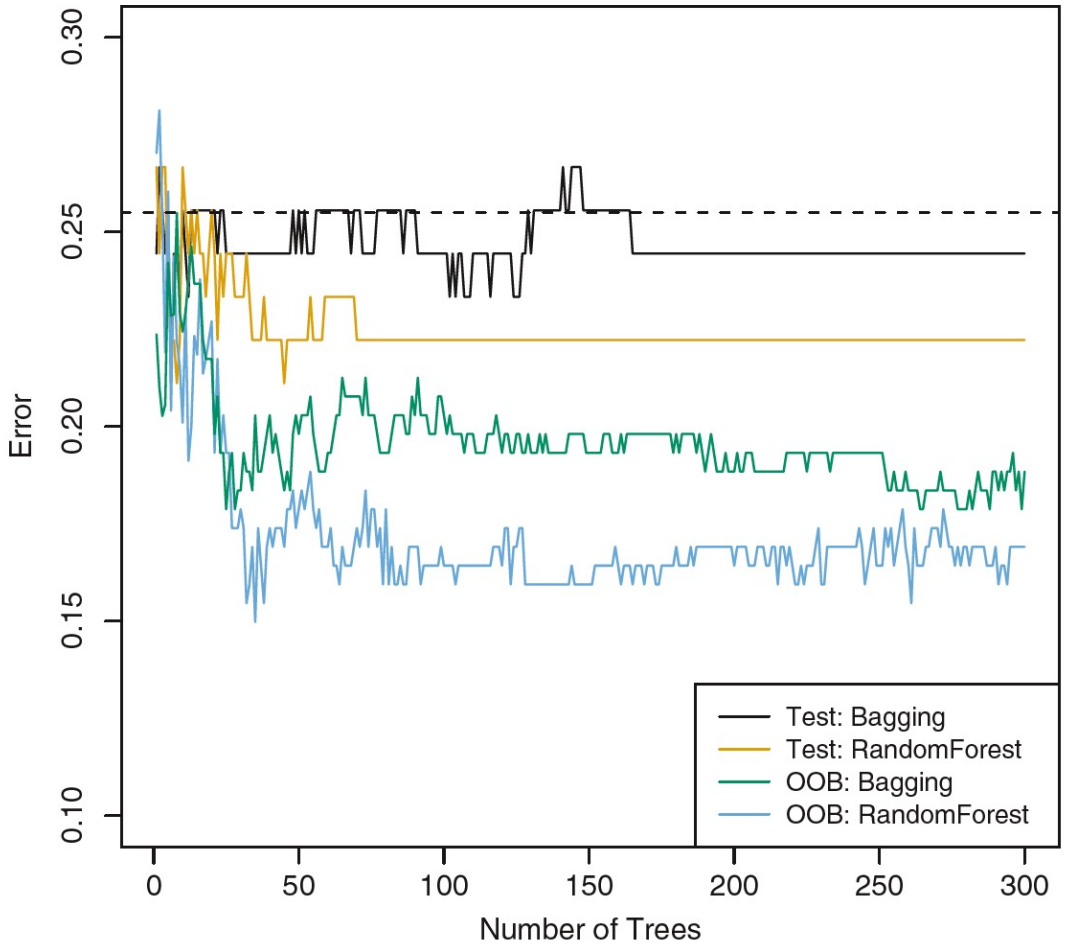
RANDOM FOREST: OVERVIEW

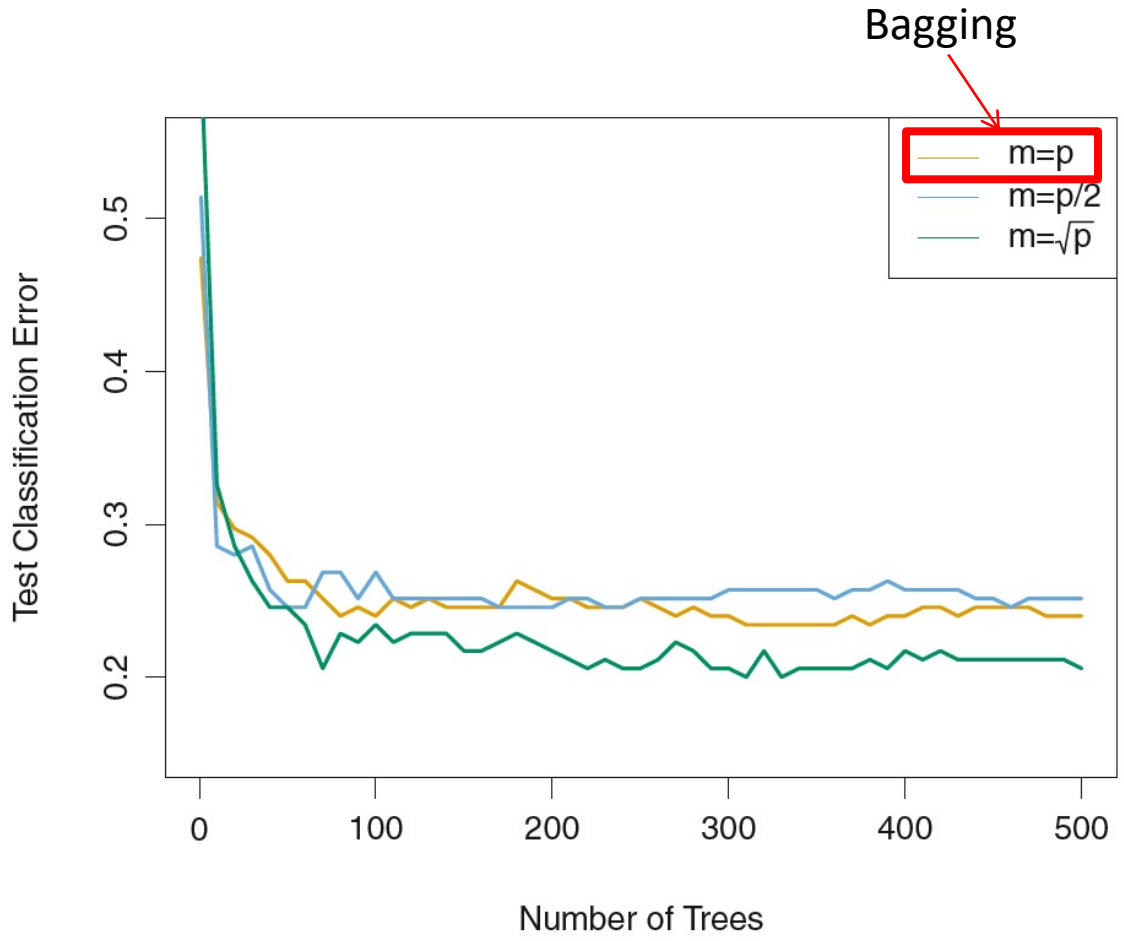
- **Problem**: The trees built using the bootstrapped samples are **correlated (High variance)**
  - Suppose that there is one very strong predictor
  - Then in the collection of bagged trees, *most of the trees will use this strong predictor in the top split*.
  - All of the bagged trees will look similar and 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
  - **Bagging will not lead to a substantial reduction in variance over a single tree in this setting**

- **Idea:** Random forests improves bagged trees by **decorrelating** the trees to reduce the variance

- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each tree is allowed to use only m predictors from the full set of p predictors
  - $m \approx \sqrt{p}$ is a good choice
- Random forest reduces to bagging if $m = p$.

EXAMPLE: HEART DATASET AND GENE EXPRESSION DATASET





OUTLINE

- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees
  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees
- Bagging
- Random Forests
- Boosting

BOOSTING: OVERVIEW

- 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 for regression or classification trees
- Recall in bagging, all the **trees are grown independently**
  - Step 1: Create multiple copies of the original training data set using the bootstrap.
  - Step 2: Fit a separate decision tree to each bootstrapped copy.
  - Step 3: Then combine all of the decision trees in order to create a single predictive model.
- Boosting works in a similar way, except that the **trees are grown sequentially**
  - Step 1: Each tree is grown using information from previously grown trees
  - Step 2: Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.
- 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 sequentially and slowly to avoid overfit**

BOOSTING: ALGORITHM

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

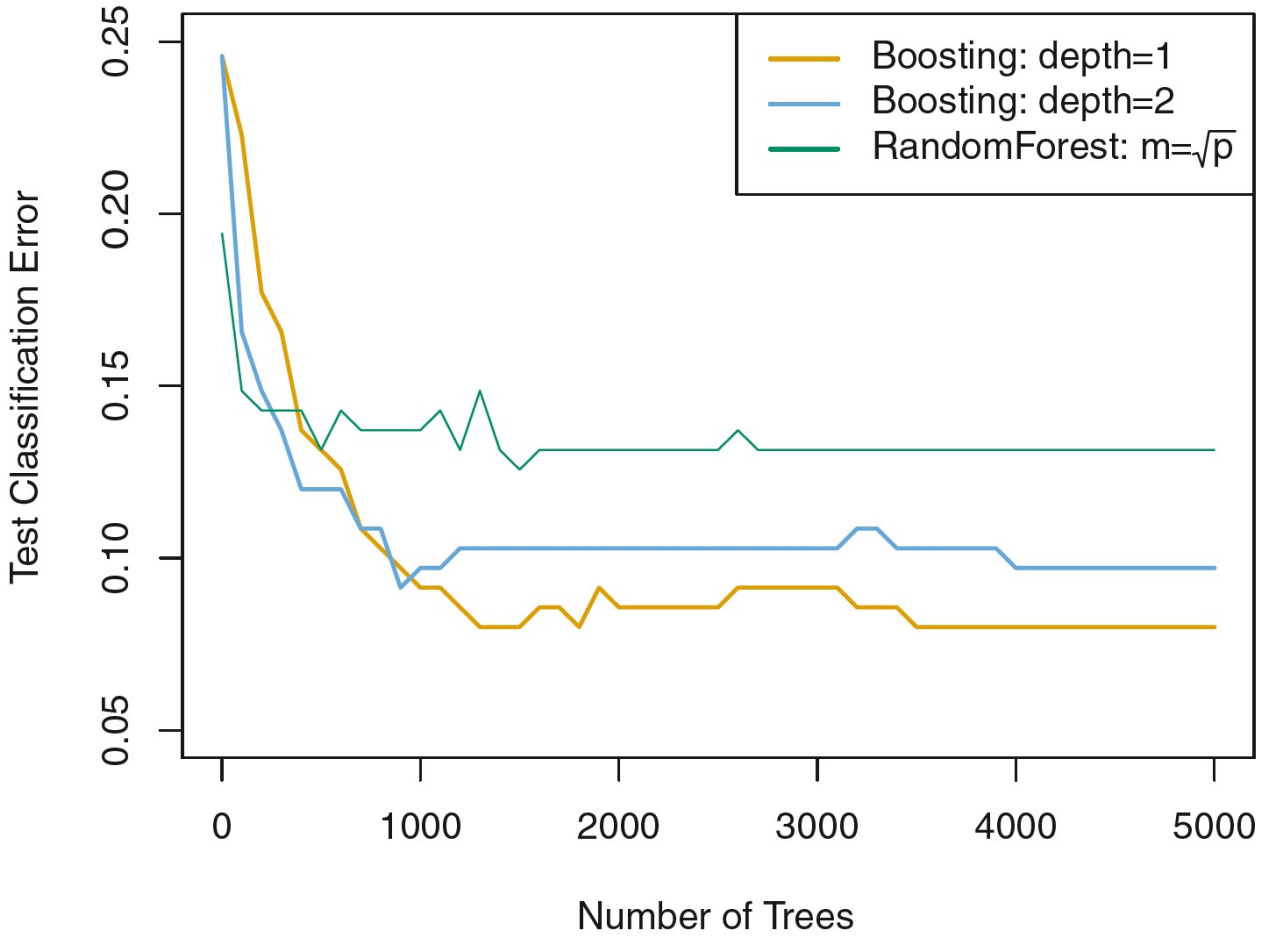
BOOSTING: DETAILS

Three tuning parameters

- 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 (CV to select B).
- 2. The shrinkage parameter λ.
  - Controls the rate at which boosting learns
  - Typical values are 0.01 or 0.001, and the right choice can depend on the application
  - Very small λ can require using a very large value of B in order to achieve good performance.
- 3. The number d splits in each tree.
  - 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.

(A good explanation of Adaboost) https://www.youtube.com/watch?v=LsK-xG1cLYA

EXAMPLE: GENE EXPRESSION DATASET



- Observations
  - Simple stumps with an interaction depth of one perform well if enough of them are included.
  - This model outperforms the depth-two model, and both outperform a random forest.
- This highlights one difference between boosting and random forests
  - In boosting, because the growth of a particular tree takes into account the other trees that have already been grown, **smaller trees are typically sufficient**

CONCLUSION

- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees
  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees
- Bagging
- Random Forests
- Boosting

Coming up next:
Support Vector Machines