CME 250: Introduction to Machine Learning

Lecture 6: Decision Trees and Random Forests

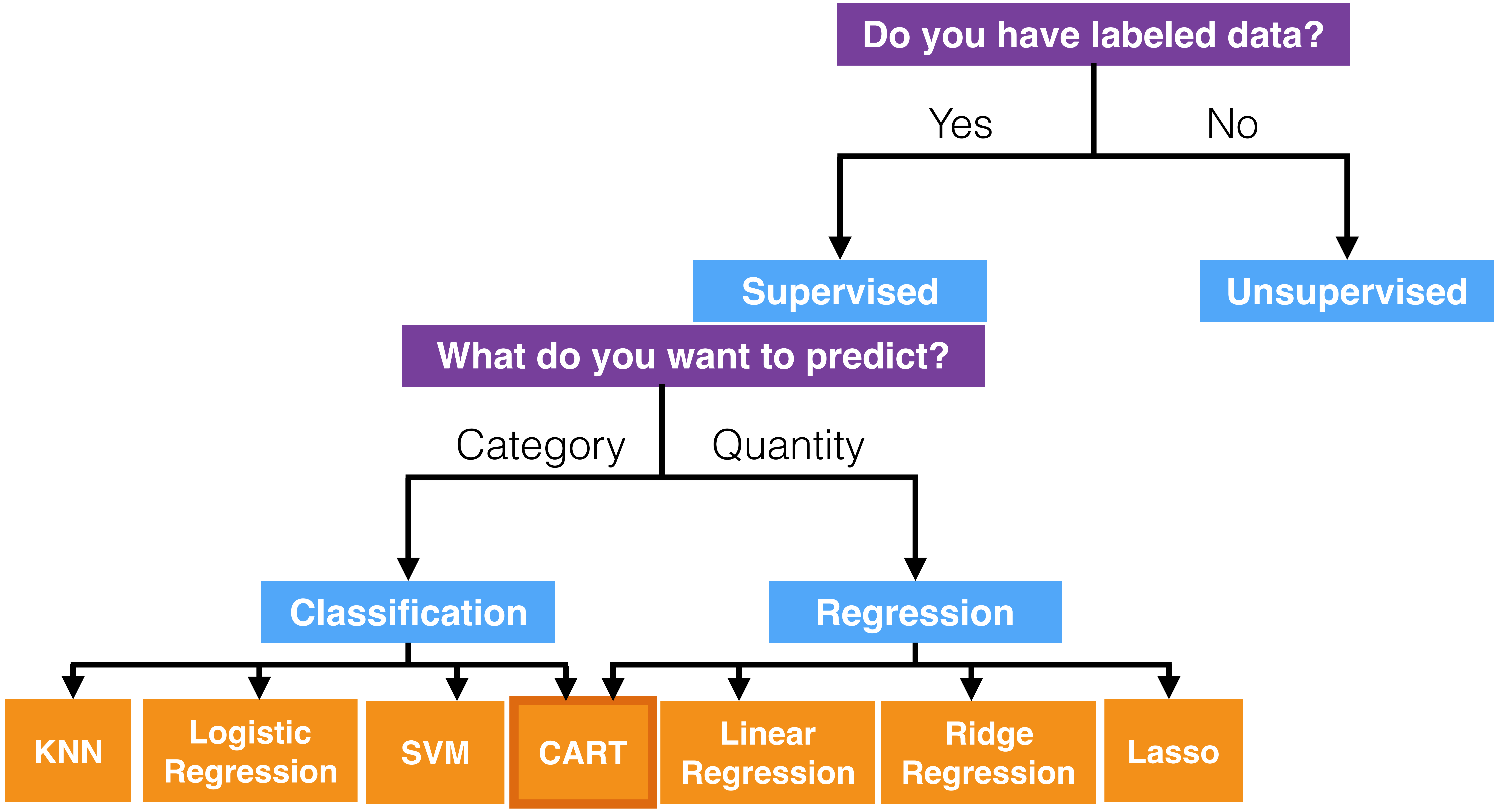
Sherrie Wang
sherwang@stanford.edu

Agenda

Slides are online at
cme250.stanford.edu

- Decision Trees (CARTs)
- Bagging
- Random Forests

Machine Learning Methods



Decision Trees

Tree-based Methods

Stratify or *segment* the feature space into regions.

To make a prediction for an observation, use the mean or mode of the training observations in the region to which it belongs.

The decisions used to segment the feature space can be represented as a tree, hence *decision trees*.

Can be used for regression or classification.

Example & Terminology

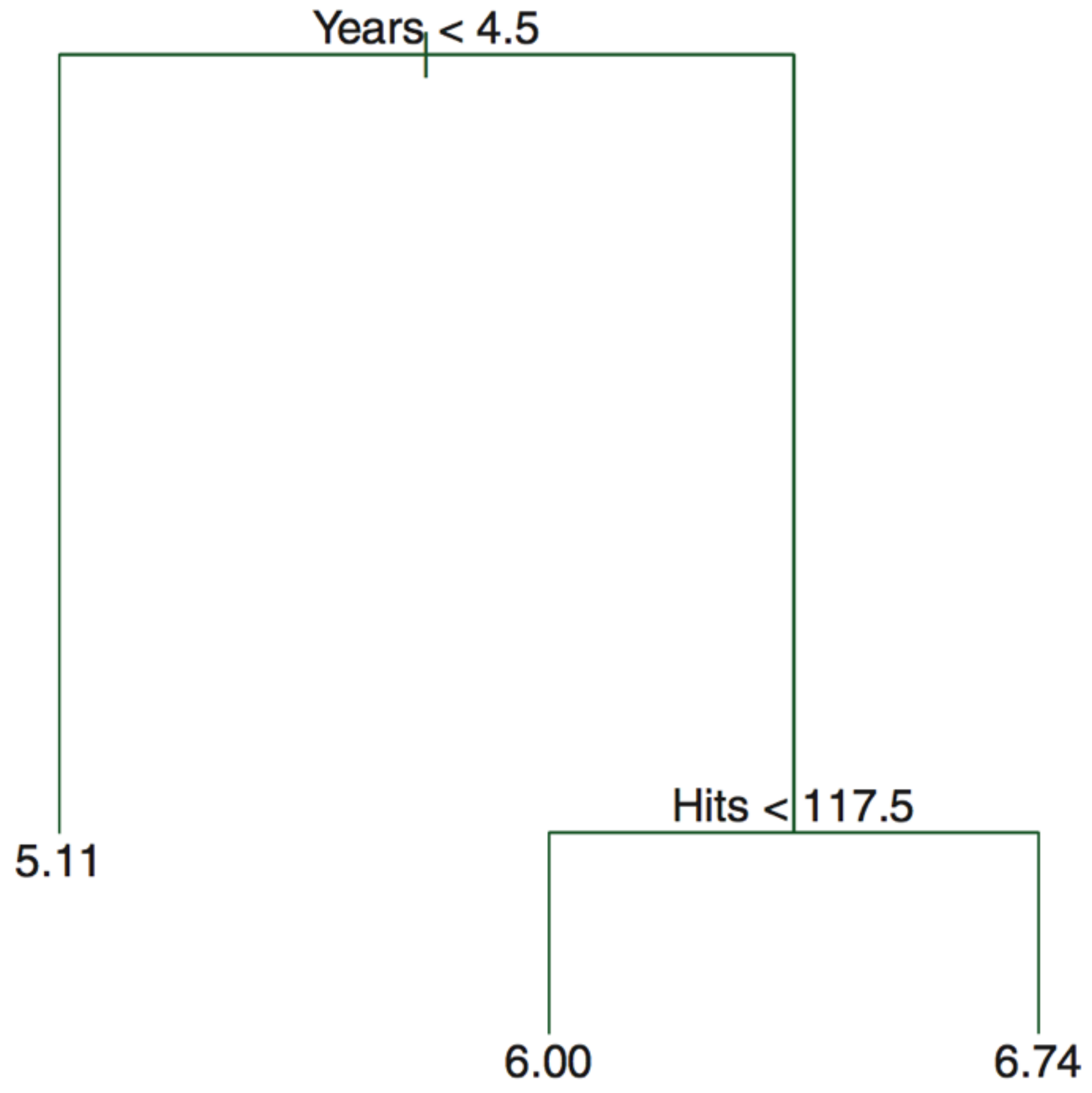


FIGURE 8.1, ISL (8th printing 2017)

Here's an example **regression tree**. It predicts a baseball player's log(salary) from their number of years in the MLB and number of hits.

Example & Terminology

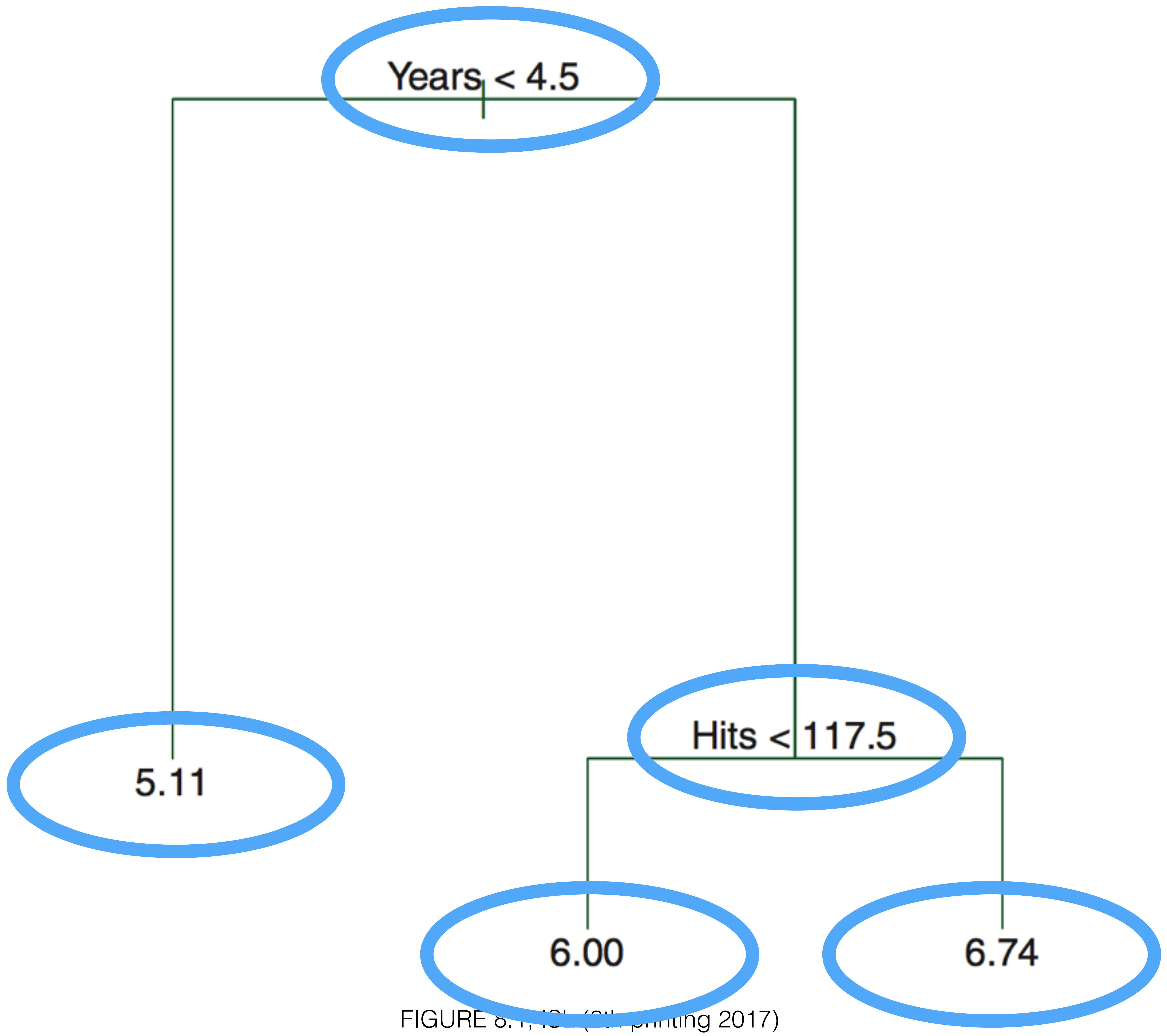


FIGURE 8.1, ISL (6th printing 2017)

Trees are a collection of **nodes**. There are 2 types of nodes:

1. Internal nodes
2. Terminal nodes (leaves)

Example & Terminology

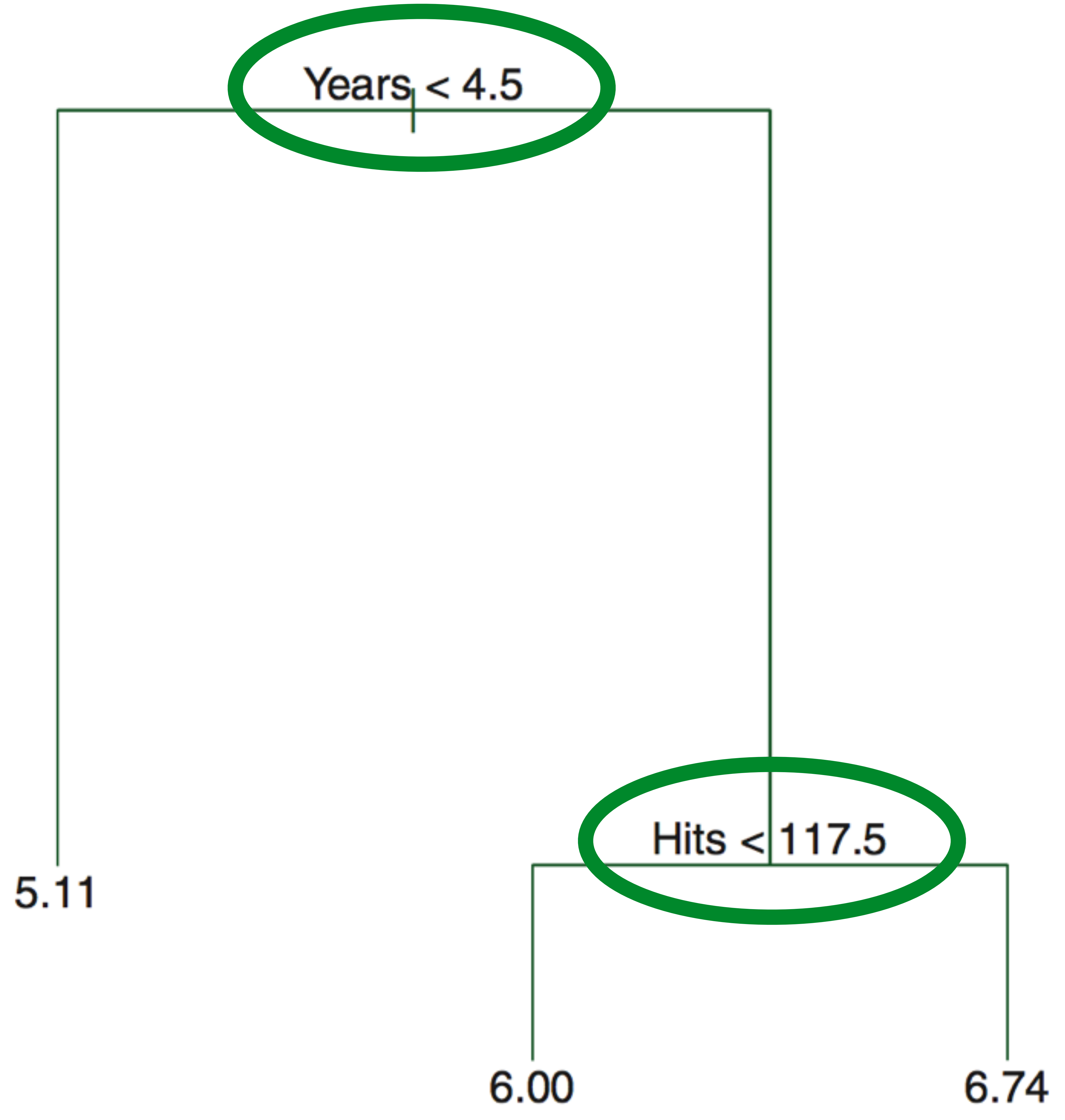


FIGURE 8.1, ISL (8th printing 2017)

Internal nodes split the feature space.

Example & Terminology

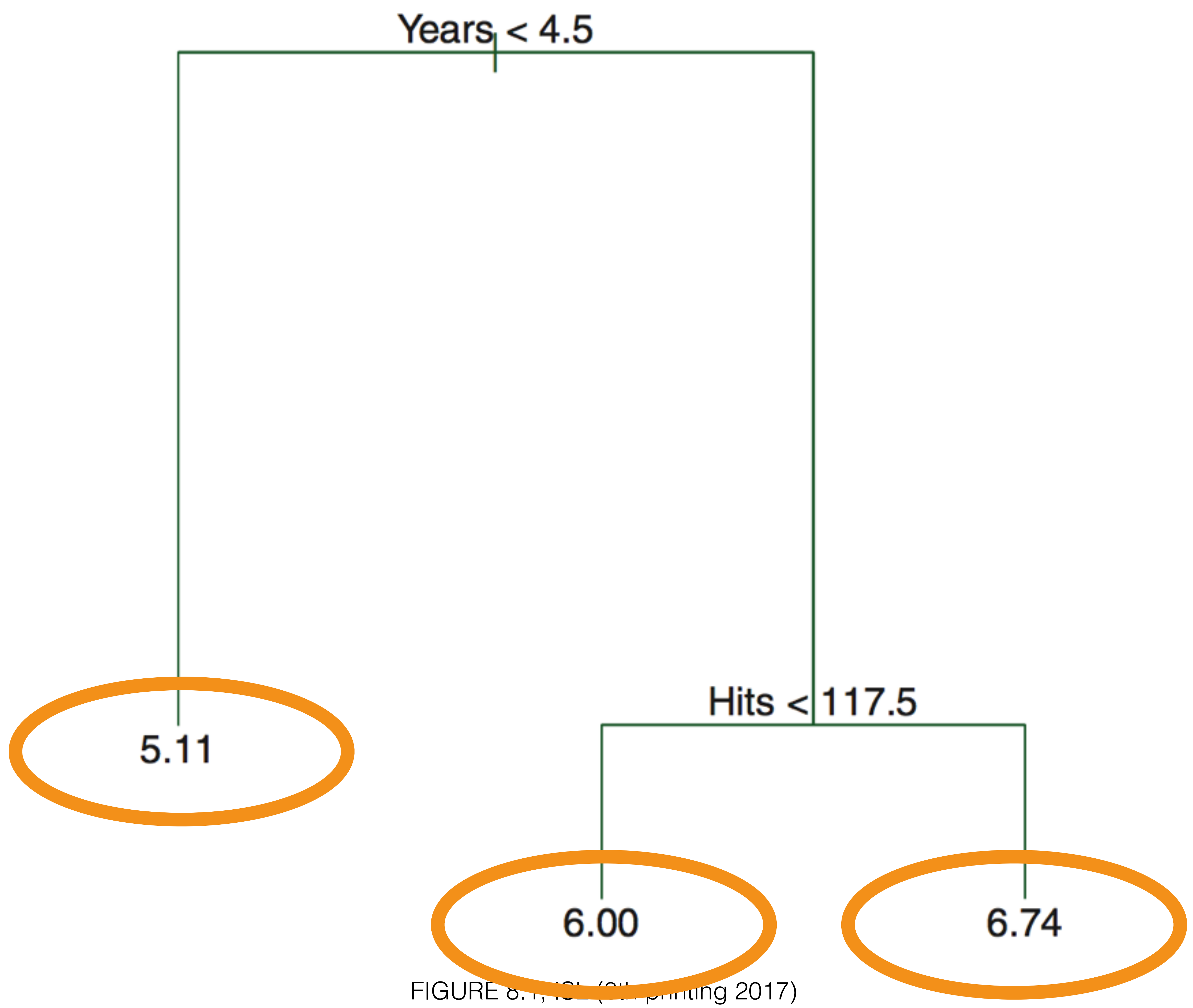


FIGURE 8.1, ISL (8th printing 2017)

Terminal nodes (leaves) show predictions for the split regions of the feature space.

Predicted salary is the mean salary of the players in the training set in that region.

Example & Terminology

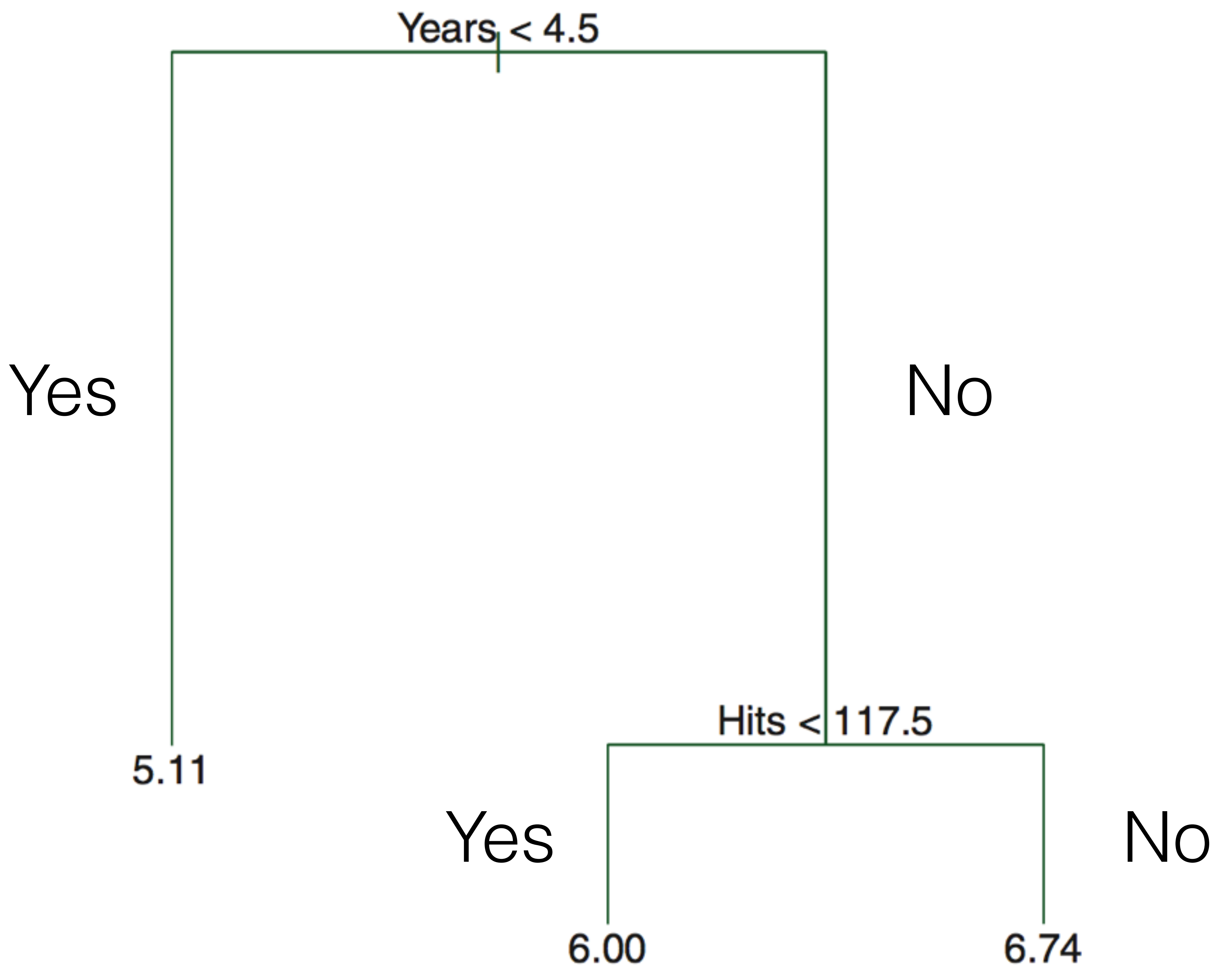


FIGURE 8.1, ISL (8th printing 2017)

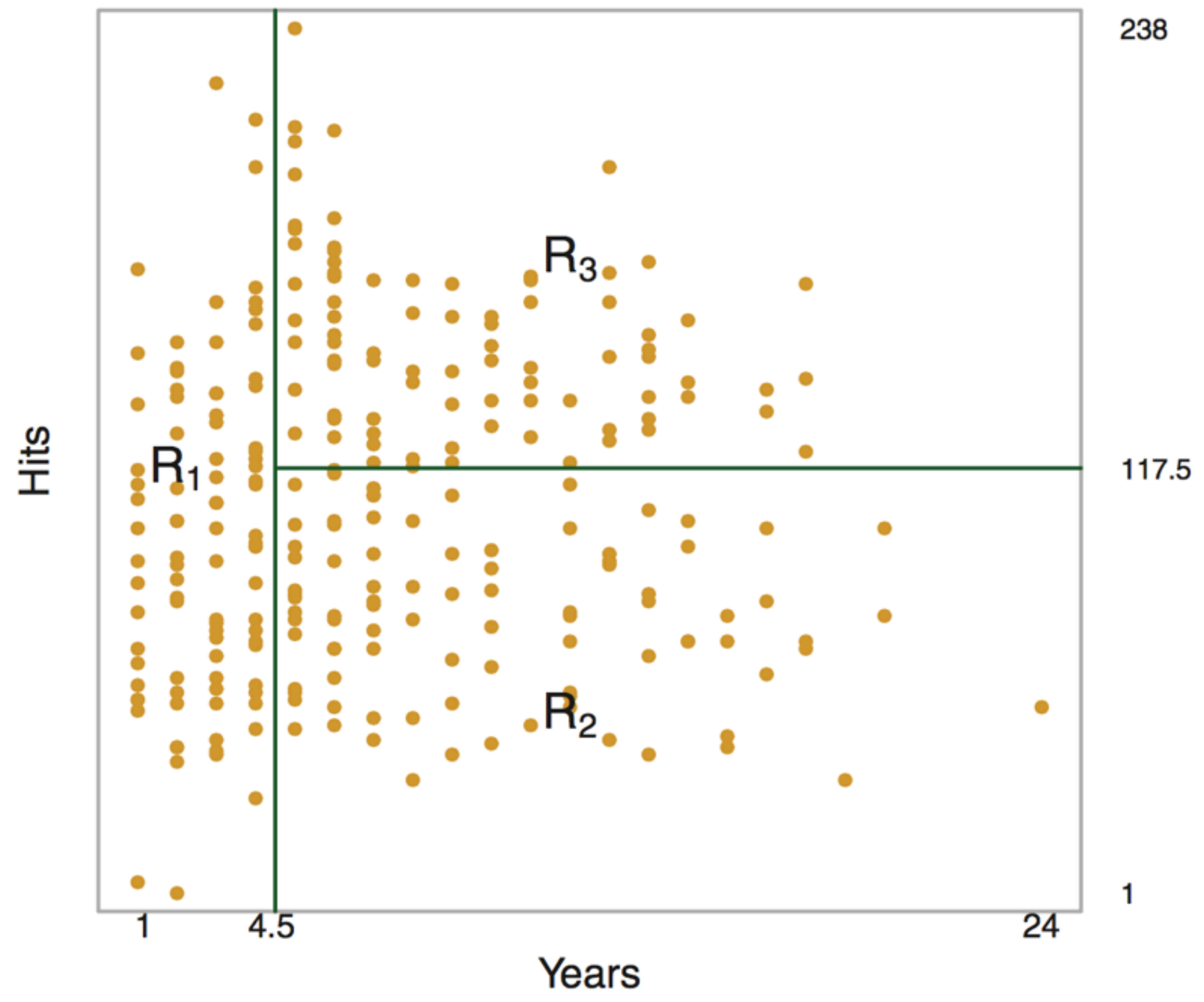


FIGURE 8.2, ISL (8th printing 2017)

Another Example

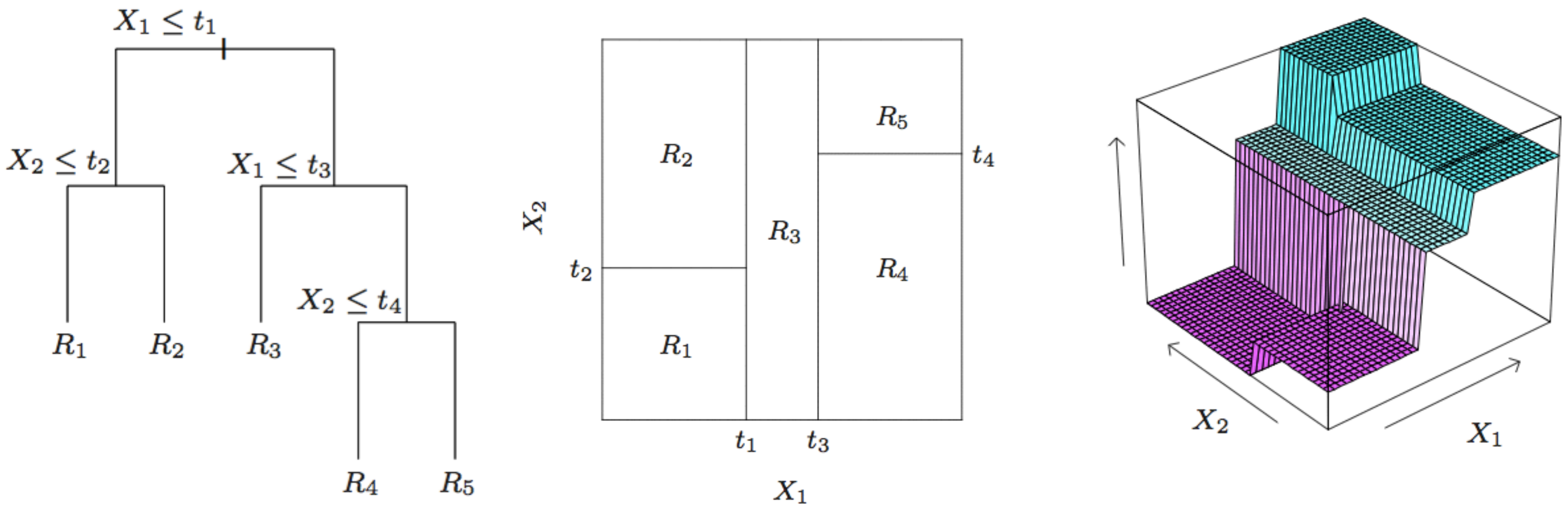


FIGURE 8.3, ISL (8th printing 2017)

Building a Regression Tree

Two steps:

1. Divide the feature space (i.e. the set of possible values for $X_1, X_2, \ldots, X_p$) into J distinct and non-overlapping regions $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, make the same prediction: the mean of the response values for the training observations in R_j.

Stratifying the Feature Space

How to construct the regions $R_1, R_2, \ldots, R_J$?

Let's limit ourselves to regions that are rectangular and axis-aligned (termed *boxes*).

We will try to minimize RSS: $$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \underbrace{\hat{y}_{R_j}}_{\text{mean response for training samples within } j\text{-th box}})^2$$

Even so, it is computationally infeasible to consider every possible partition of the feature space into J boxes.

Greedy Algorithm

Recursive binary splitting is a top-down region splitting approach.

1. Begin at the top of the decision tree. At this point, all points belong to a single region.

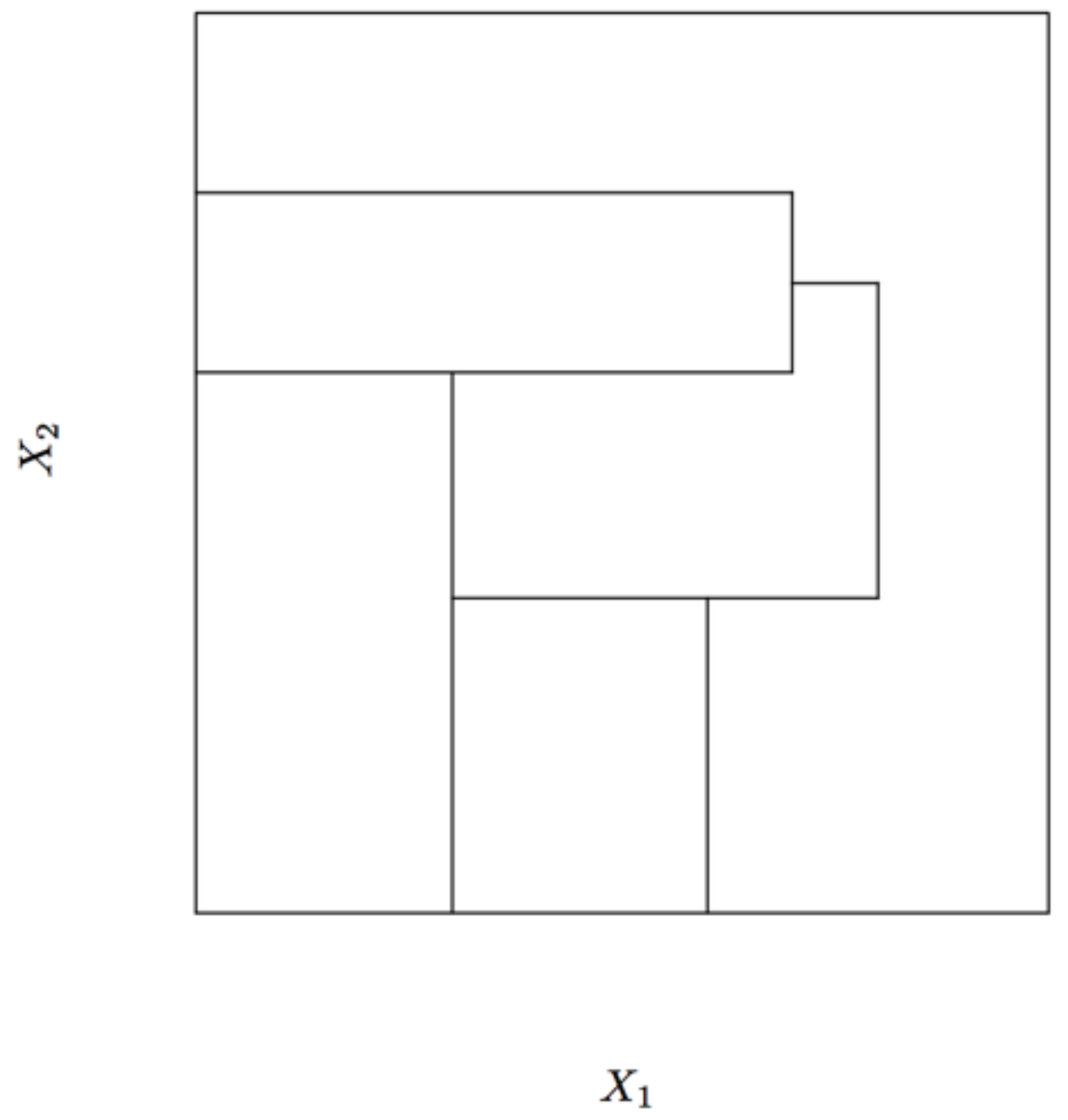
2. Find the splitting variable j and split point s that minimize the RSS most. Split here to create two regions where there used to be one.

3. Repeat this, looking for the best feature and best split point in each of the previously identified regions, until *stopping criterion* is met.

Termed "greedy" because it does not look ahead to future steps.

Note on Possible Partitions

Recursive binary splitting does limit the possible partitions of feature space that can be achieved. The left partition is impossible under this algorithm.



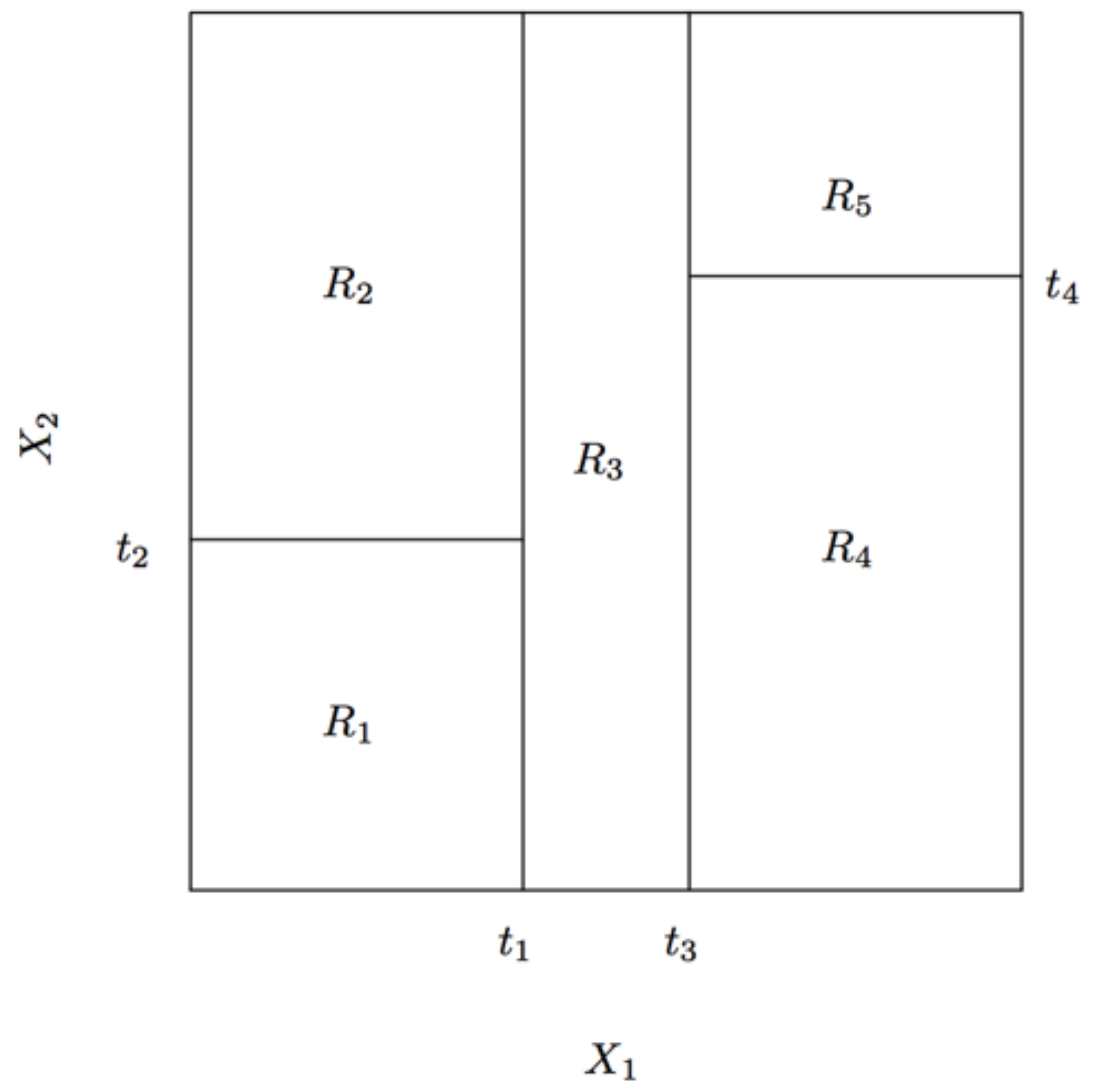


FIGURE 8.3, ISL (8th printing 2017)

Stopping Criteria

Imagine a tree that didn't stop until each leaf contained only one training sample. This would potentially be a very large (computationally expensive) tree, and not generalize well to new data.

Stopping criteria can limit the size of trees and control overfitting. Common ones include:

- Number of samples in leaf is less than some number.
- Purity of node is more than some number.
- Depth of node is more than some number.
- Response values are all identical.

Tree Pruning

Stopping criteria alone may not lead to high-performing trees. We could, for example, decide to stop splitting regions when RSS does not decrease much in a step. But it's possible that in a future step, it will decrease by a large amount.

Instead of limiting the tree severely when building the tree, grow a very large tree T_0 and then *prune* it back to obtain a *subtree*.

How to prune a tree?

Considering every possible subtree is computationally infeasible.

Cost complexity pruning (aka weakest link pruning): consider a sequence of trees indexed by a nonnegative tuning parameter α.

For each α, find the tree $T \subset T_0$ that minimizes

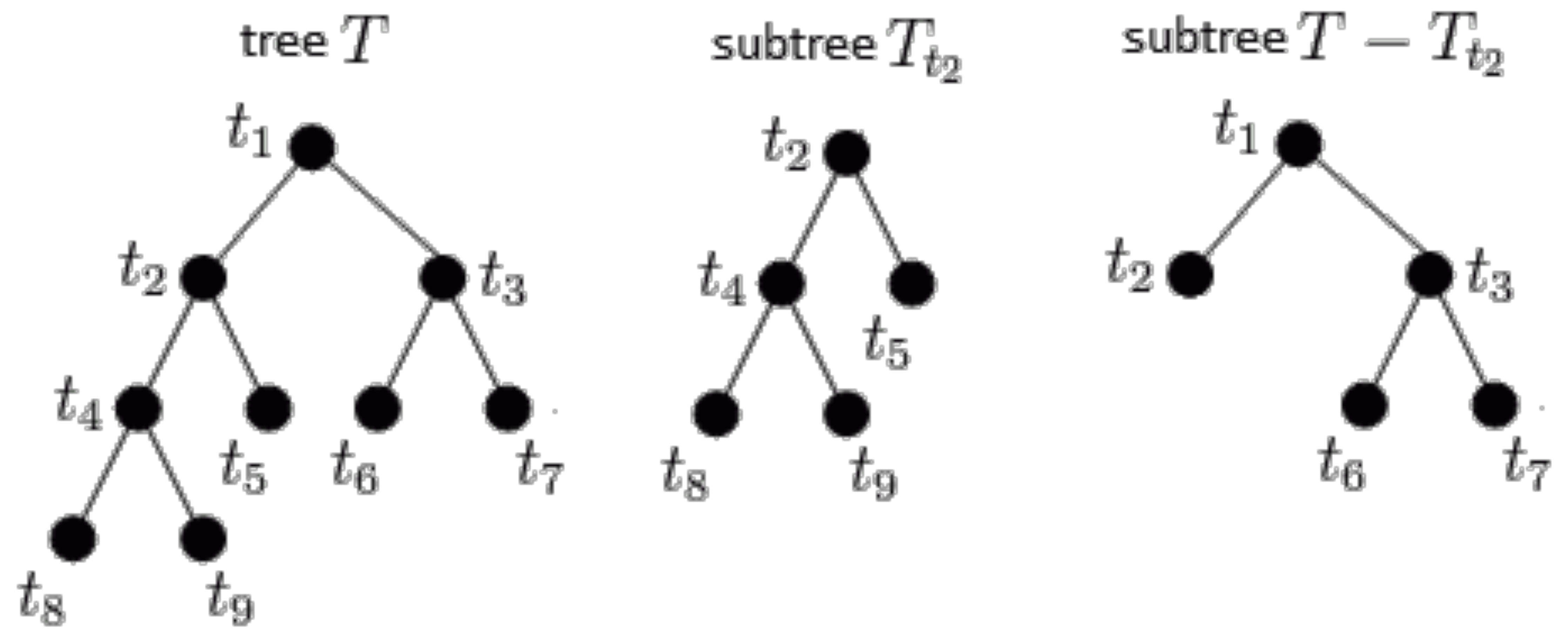
$$\sum_{m=1}^{|T|} \sum_{i:\ x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

number of terminal nodes of the tree T

Tree Pruning

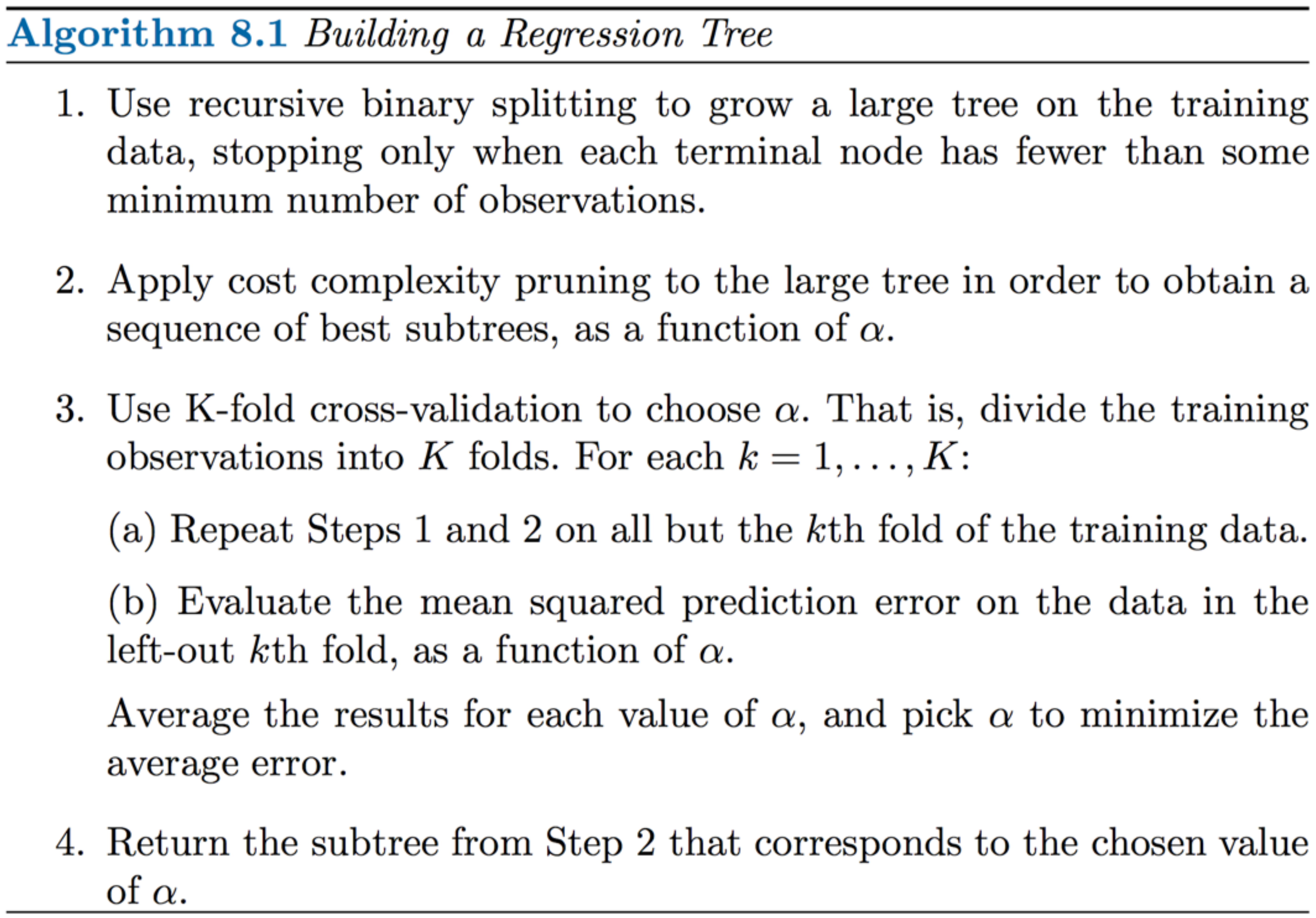
It turns out that as α increases, branches get pruned in a nested and predictable way. Cost complexity pruning is done by generating a sequence of subtrees T_0, T_1, …, T_m, where T_i is T_{i-1} with one node plucked away and T_m is the root node.

We choose the best α via cross-validation.



http://mlwiki.org/index.php/Cost-Complexity_Pruning

Regression Tree 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.

Selecting the Best Subtree

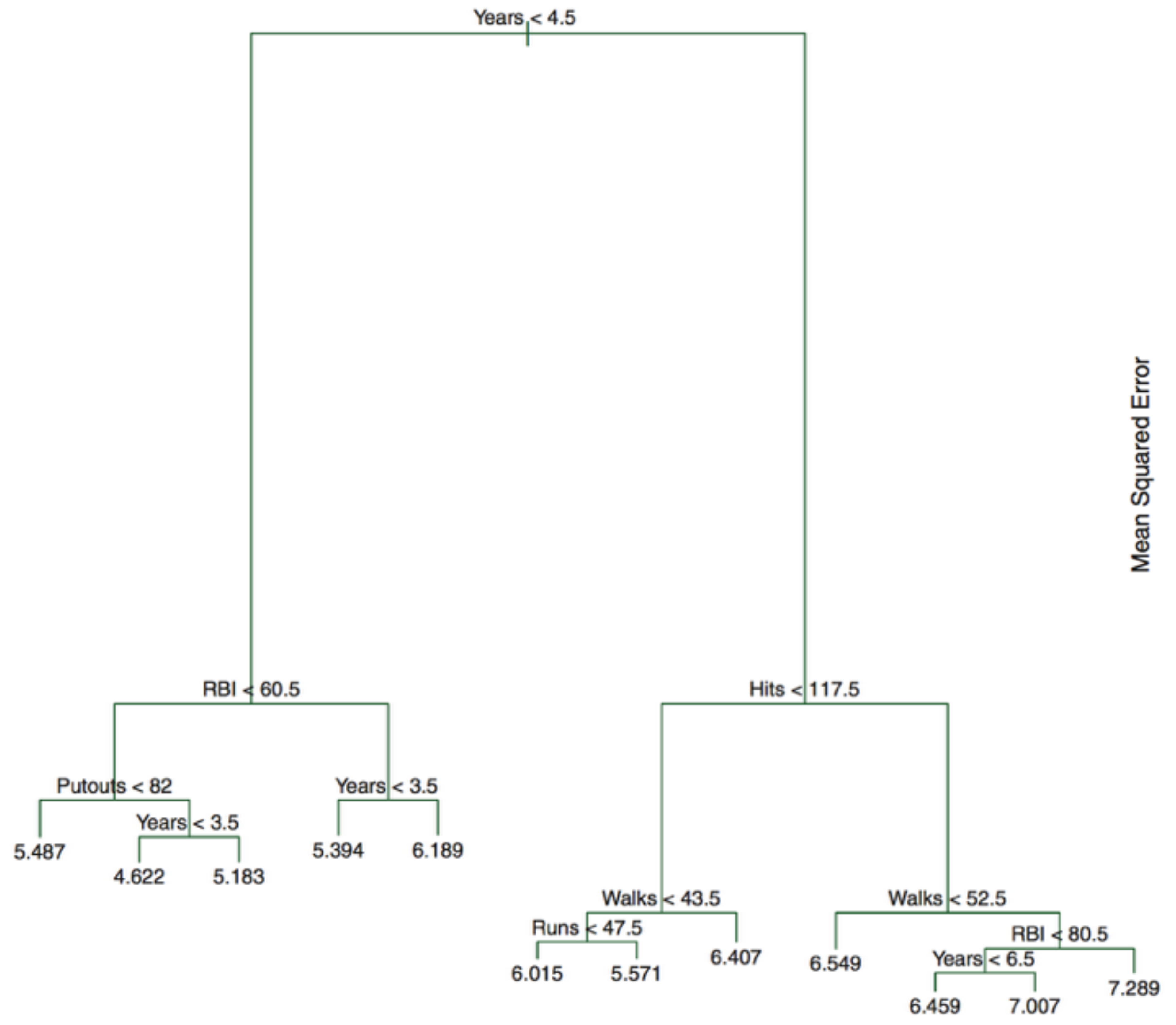


FIGURE 8.4, ISL (8th printing 2017)

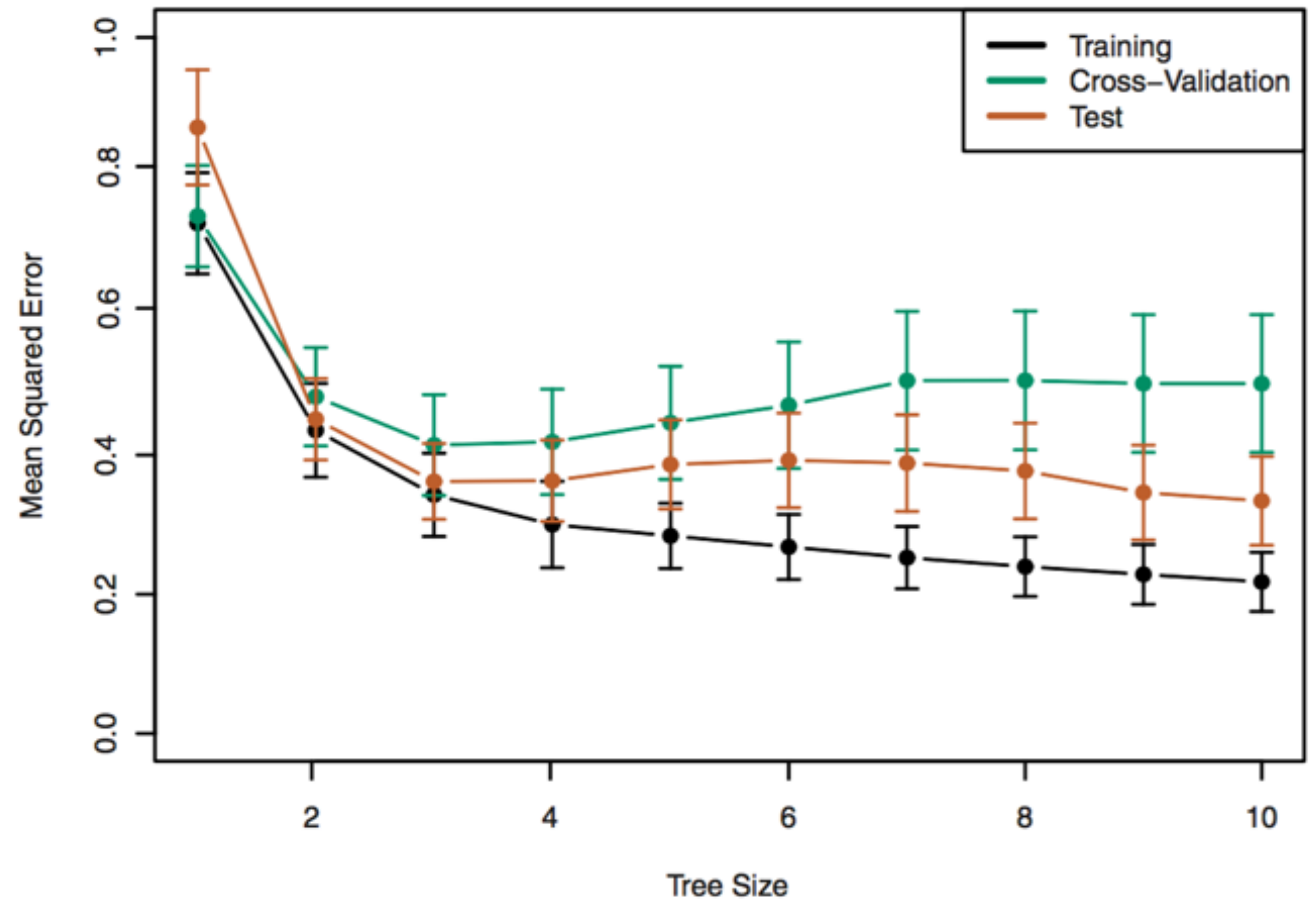


FIGURE 8.5, ISL (8th printing 2017)

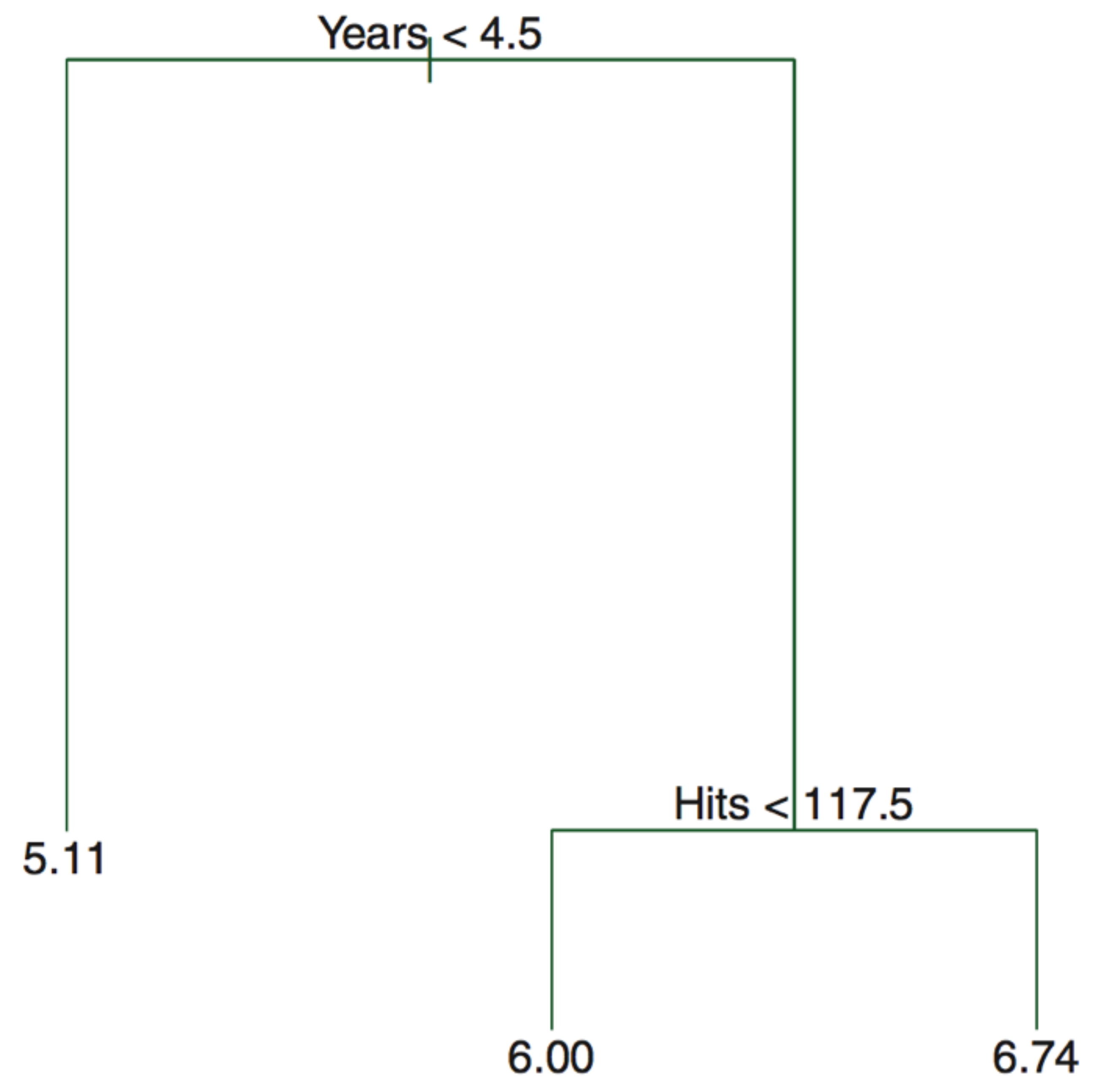


FIGURE 8.1, ISL (8th printing 2017)

Building a Classification Tree

Very similar to a regression tree.

1. Divide the feature space (i.e. the set of possible values for $X_1, X_2, \ldots, X_p$) into J distinct and non-overlapping regions $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, make the same prediction: the *mode* among the response classes for the training observations in R_j. (Class proportion can be interpreted as probabilities.)

Stratifying the Feature Space

On what criteria do we split the regions $R_1, R_2, \ldots, R_J$?

Recall that in regression we used RSS.

In classification we care about accuracy, but it turns out accuracy alone is not sensitive enough to build a good classification tree.

Two commonly used metrics to evaluate potential splits:

1. **Gini index**
2. **Information entropy**

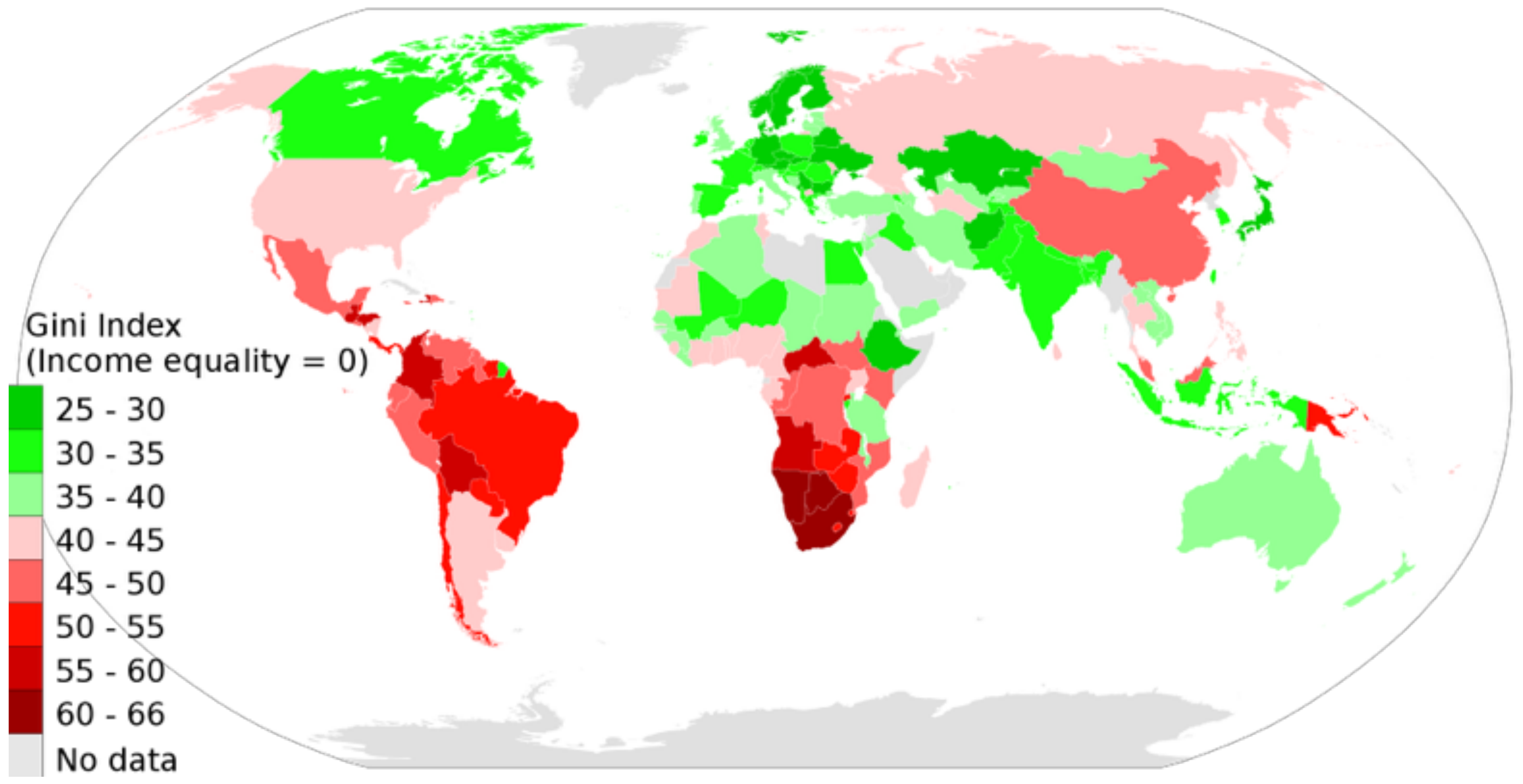
Gini Index

Defined as

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

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

Measures *node purity*. A small value means observations are mostly from a single class.



It is also a metric commonly used to measure income inequality.

Information Entropy

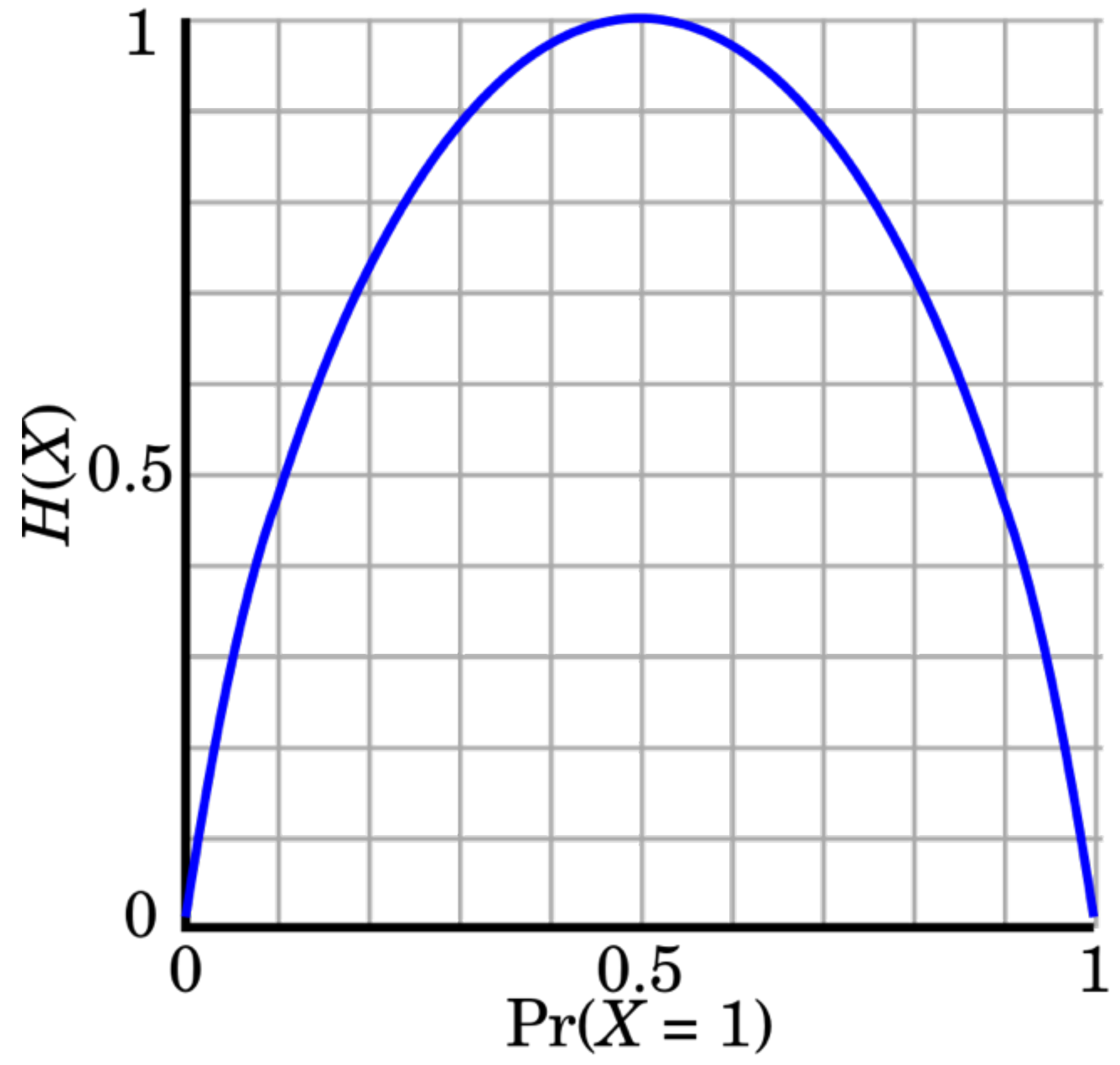
Defined as

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

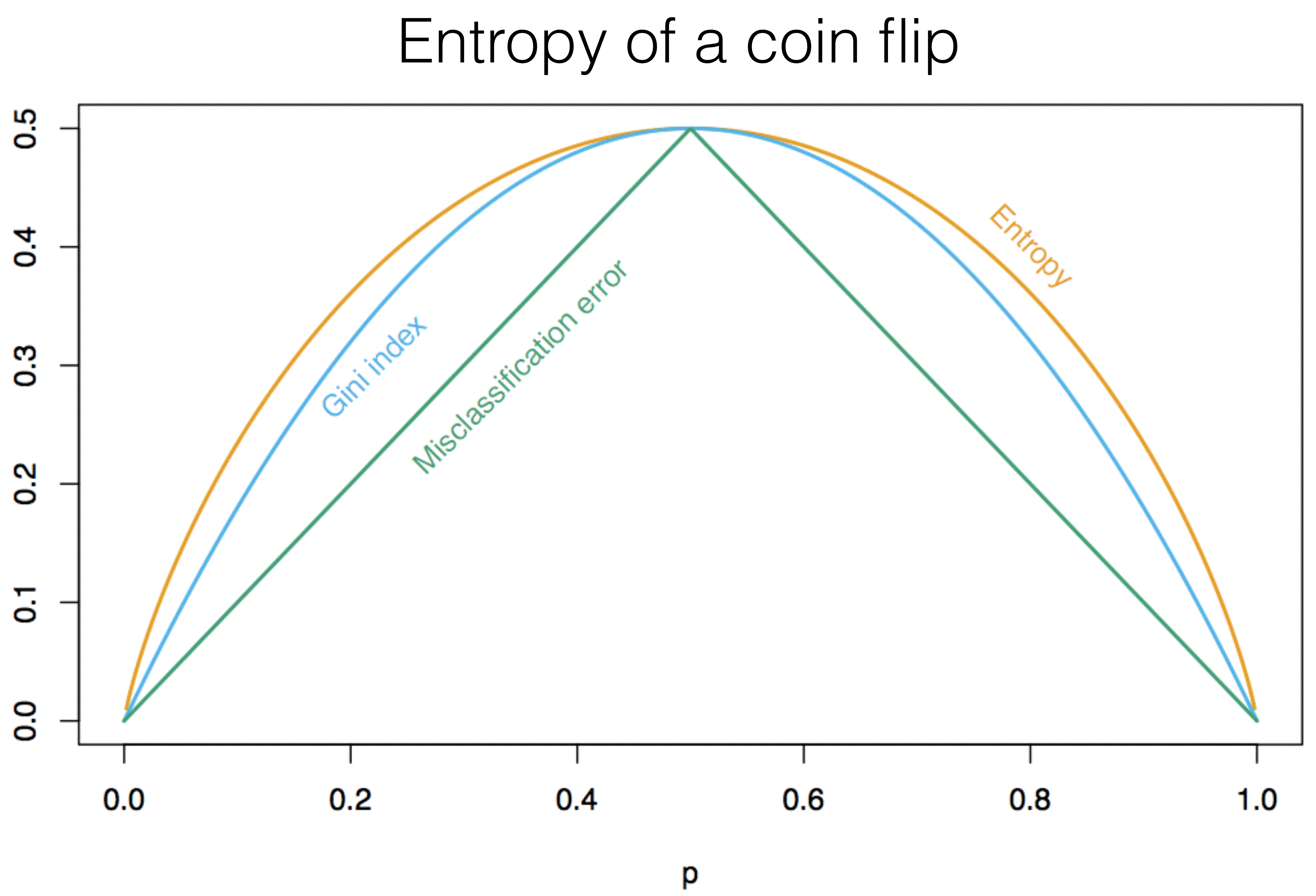
Entropy is a nonnegative value. If $\hat{p}_{mk}$ are close to 0 or 1, entropy will be near 0.

Also measures node purity. We look for the split that leads to the greatest drop in entropy.

Entropy of a coin flip



Gini Index vs. Entropy



Hastie, Trevor et al. Elements of Statistical Learning. Vol 2 No 1. New York: Springer, 2009.

The two metrics are quite similar, and both result in building better trees than misclassification error.

Example Classification Tree

Predict presence of heart disease (Yes, No)

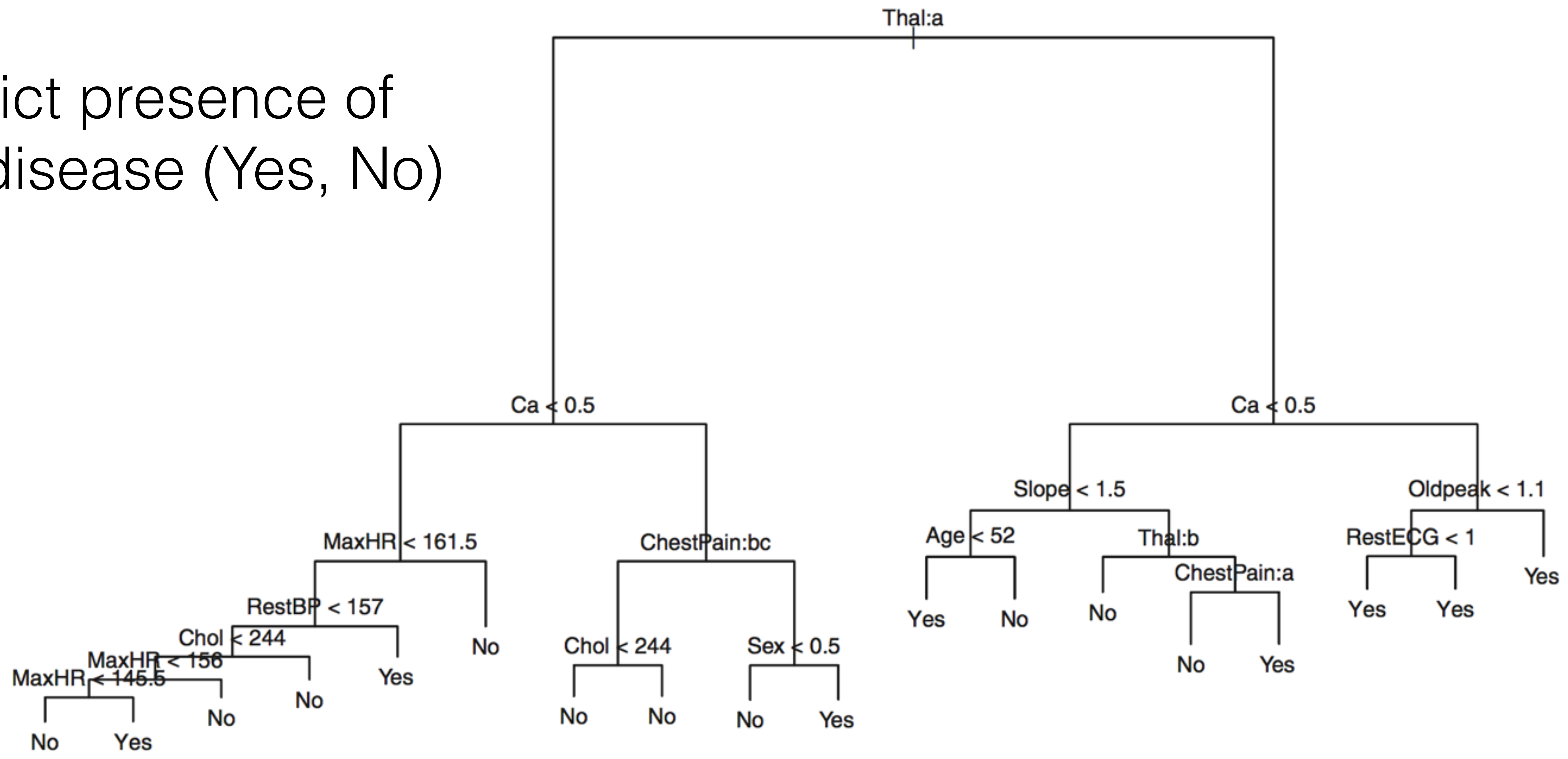


FIGURE 8.6, ISL (8th printing 2017)

Example Classification Tree

Predict presence of heart disease (Yes, No)

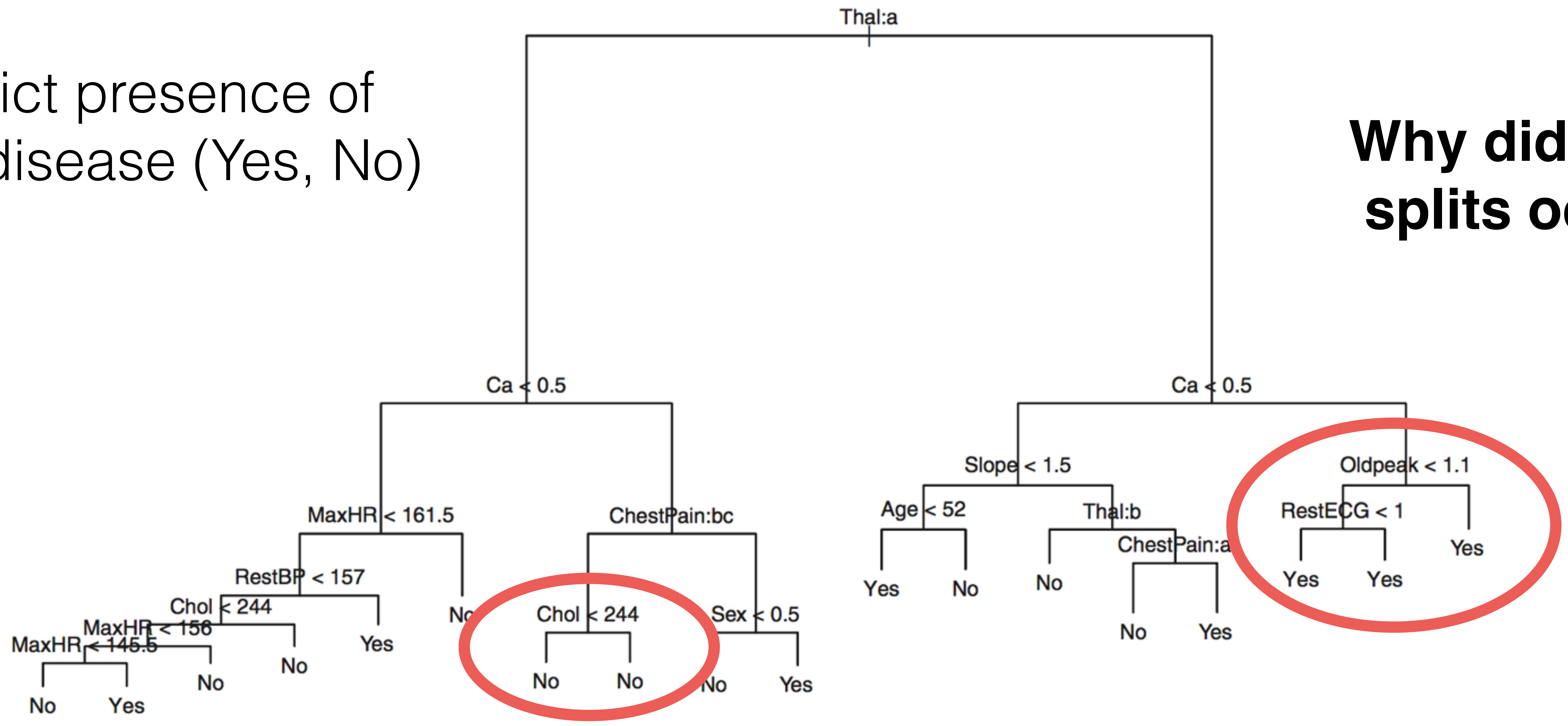


FIGURE 8.6, ISL (8th printing 2017)

Why did these splits occur?

Trees vs. Linear Models

Linear regression takes the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

Regression trees take the form

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

Trees vs. Linear Models

Which one will perform better? Depends on your problem.

Consider the classification problem at the right, and the boundaries discovered by logistic regression versus a classification tree.

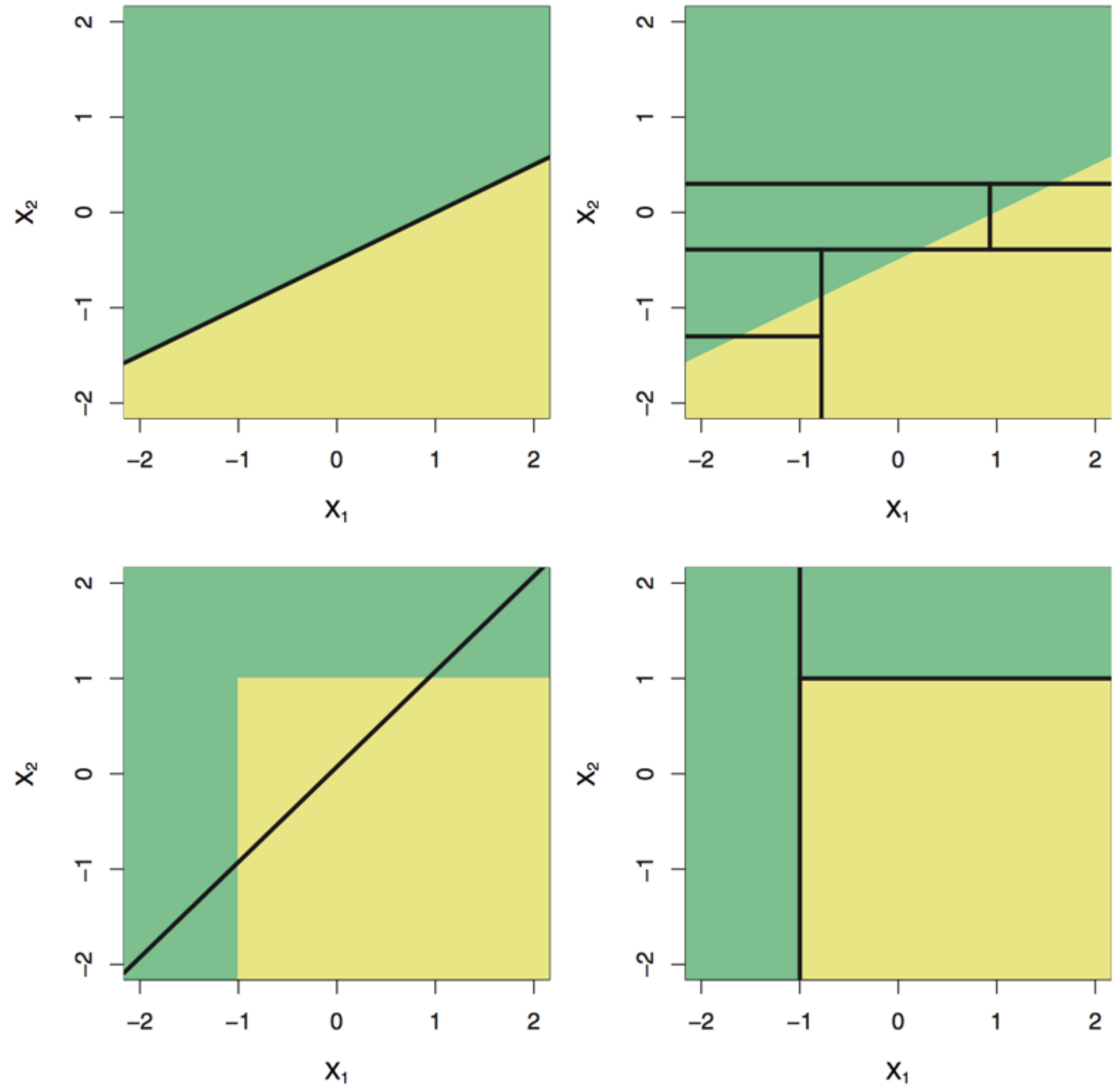


FIGURE 8.7, ISL (8th printing 2017)

Pros and Cons

Pros:

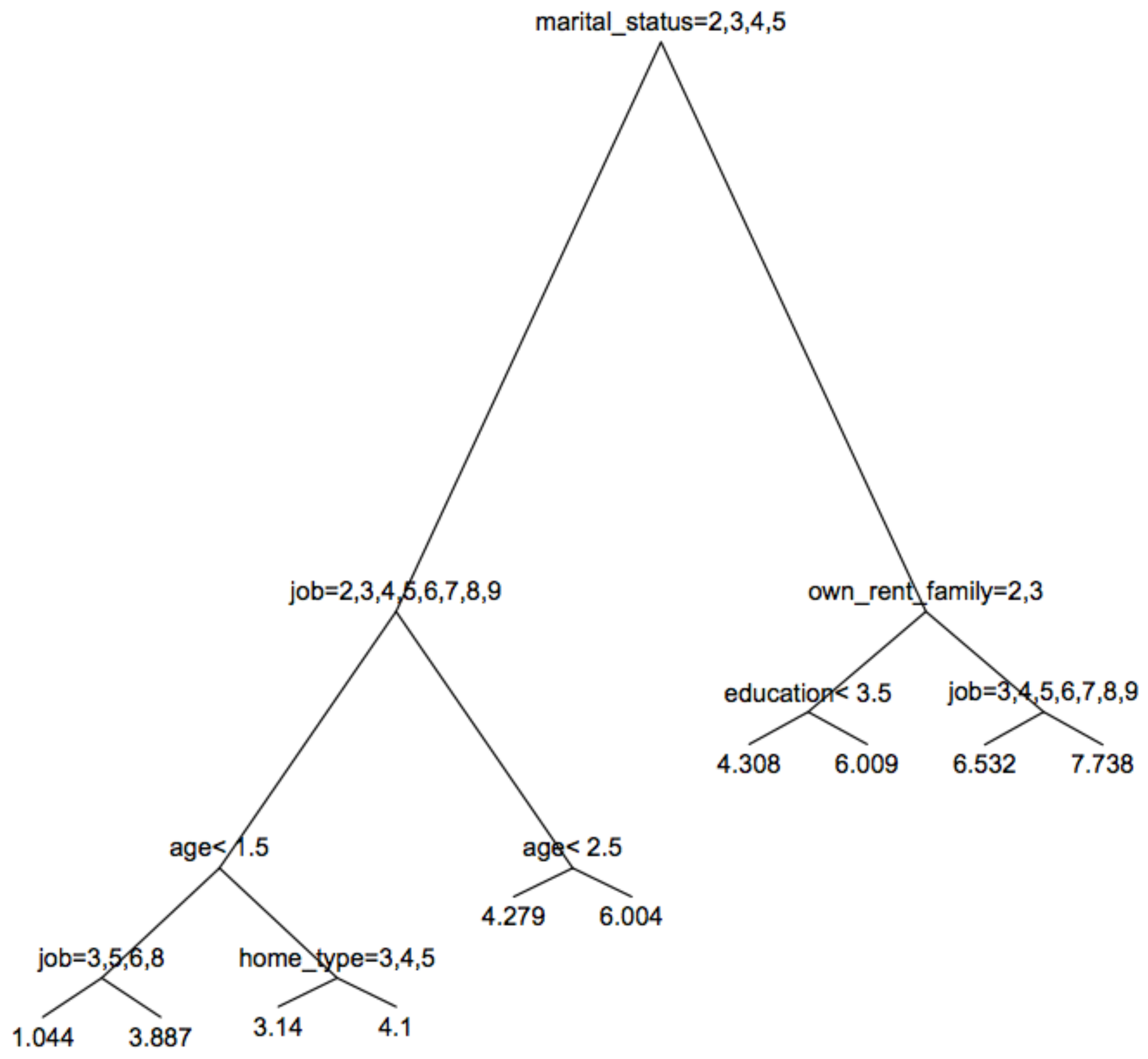
- Interpretable. Even more so than linear regression!
- Captures interactions between features
- Qualitative variables don't need to be transformed to quantitative ones

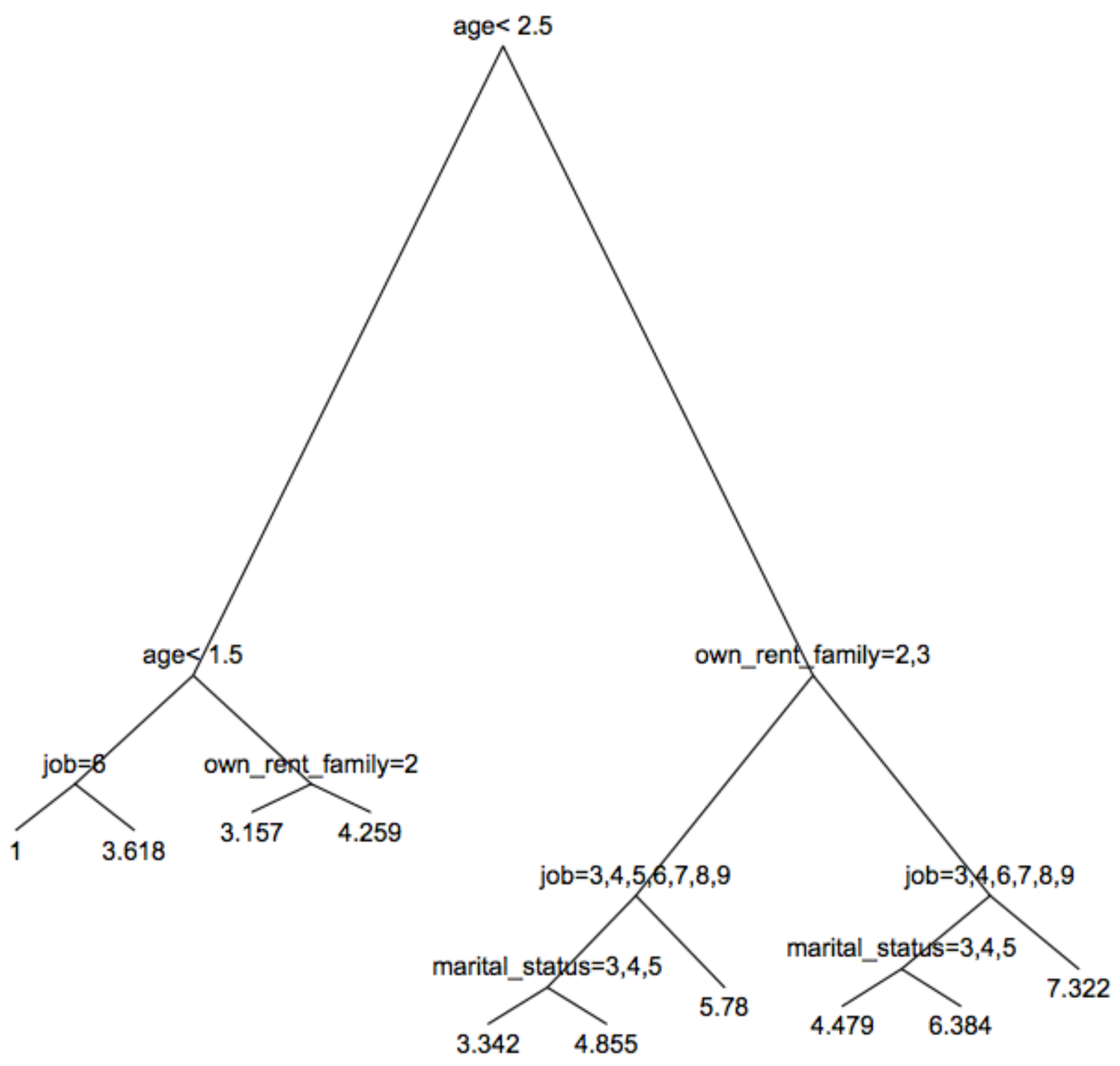
Cons:

- Trees can be unstable (high variance w.r.t. dataset)
- Lacks smoothness
- Hard to capture additivity

Tree Instability

Very different trees can result from a slight dataset perturbation.





Bagging

Bootstrap **Ag**gregation, “Bagging”

Bootstrapping is a technique that uses random sampling with replacement to perform tests or compute metrics (e.g. computing the standard deviation of a quantity).

Bootstrap aggregation, or **bagging**, is a general method used to improve machine learning methods by reducing variance.

Recall: Variance of Mean

Recall that the variance of the mean of n i.i.d. observations each with variance σ^2 is σ^2 / n.

In other words, *averaging a set of observations reduces variance*.

We can sample many training sets from the population, build a separate prediction model using each training set, and average the predictions to obtain a model with lower variance.

$$\hat{f}^1(x), \hat{f}^2(x), \dots, \hat{f}^B(x) \Longrightarrow \hat{f}_{\text{avg}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

Bootstrap Training Sets

Since we can't usually sample many training sets from the population, we take repeated samples from the one training dataset that we do have.

We then train models $\hat{f}^{*b}(x)$ on each of the bootstrapped datasets and average their predictions.

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

Bagging for Decision Trees

Due to the instability of individual decision trees, bagging is particularly useful for improving their performance.

Construct B regression or classification trees using B bootstrapped training sets and average or take the majority of the resulting predictions.

Can combine hundreds or even thousands of trees into a single *ensemble*.

Feature Importance

While individual decision trees are highly interpretable, B decision trees in aggregate become less interpretable, especially as B gets large.

However, we can still understand which features were most important for regression or classification by summing the total amount that the RSS, Gini index, or entropy decreased by splits on a given feature.

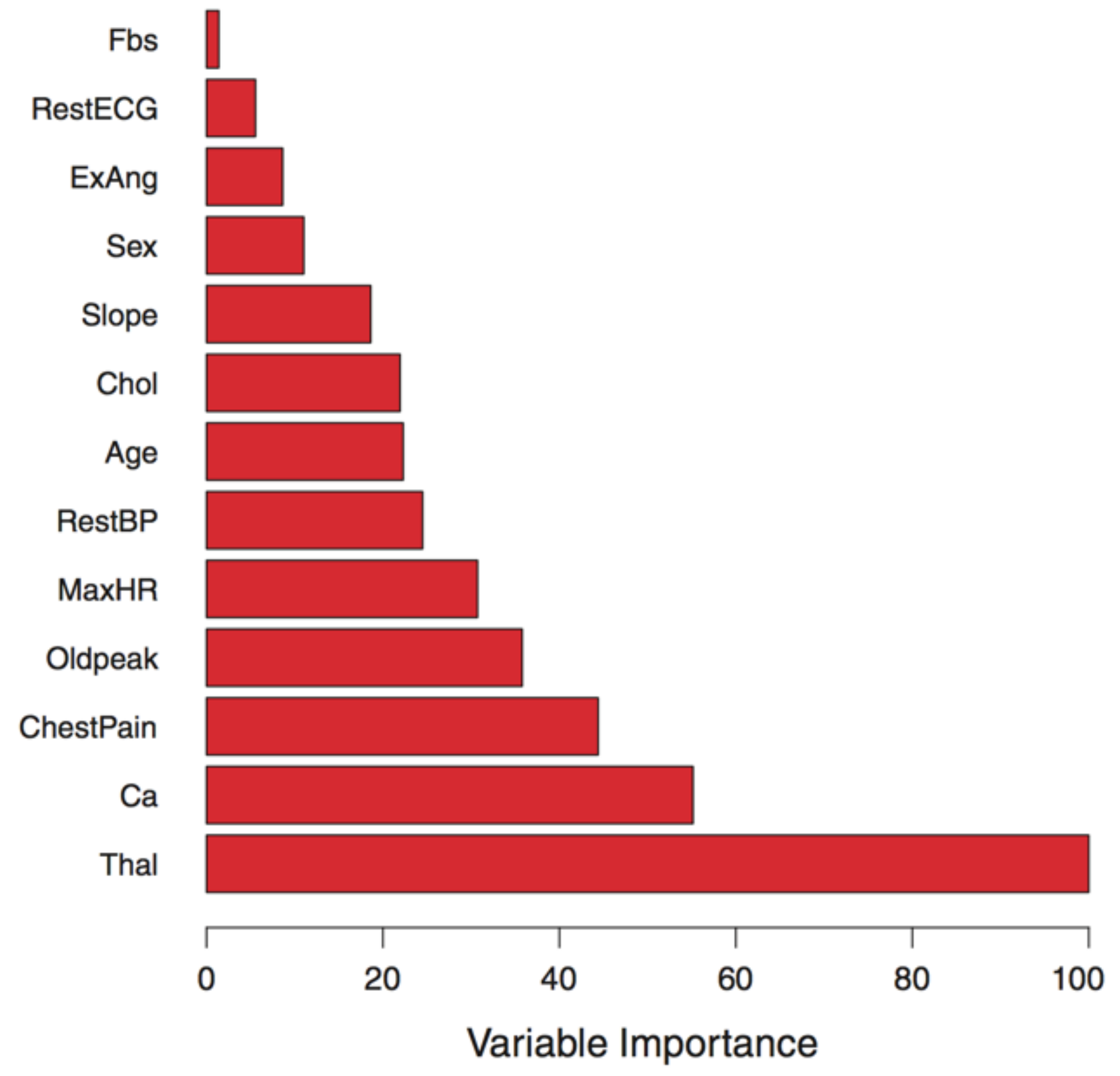


FIGURE 8.9, ISL (8th printing 2017)

Random Forests

Further Reducing Variance

Imagine that a dataset contains one feature that is very strongly correlated with the response variable.

Then in the bagged trees, most or all of the trees will use this feature in the top split, and the trees will look similar to each other.

Predictions will be highly correlated. And as we've discussed, the variance of the average of correlated quantities is not as small as the variance of the average of uncorrelated quantities.

$$\operatorname{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} \operatorname{Cov}(X_i, X_j) = \sum_{i=1}^{n} \operatorname{Var}(X_i) + 2 \sum_{1 \le i < j \le n} \operatorname{Cov}(X_i, X_j)$$

Random Forests

Random forests reduce tree similarity and prediction correlation by limiting each split to consider only a subset of predictors.

If we have p features, at each split we randomly remove p-m features from consideration, and split on only m features.

Note if p=m, this is simply bagging.

Random Forests

Random forests with 3 different numbers of features available shown at right. Task is a 15-class gene expression classification.

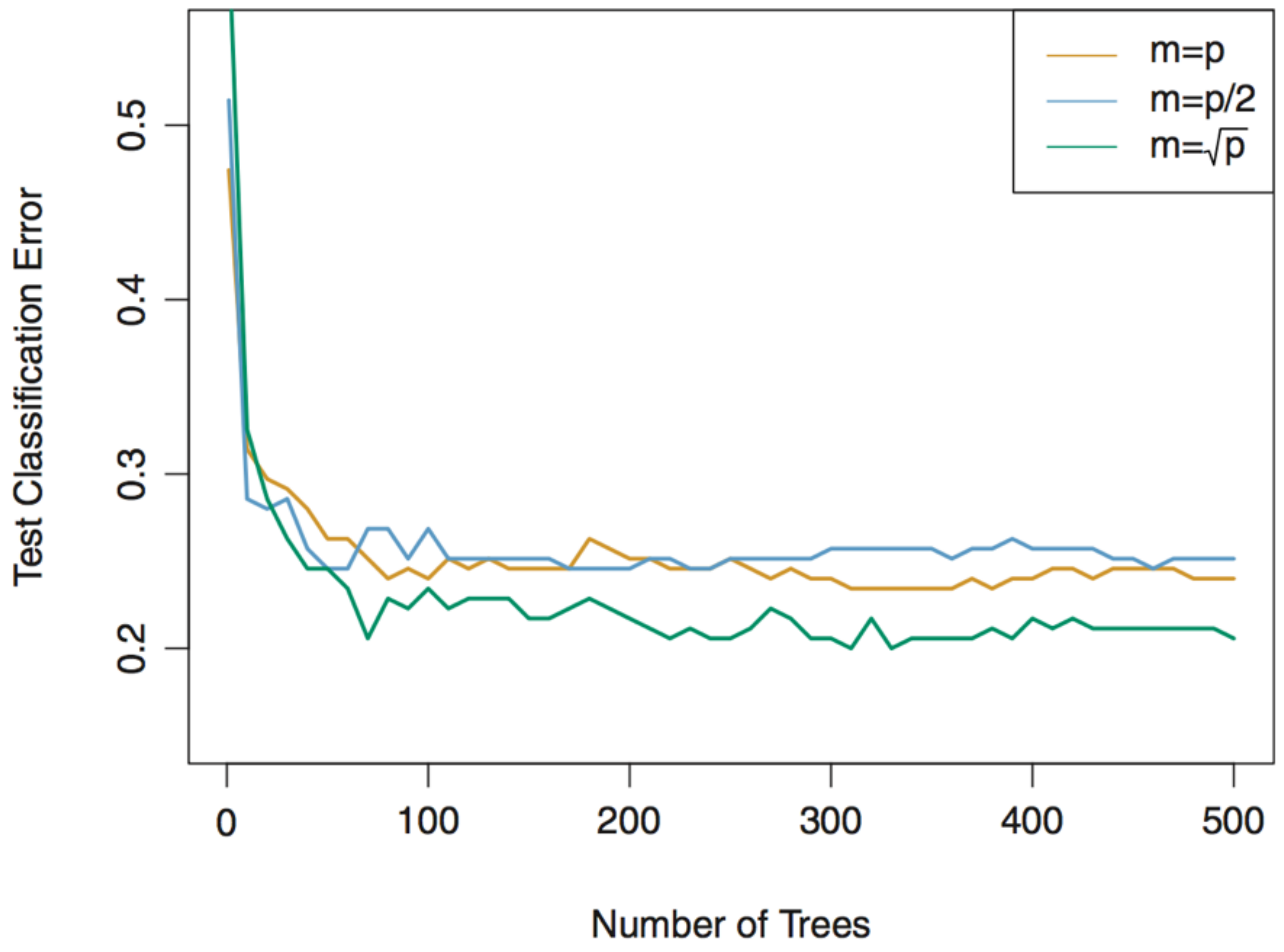
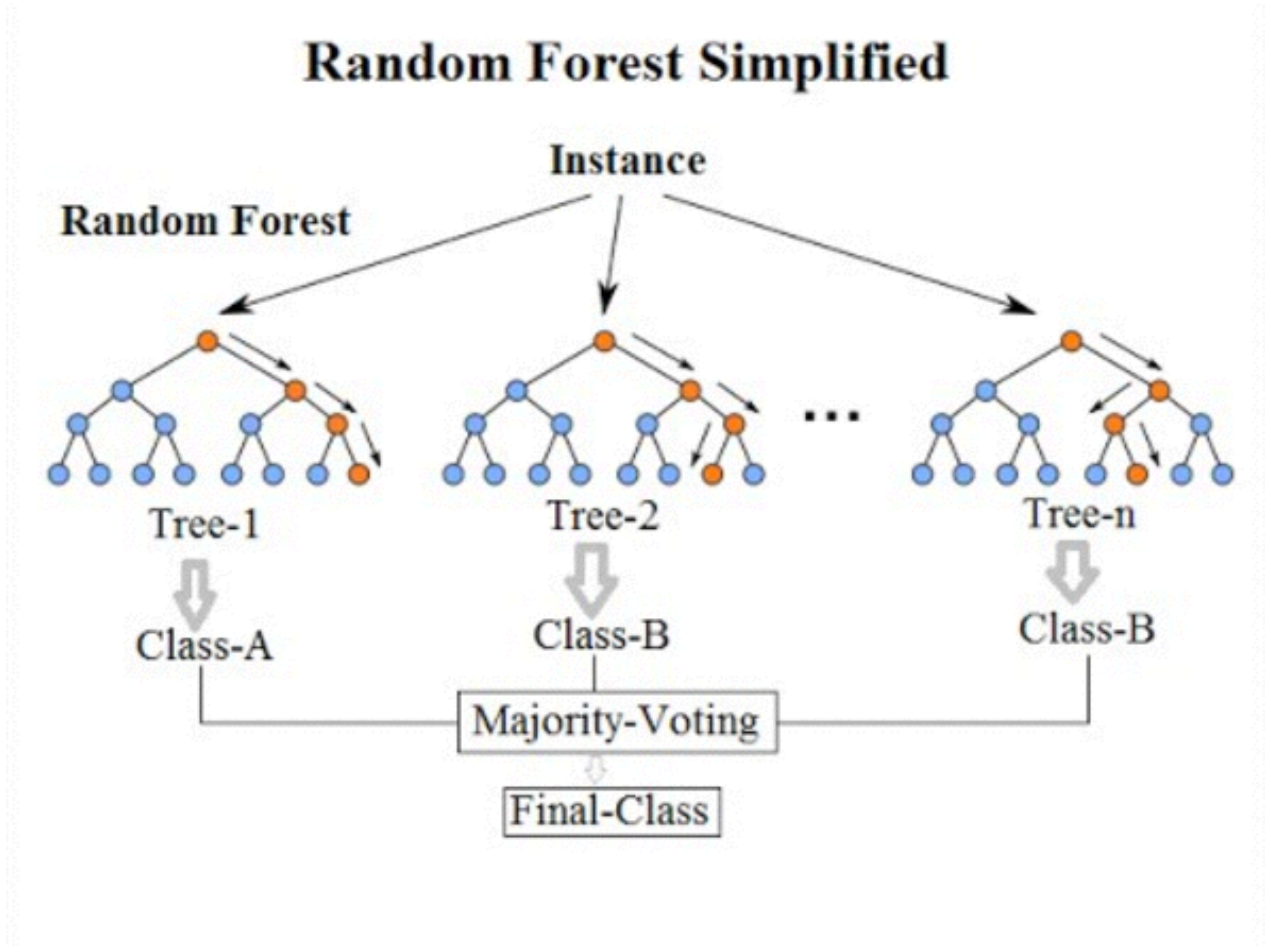


FIGURE 8.10, ISL (8th printing 2017)

Random Forests

One of the best out-of-the-box supervised learning algorithms out there.



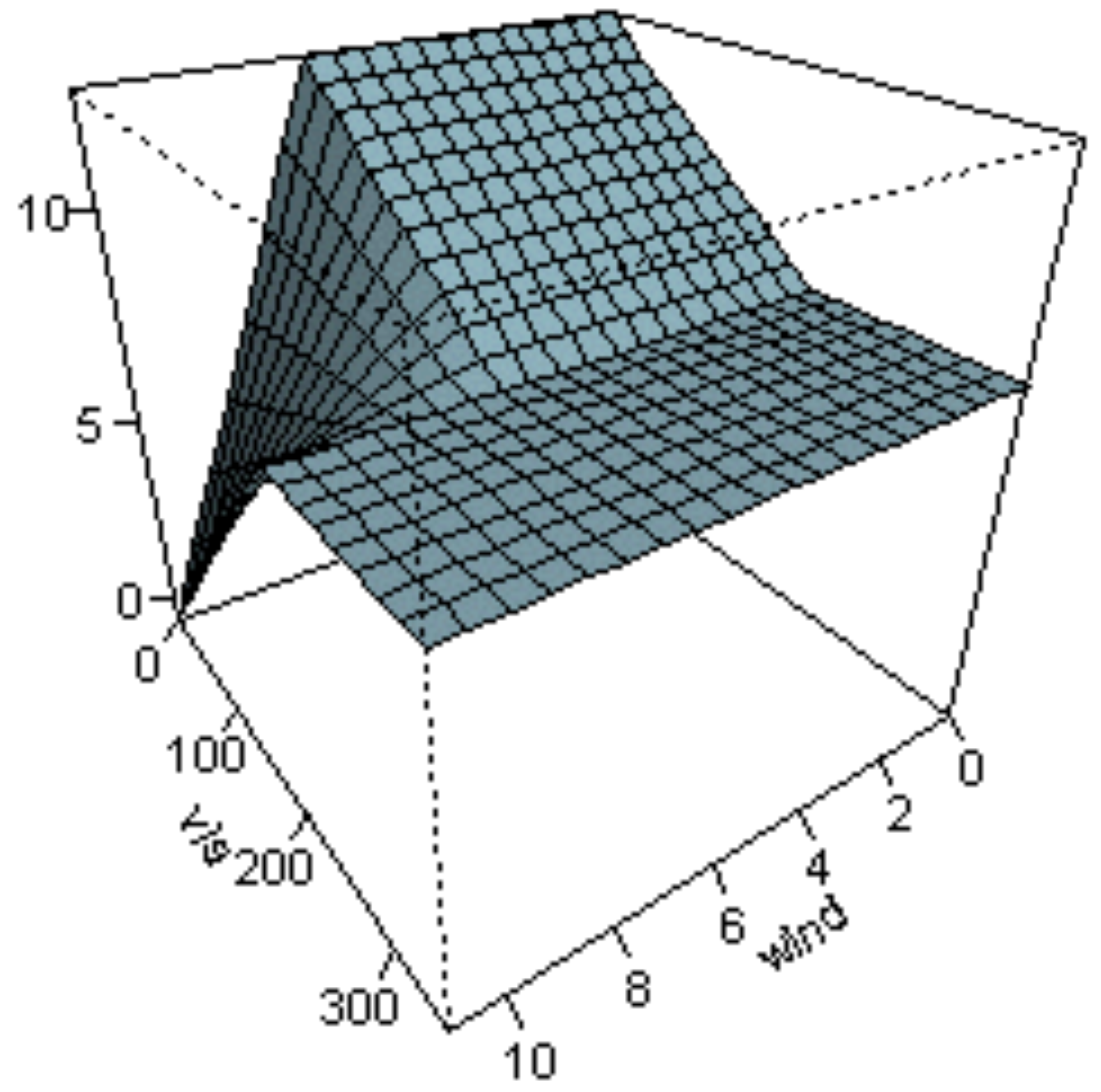
Random Forests in Python

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100,
criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1,
max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True)
```

As always, tune hyperparameters via CV.

Smoothness & Additivity: MARS

“Multivariate Adaptive Regression Splines” invented by Jerome Friedman in 1991.


vis
wind