

# Tree pruning and bagging

STAT 471

November 4, 2021

# Rolling into a new unit!

✓ **Unit 1:** Intro to modern data mining

✓ **Unit 2:** Tuning predictive models

✓ **Unit 3:** Regression-based methods

**Unit 4:** Tree-based methods

**Unit 5:** Deep learning

**Lecture 1:** Growing decision trees

**Lecture 2:** Tree pruning and bagging

**Lecture 3:** Random forests

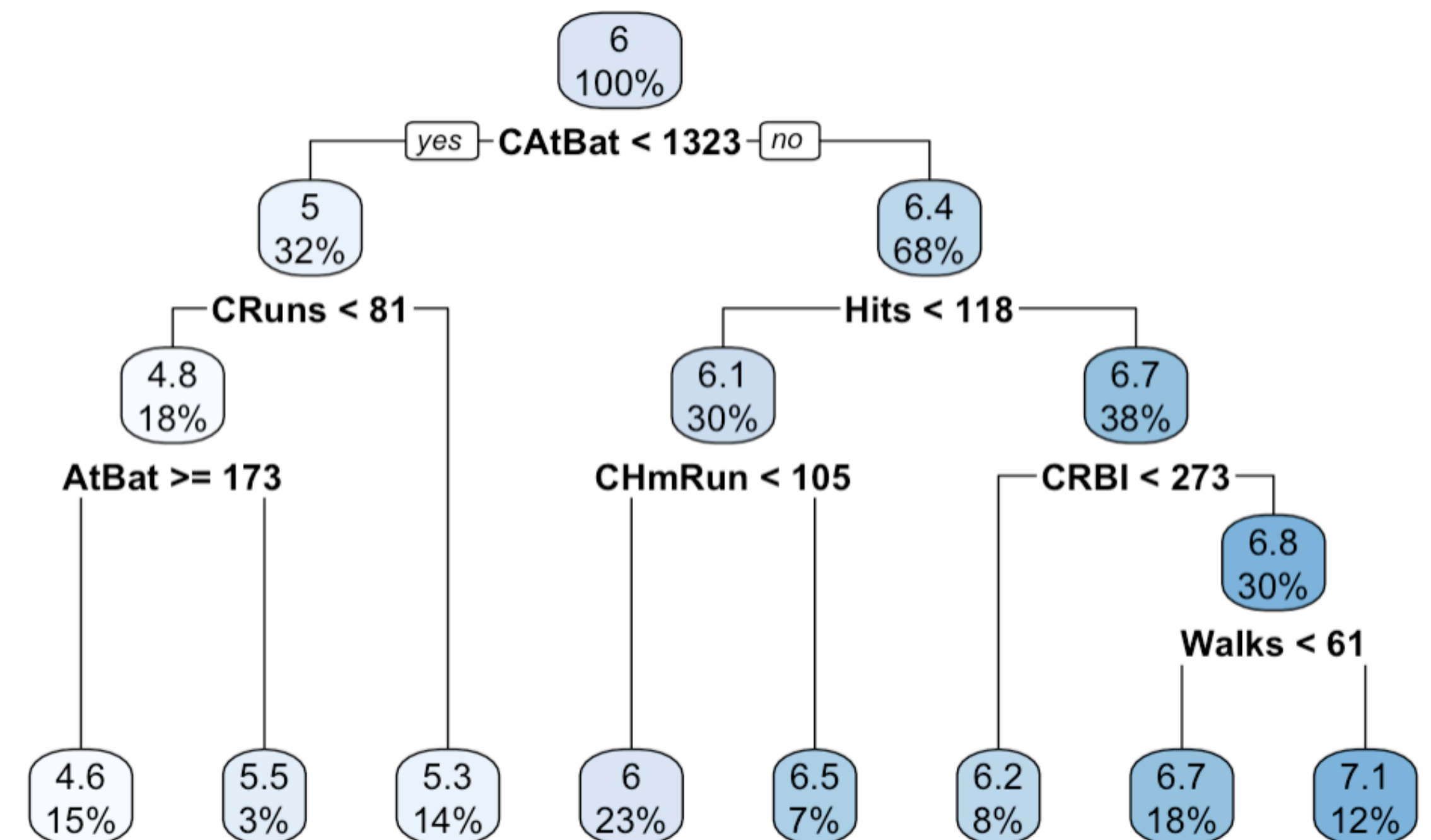
**Lecture 4:** Boosting

**Lecture 5:** Unit review and quiz in class

Homework 4 due the following **Wednesday**.

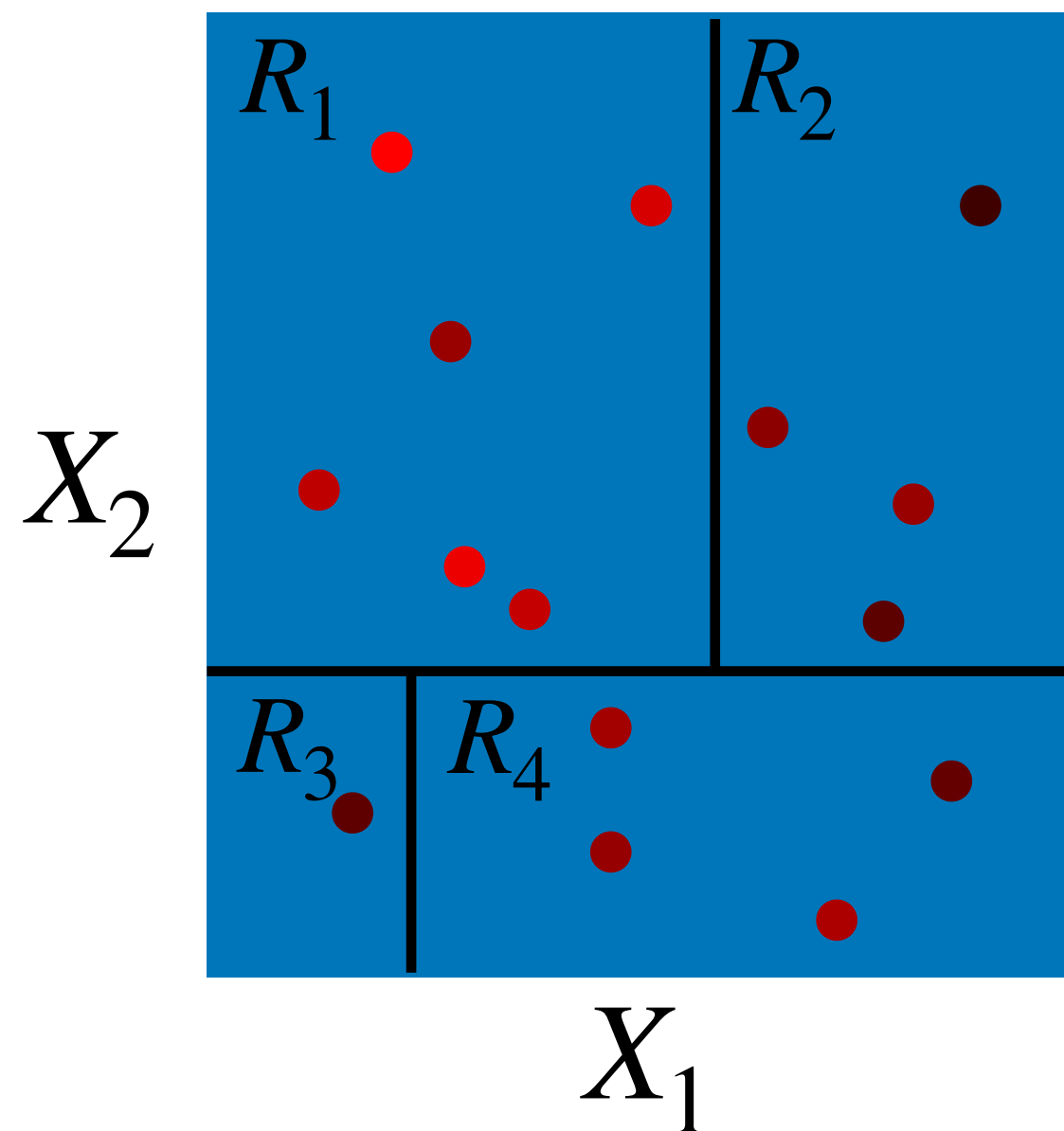
# Recall: Decision trees

- Create a partition of feature space by recursively splitting on different features
- Regression and classification trees
- Terminal nodes in the tree correspond to the rectangles in the partition
- Predict a single number (category) for each terminal node in a regression (classification) tree

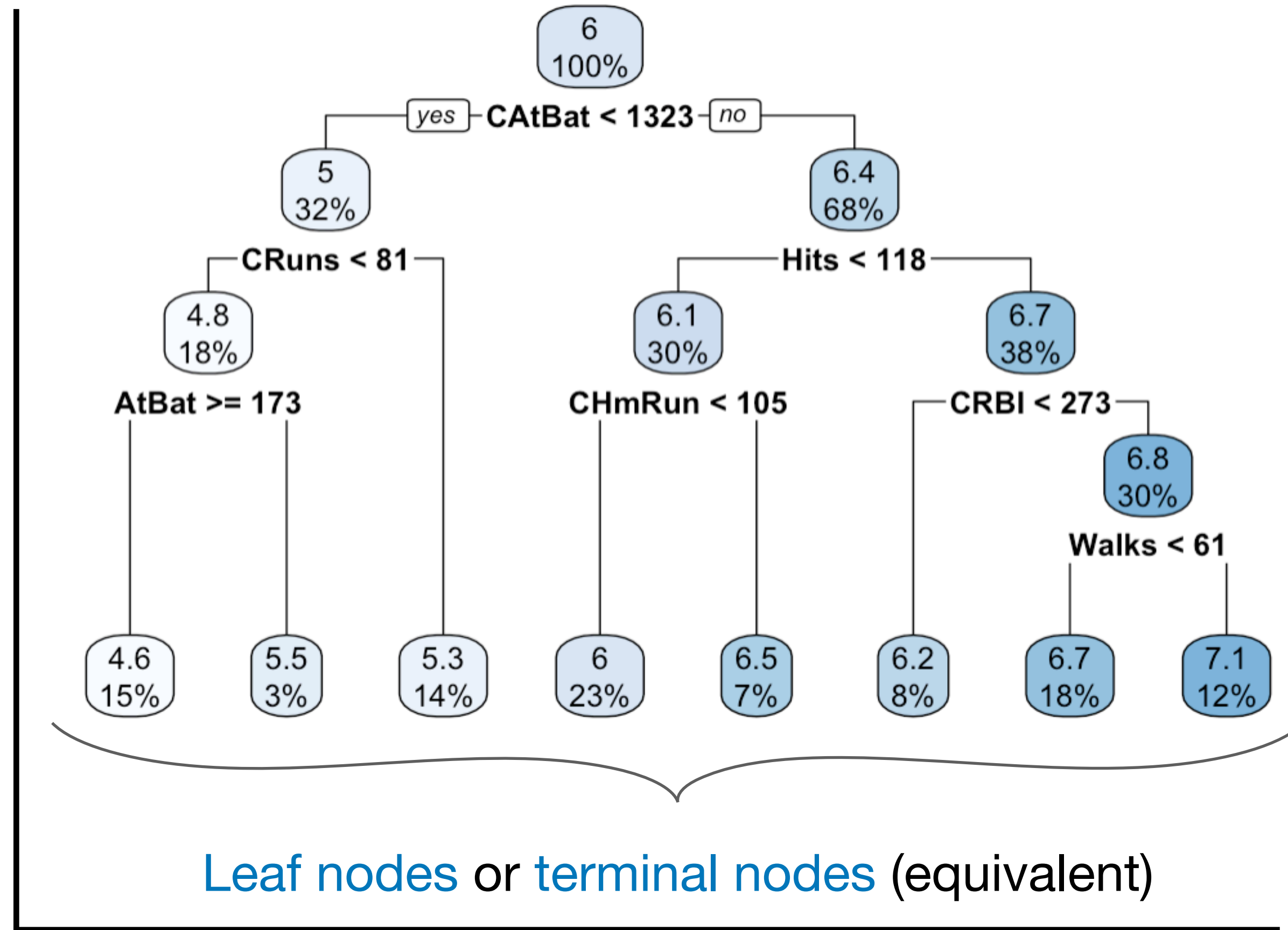
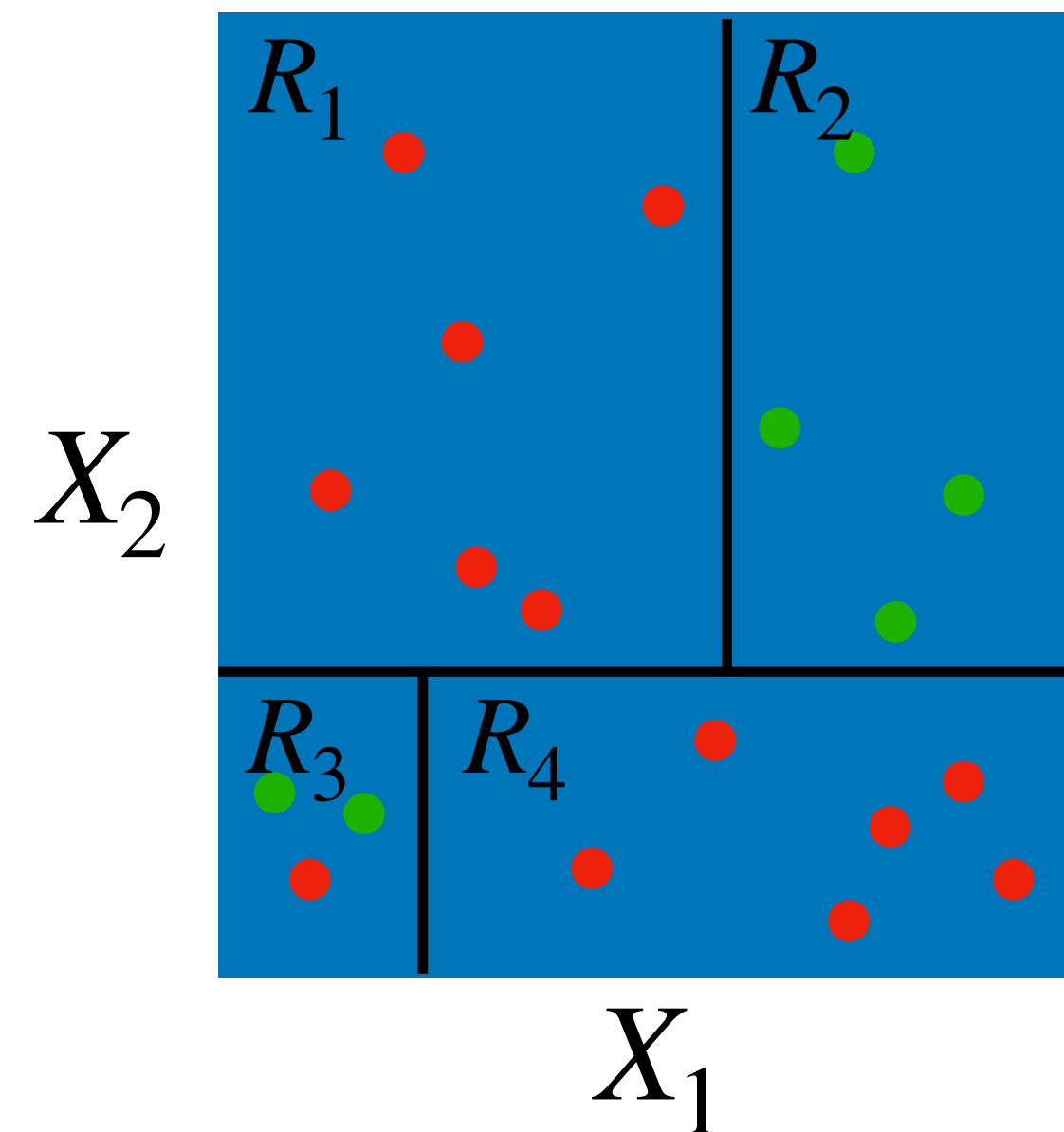


# Clarifications from last time

Regression tree



Classification tree



$$\text{Regression: Total RSS} = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 = \sum_{i \in R_1} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_2} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_3} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_4} (Y_i - \hat{Y}_i)^2$$

$$\text{Classification: Total Gini} = n_1 \cdot 2\hat{p}_1(1 - \hat{p}_1) + n_2 \cdot 2\hat{p}_2(1 - \hat{p}_2) + n_3 \cdot 2\hat{p}_3(1 - \hat{p}_3) + n_4 \cdot 2\hat{p}_4(1 - \hat{p}_4)$$

# Complexity of a decision tree

The more terminal nodes (regions), the more flexibly the tree fits training data:

- if there are as many terminal nodes as training points, training error = 0
- If there is just one terminal node, we are fitting a constant model

As with any prediction method, there is a bias-variance tradeoff based on model complexity.

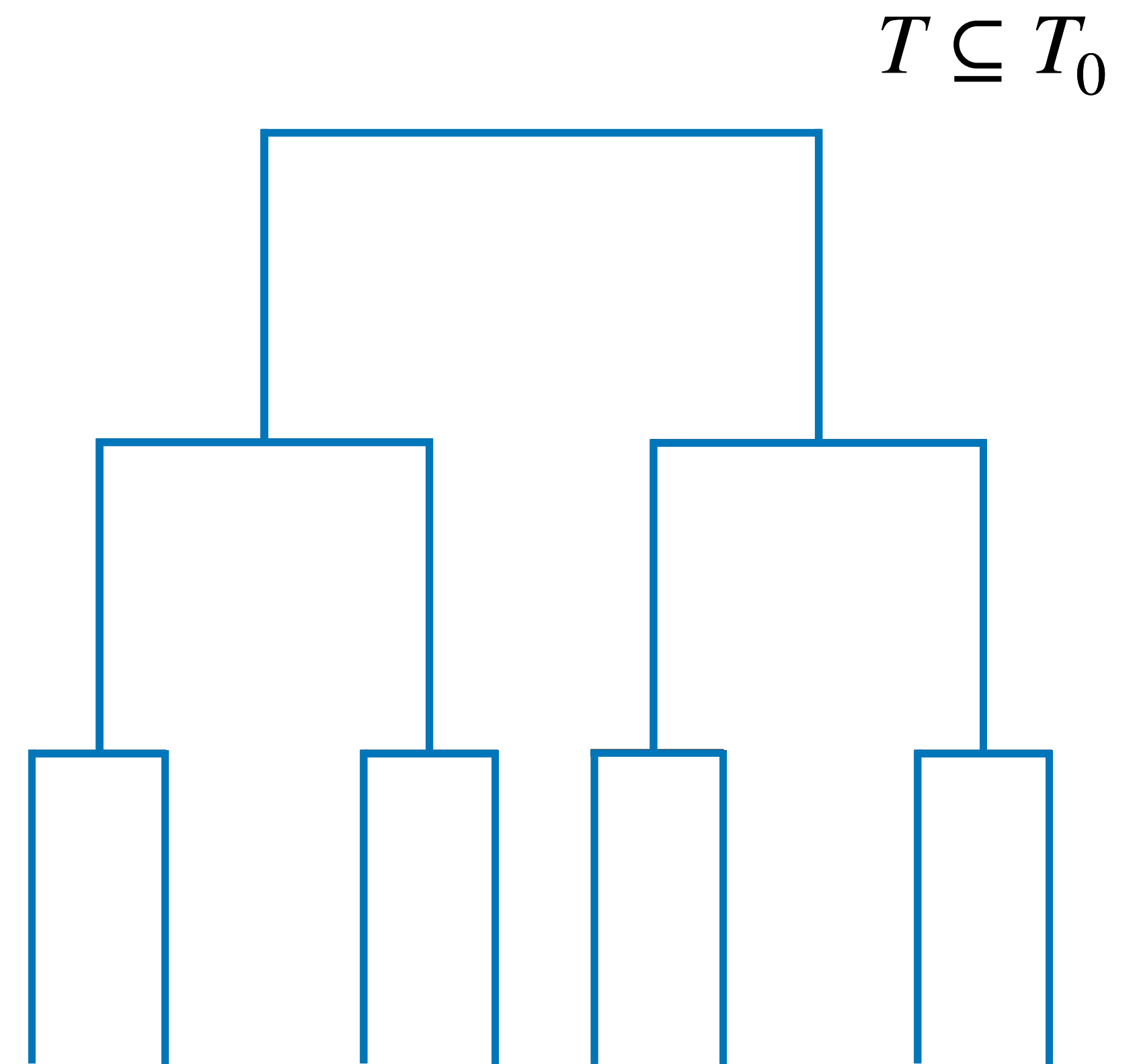
How to choose the best model complexity? [Cross-validation](#).

# A family of decision trees of varying complexity

- First grow out our tree about as far as we can to obtain a big tree  $T_0$ .
- We can then consider any subtree  $T \subseteq T_0$ .

Note: There are several subtrees  $T$  for each complexity value.

In other model selection scenarios, we have just had one model for each complexity.



# Imposing a penalty for terminal nodes

Let  $|T|$  be number of terminal nodes in tree  $T$ . Fixing some  $\alpha \geq 0$ , consider

$$T_\alpha = \arg \min_{T \subseteq T_0} \text{RSS}(T) + \alpha |T|.$$

Like lasso, varying  $\alpha$  leads to sequence of trees; higher  $\alpha$  leads to smaller trees.

Interpretation of  $\alpha$ : RSS should decrease by at least  $\alpha$  to make a split worth it.

Unlike lasso, there is a discrete set of  $\alpha$  values  $\alpha_1 < \alpha_2 < \dots < \alpha_M$  giving all possible solutions as  $\alpha$  varies:  $T_{\alpha_m}$  is the tree with  $m$  terminal nodes with lowest RSS.



# Cost complexity pruning

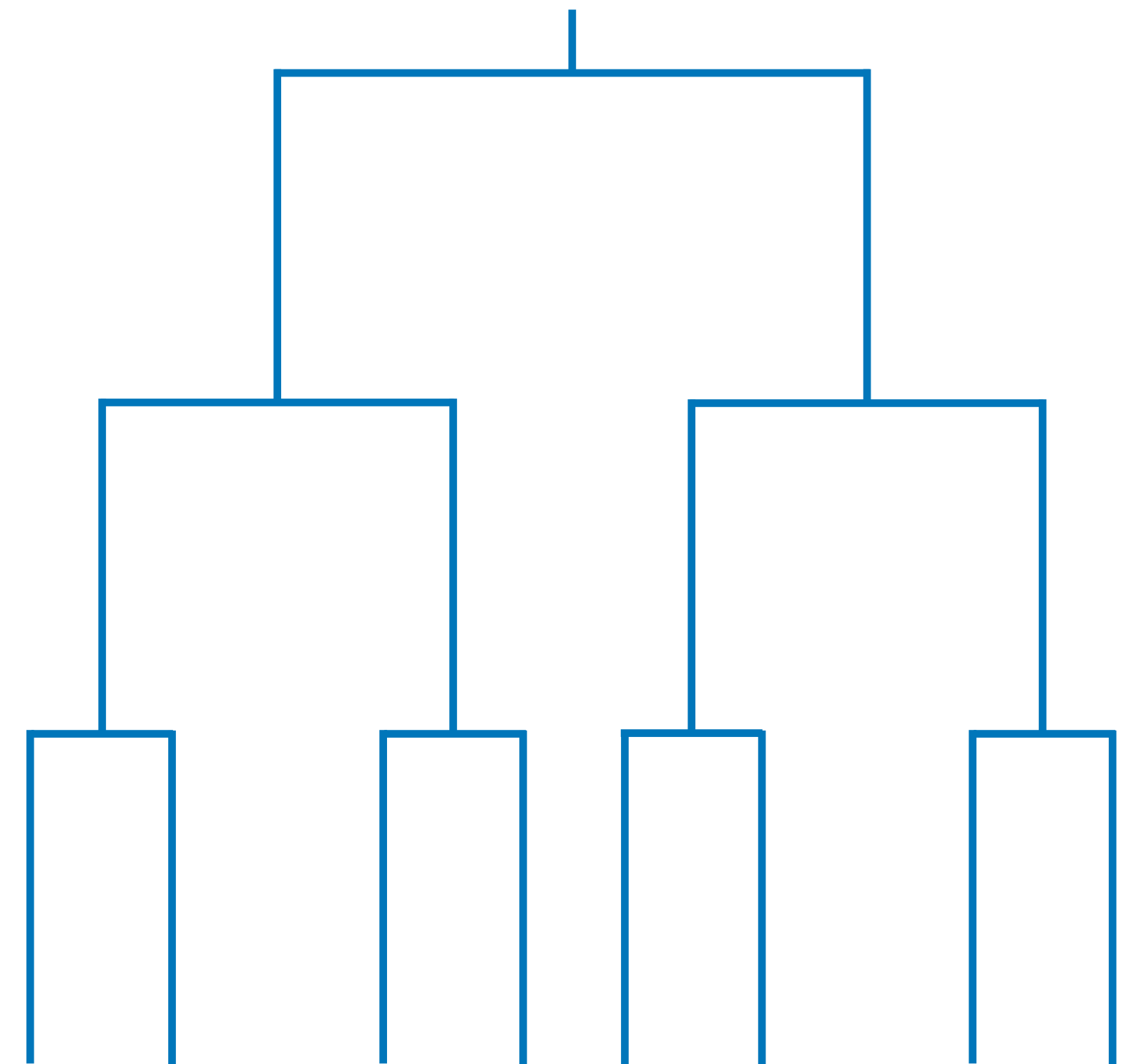
## Finding the sequence of trees $T_{\alpha_m}$

Given a fully grown tree  $T_0$ , **cost complexity pruning** is an algorithm that provably finds optimal sequence

$$T_0 \supseteq T_{\alpha_1} \supseteq T_{\alpha_2} \supseteq \cdots \supseteq T_{\alpha_M},$$

which turns out to be nested:

- Set  $T_{\alpha_1} = T_0$ , the fully grown tree
- Find the **weakest link**: the split in  $T_{\alpha_1}$  that reduces cost (i.e. RSS or misclassification error) the least.
- Set  $T_{\alpha_2}$  equal to  $T_{\alpha_1}$  but without the weakest link
- Repeatedly remove weakest link until arriving at  $T_{\alpha_M}$ , the tree with one terminal node.





# Cross-validation

## To find the optimal $\alpha$ for prediction

- Grow a full tree  $T_0$  on the whole training data
- Prune the tree to get a sequence  $T_0 \supseteq T_{\alpha_1} \supseteq T_{\alpha_2} \supseteq \dots \supseteq T_{\alpha_M}$
- Split the training samples into  $K$  folds
- For each fold  $k$ ,
  - grow a full tree  $T_0^{-k}$  on the out-of-fold data
  - find the sequence  $T_0 \supseteq T_{\alpha_1}^{-k} \supseteq T_{\alpha_2}^{-k} \supseteq \dots \supseteq T_{\alpha_M}^{-k}$  by pruning (but using same  $\alpha_m$ )
  - using these trees, make predictions  $\hat{Y}_i^m$  for each in-fold observation  $i$  and each  $m$
- Find CV estimates and standard errors as usual; choose  $\hat{\alpha}$  based on 1-standard-error rule
- Output the final decision tree  $T_{\hat{\alpha}}$

# Tree growing versus tree pruning

Growing proceeds from smaller to larger trees; pruning from larger to smaller.

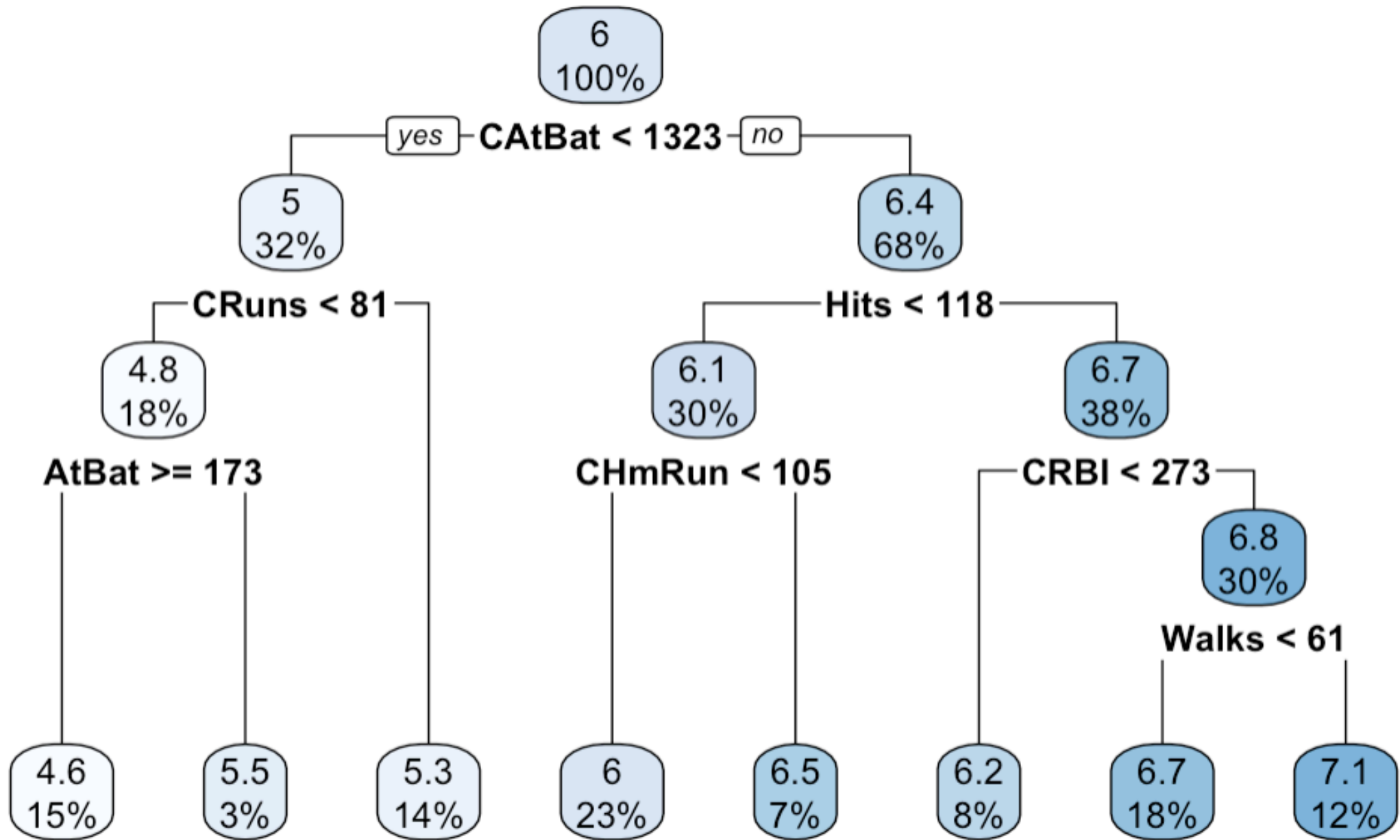
For regression trees, growing and pruning are both based on RSS.

For classification trees, growing based on Gini index but pruning based on misclassification error.

Growing and pruning both define a sequence of trees, but it may not be the same sequence. The sequence of trees for growing not used except to get the big tree  $T_0$ ; the sequence of trees for pruning is the one used for cross-validation.

# Summary: Decision Trees

- Nonlinear method for regression or classification based on recursive partitioning of feature space
- Trees grown in greedy top-down fashion, choosing feature and split point to maximally improve purity of terminal nodes.
- The complexity of a tree increases with the number of terminal nodes.
- A sequence of trees of varying complexities obtained from cost complexity pruning of a maximally-grown tree.
- Final tree chosen by cross-validation on penalty parameter  $\alpha$ .



Pros	Cons
Easily interpretable	Tree structure very sensitive to training data
Captures non-linear relationships	High variance predictions; suboptimal predictive performance

# How can we reduce the variance of trees?

When it comes to prediction accuracy, trees suffer because of their high variance.

Here's an idea for how we can obtain a prediction method with lower variance:

- “Shake up” the training data lots of times (bootstrap)
- For each version of the training data, fit a different tree
- Use the average of all these trees to make predictions (aggregation)

Bagging = Bootstrap Aggregation.

**Intuition:** By averaging a bunch of trees, we are reducing the variance while keeping the bias about the same. This should yield better predictive performance!

# What does it mean to “shake up” the training data?

What we ideally would have wanted is to get many different random realizations of the training data, on which we could fit different trees.

We only get one realization of the training data, but we can still get different random versions of our data by **bootstrapping**:

A **bootstrap sample** is a new data set with the same number of observations, generating by sampling observations from the original data **with replacement**.

The idea is that your bootstrap samples are slightly different versions of your training data, allowing you to fit different trees to these different training sets.

# The bootstrap: An example

Original training data

Observation ID	$X$	$Y$
1	$X_1$	$Y_1$
2	$X_2$	$Y_2$
3	$X_3$	$Y_3$
4	$X_4$	$Y_4$
5	$X_5$	$Y_5$

Bootstrap sample 1

Observation ID	$X$	$Y$
5	$X_5$	$Y_5$
3	$X_3$	$Y_3$
2	$X_2$	$Y_2$
3	$X_3$	$Y_3$
1	$X_1$	$Y_1$

⋮

⋮

Bootstrap sample  $B$

Observation ID	$X$	$Y$
4	$X_4$	$Y_4$
1	$X_1$	$Y_1$
1	$X_1$	$Y_1$
5	$X_5$	$Y_5$
4	$X_4$	$Y_4$



# Bagging

