# Intro to Trees

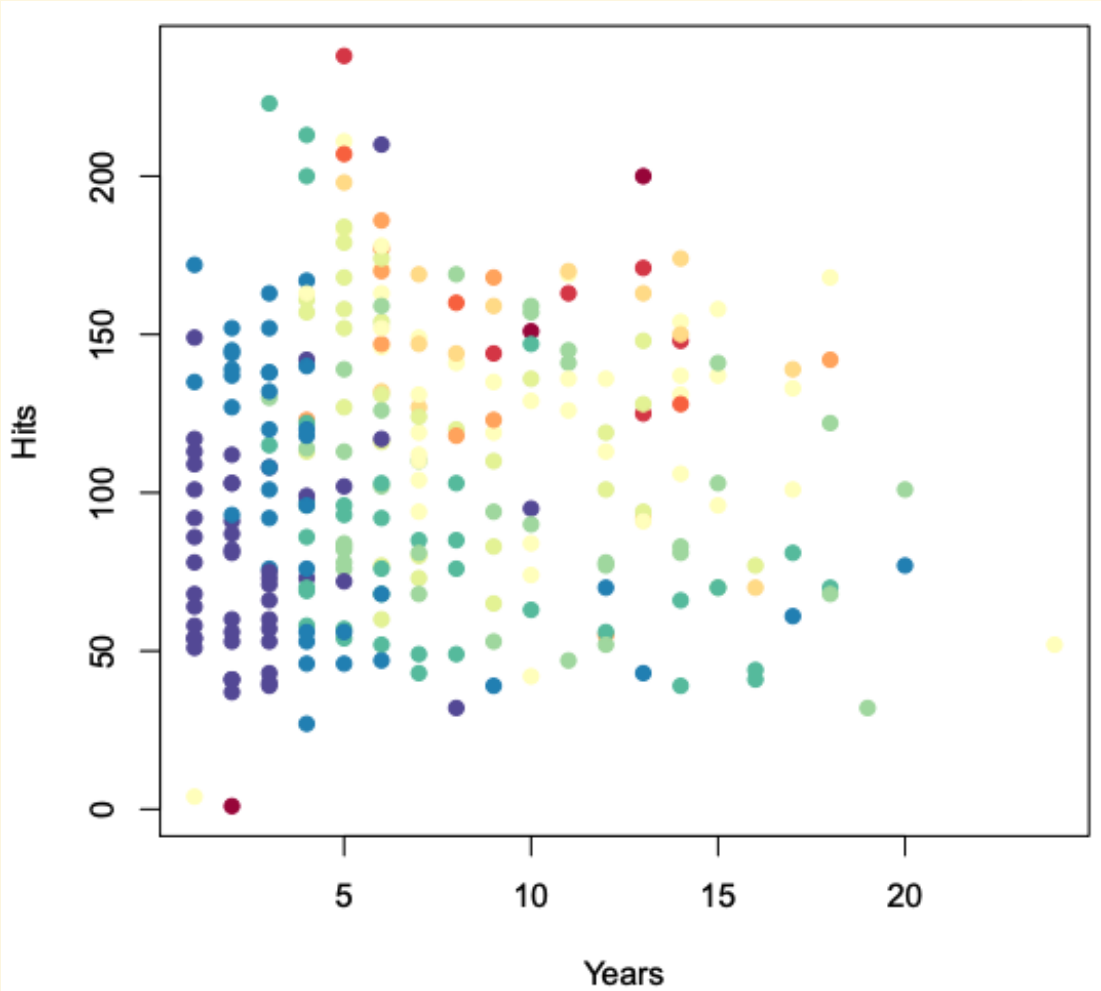
Instructor: Haoran LEI

Hunan University

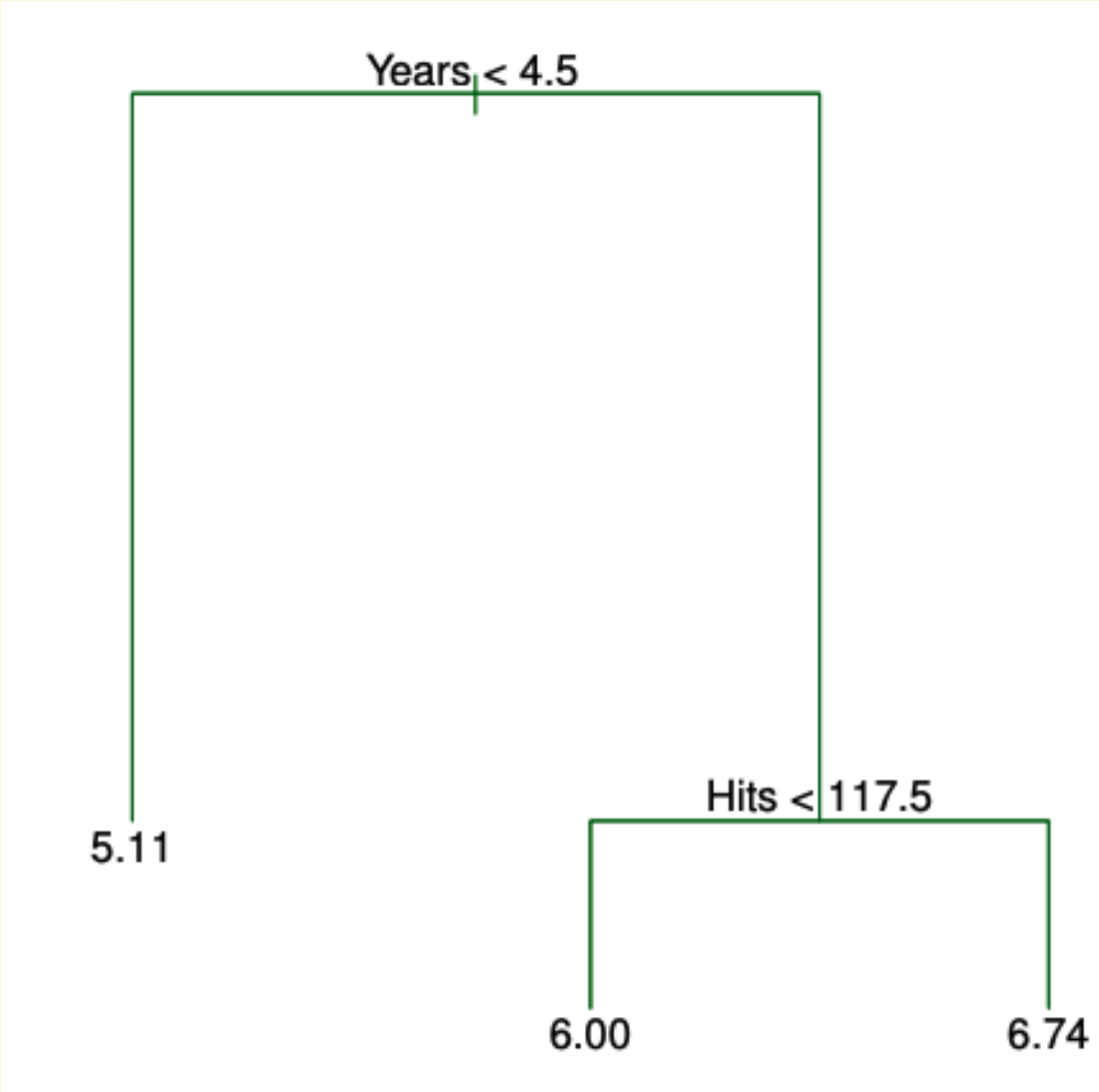
# Tree-based Methods

- We describe **tree-based methods** for regression (and classification).
- These involve *stratifying* (分層) or *segmenting* (分割) the predictor space into a number of simple regions.
- 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.

Baseball salary data: how would you stratify it?

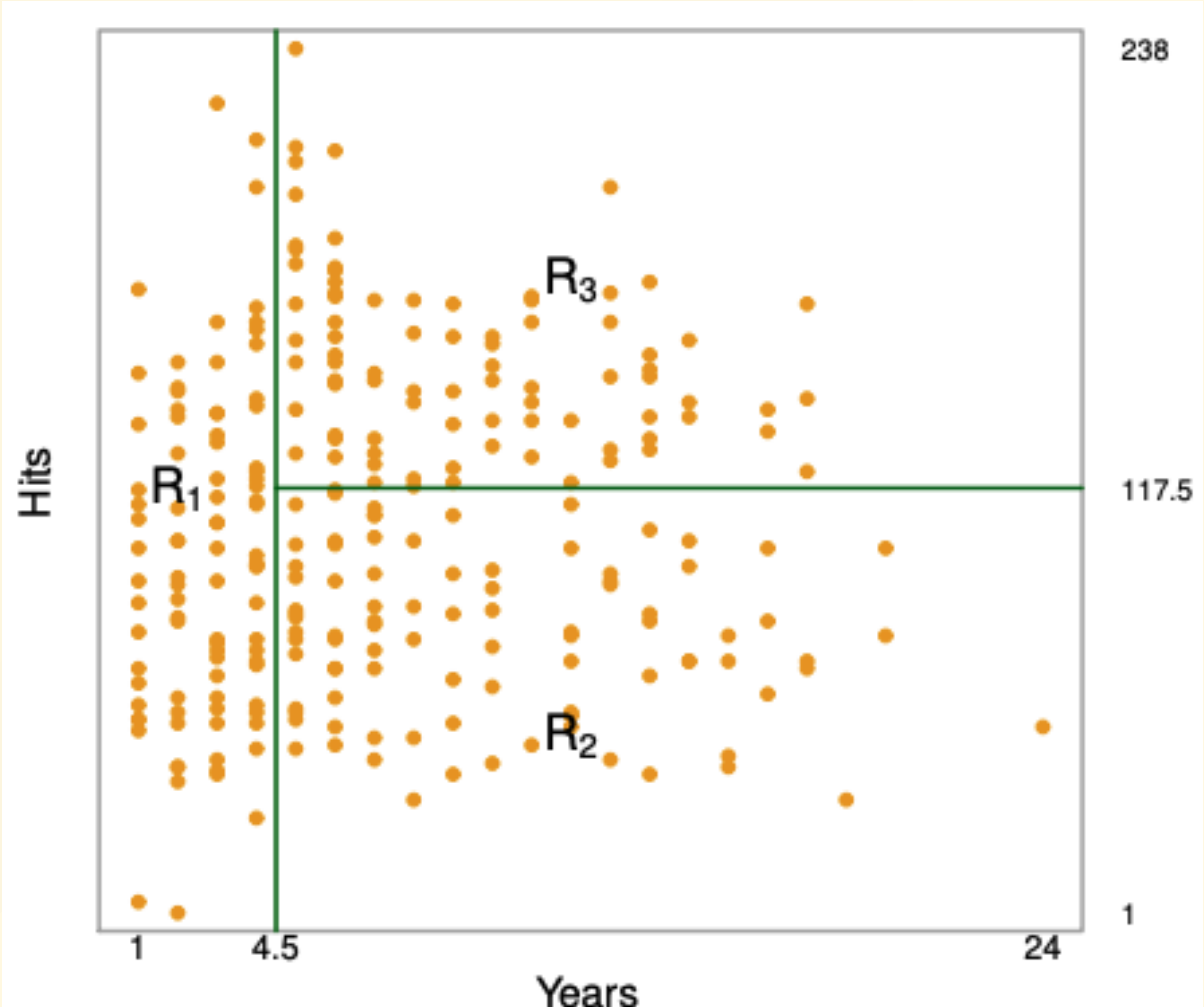



Salary is color-coded from **low** (blue, green) to **high** (yellow,red)




Example：Stratifying Baseball salary data

- The tree stratifies (segments) the players into three regions of predictor space: $R_1$, $R_2$, $R_3$.



# Terminology for Trees

- The regions $R_1$, $R_2$, and $R_3$ are known as *terminal nodes* (or *leaves*).
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*.

# Pros and Cons

- In the hitters tree, the two internal nodes are indicated by the text `Years < 4.5` and `Hits < 117.5`.
- Surely an over-simplification, but compared to a regression model, it is **easy to display, interpret and explain**.

# Pros and Cons

- Tree-based methods are simple and useful for interpretation.
- However, they typically are 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** which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in **prediction accuracy**, at the expense of some **loss interpretation**.

# Tree-building process

1. We divide the *predictor space* — that is, the set of possible values for $X_1$, $X_2$, ..., $X_p$ — into $J$ distinct and non-overlapping regions, $R_1$, $R_2$, . . . , $R_J$.

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 $R_j$.

## More details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find *boxes* $R_1, \ldots, R_J$ that minimize the RSS (training error).

## More details of the tree-building process

- However, even if we focus on boxes, it is still computationally infeasible to consider every possible partition of the feature space into $J$ boxes.
- For this reason, we take a **top-down, greedy** approach that is known as *recursive binary splitting.*
  - It's **top-down** because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
  - It is **greedy** because at each step of the tree-building process, *the best split is made at that particular step* (with no looking ahead)

## Recursive binary splitting

- First, select the predictor $X_j$ and the cutpoint $s$, such that this two-segmentation has the least RSS.

# Recursive binary splitting

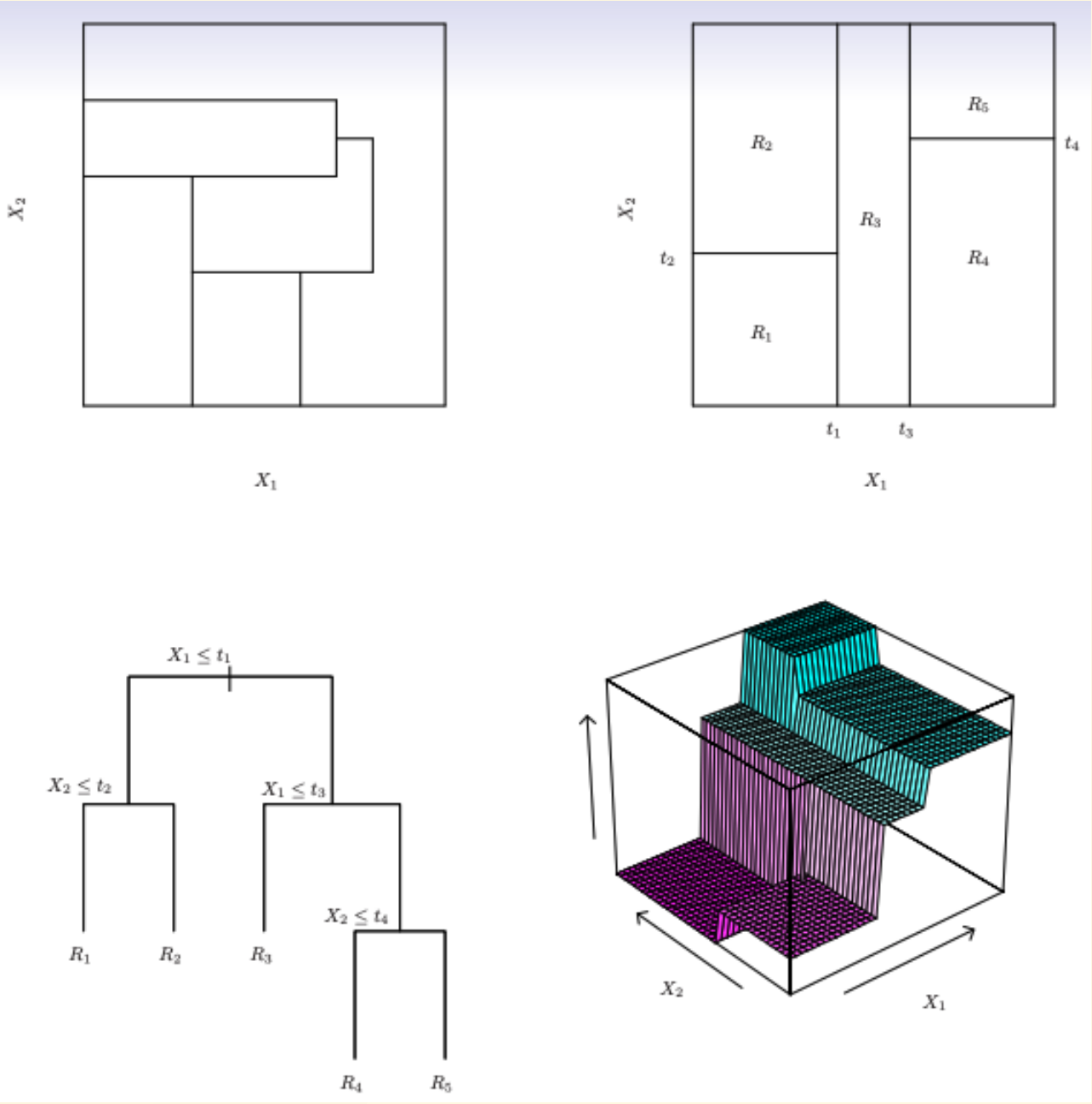
- First, select the predictor $X_j$ and the cutpoint $s$, such that this two-segmentation has the least RSS.
- Second, we **repeat the process:** looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
  - However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions.
  - We now have three regions.

# Recursive binary splitting

- First, select the predictor $X_j$ and the cutpoint $s$, such that this two-segmentation has the least RSS.
- Second, we **repeat the process:** looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting 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; Eg, we may continue until no region contains more than five observations.

## Predictions based on (decision) trees

- 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.
- A five-region example ($J = 5$) of this approach is shown in the next slide.


$X_2$
$X_1$
$R_1$
$R_2$
$R_3$
$R_4$
$R_5$
$t_1$
$t_2$
$t_3$
$t_4$
$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
$R_1$
$R_2$
$R_3$
$R_4$
$R_5$
$X_2$
$X_1$

# Details of previous figure

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

## Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to **overfit** the data, leading to poor test set performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1$, ..., $R_J$) might lead to lower variance and better interpretation at the cost of a little bias.

# Pruning a tree

- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on.

## Pruning a tree: Cost complexity pruning

- A better strategy is to **firstly grow a very large tree** $T_0$, and then prune it back in order to obtain a subtree.
  - *Cost complexity pruning* (or *weakest link pruning*).

## Pruning a tree: Cost complexity pruning

- Consider a sequence of trees indexed by a nonnegative tuning parameter $\alpha$. For each value of α there corresponds **a subtree** $T (\subset T_0)$ such that

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

- is as small as possible, where $|T|$ is the number of leaves of the tree $T$ and $R_m$ is the box corresponding to the $m$-th leave.

# Choosing the best subtree

- The tuning parameter $\alpha$ controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

## Summary: tree algorithm

1. Use *recursive binary splitting* to grow a large tree greedily on the training data --- get $T_0$.

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

## Summary: tree algorithm

1. Use *recursive binary splitting* to grow a large tree greedily on the training data --- get $T_0$.

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

## Summary: tree algorithm

3. Use *K-fold cross-validation* to choose α. For each $k = 1, \ldots, K$:
   - Repeat Steps 1 and 2 on the $(K-1)/K$-th fraction of the training data, excluding the $k$-th fold.
   - Evaluate the mean squared prediction error on the data in the left-out $k$-th fold, as a function of α.

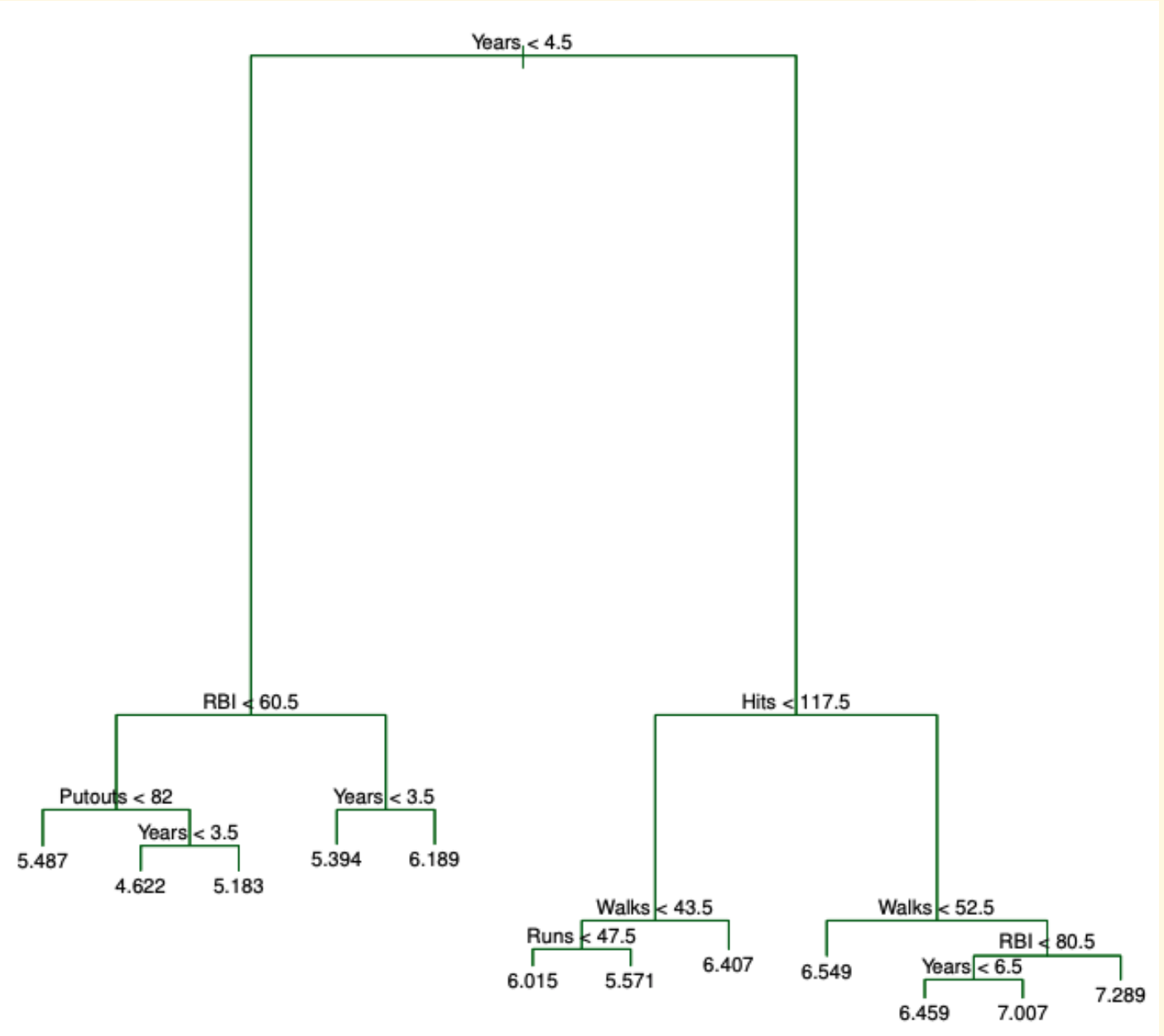
   Average the results, and pick α to minimize the average error.

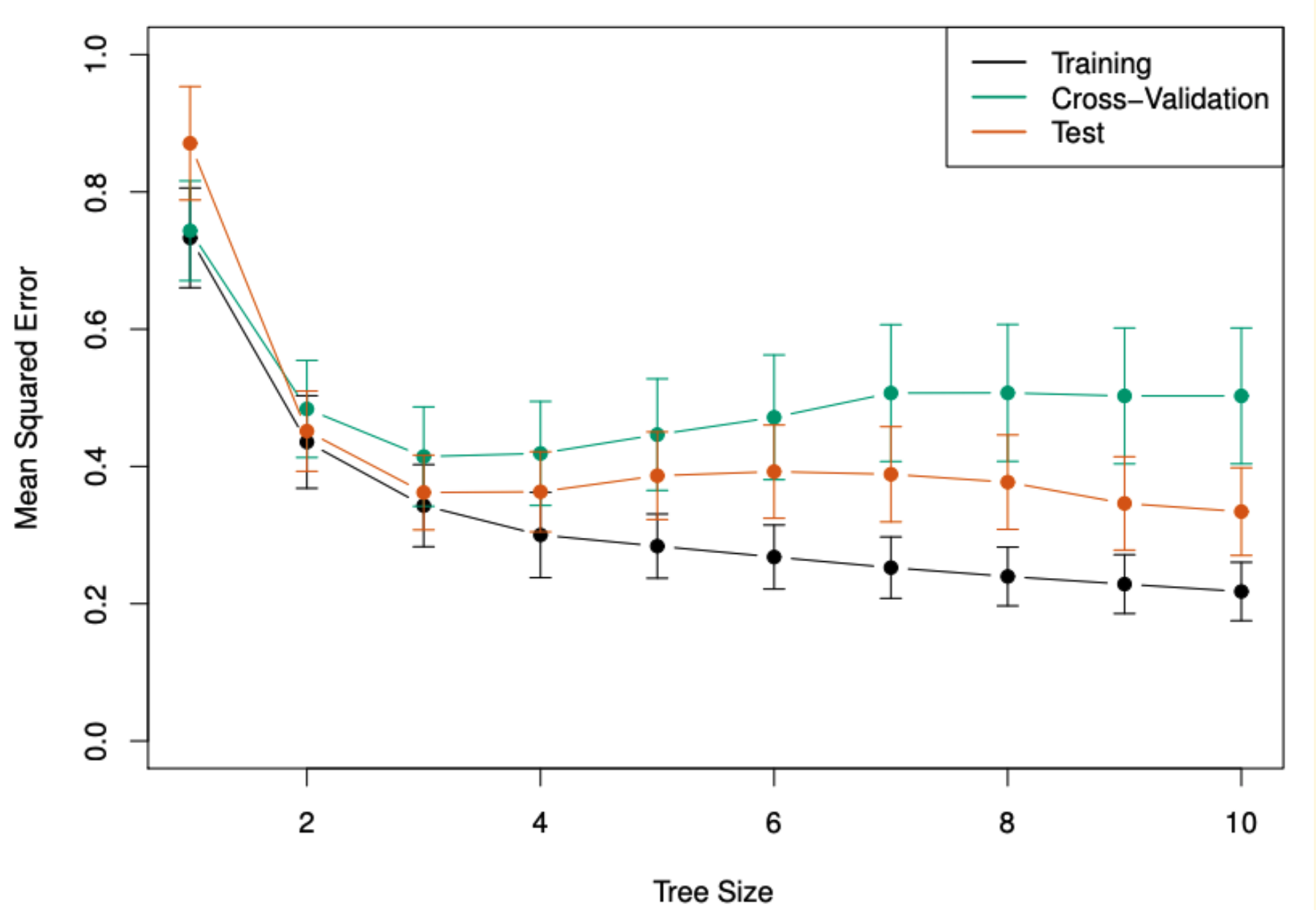
4. Return the subtree from Step 2 that corresponds to the chosen value of $\alpha$.

## Baseball example continued

- First, randomly divide the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- Then, build a large regression tree on the training data and varied $\alpha$ in in order to create subtrees with different numbers of terminal nodes.
- Finally, perform 6-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of $\alpha$.

$T_0$:


Years < 4.5
RBI < 60.5
Hits < 117.5
Putouts < 82
Years < 3.5
Years < 3.5
Walks < 43.5
Walks < 52.5
Runs < 47.5
RBI < 80.5
Years < 6.5
5.487
4.622
5.183
5.394
6.189
6.015
5.571
6.407
6.549
6.459
7.007
7.289


Training
Cross–Validation
Test
Mean Squared Error
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Tree Size