Tree-Based Methods

Prof. Asim Tewari
IIT Bombay

What are Tree-Based Methods?

- These involve stratifying or segmenting the predictor space into a number of simple regions.
- These methods are used for both regression and classification.
- Typically use splitting rules (based on the mean or the mode of the training observations in the region to which it belongs) to segment the predictor space.

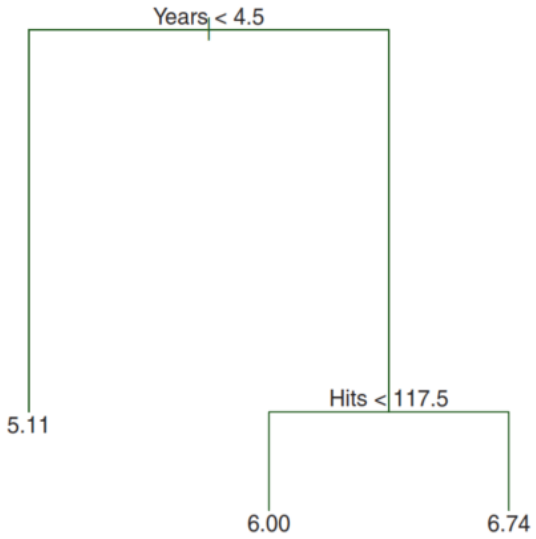
Variants of Tree based Methods

- Bagging
- Random forests
- Boosting
- Each of these approaches involves producing multiple trees which are then combined to yield a single consensus prediction.

Basics of Decision Trees

- Predict a baseball player's Salary based on Years (the number of years that he has played in the major leagues) and Hits (the number of hits that he made in the previous year).

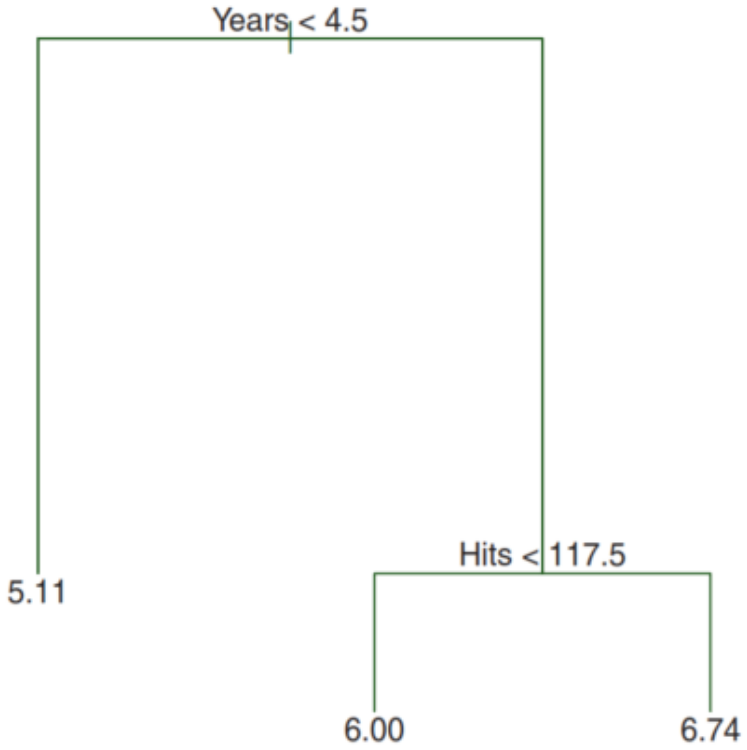
Basics of Decision Trees

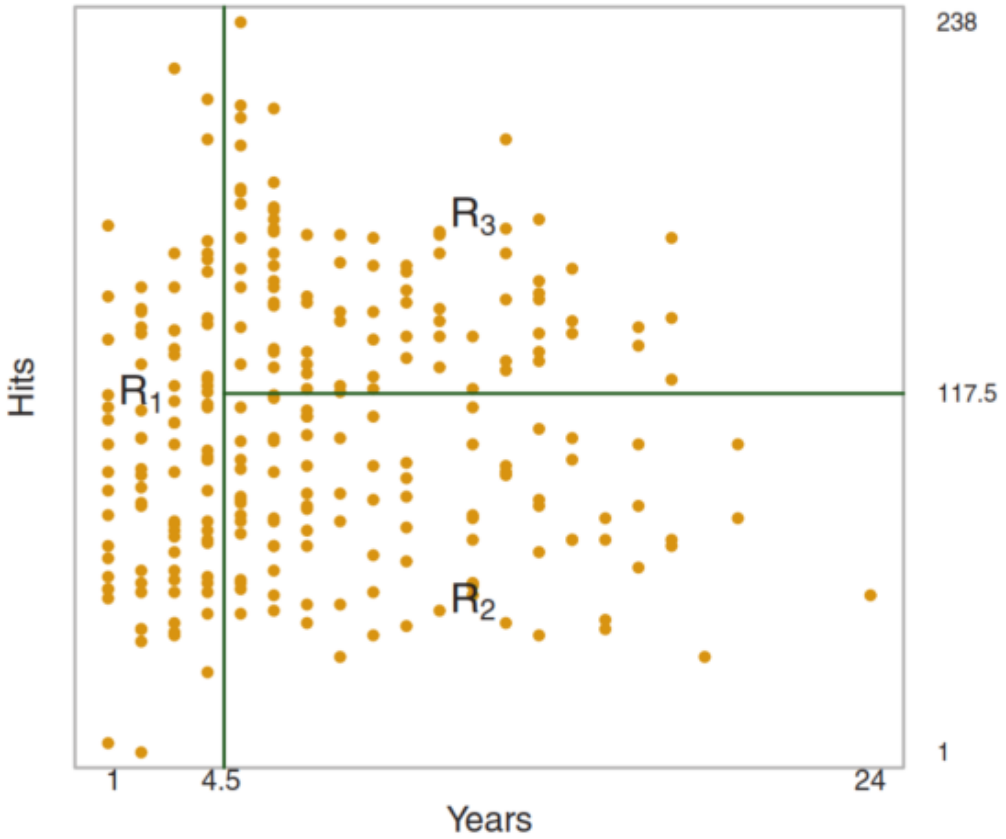


For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form Xj <tk) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to Xj ≥tk.

For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to Years<4.5, and the right-hand branch corresponds to Years ≥ 4.5. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there

Basics of Decision Trees





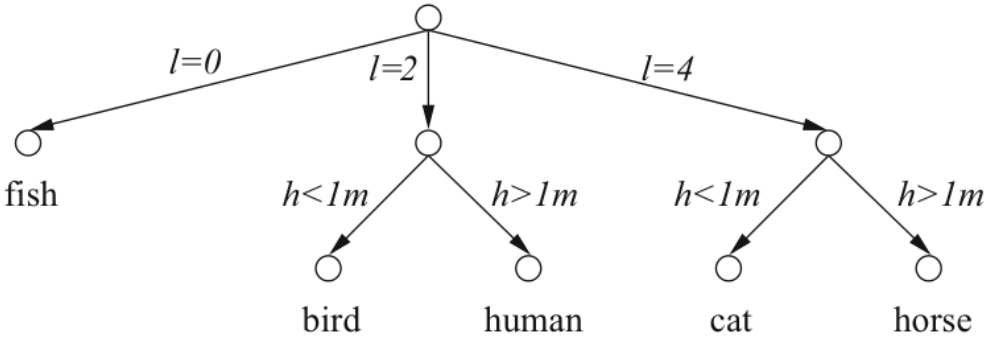
The three-region partition for the Hitters data set from the regression tree illustrated in the left Figure.

- R1, R2,and R3 are known as terminal nodes or leaves of the tree.

Basics of Decision Trees

index	1	2	3	4	5	6	7	8
height h [m]	0.1	0.2	1.8	0.2	2.1	1.7	0.1	1.6
legs l	0	2	2	4	4	2	4	2
class	fish	bird	human	cat	horse	human	cat	human

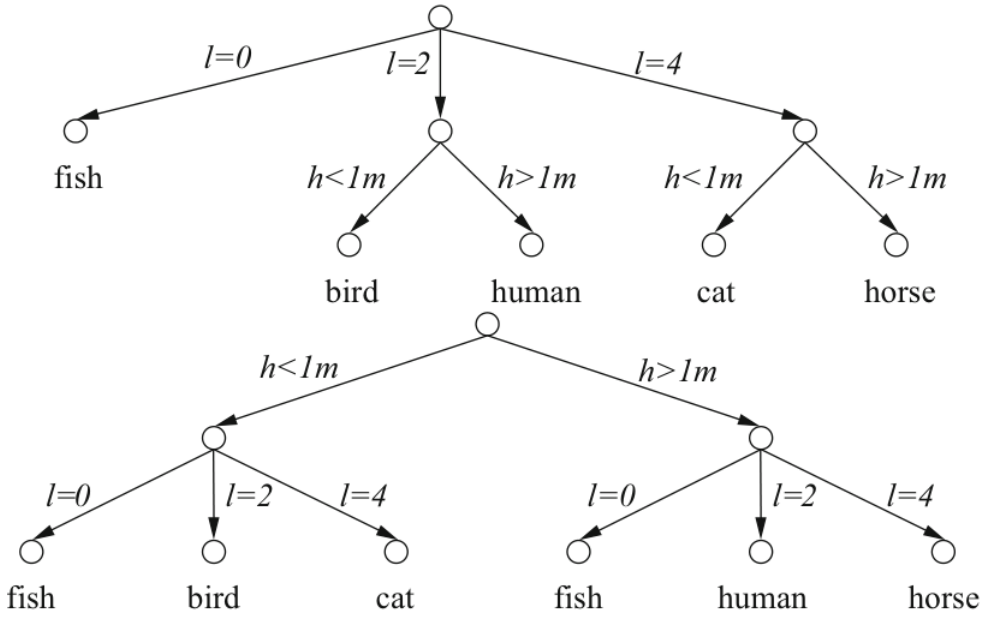
Decision trees



Basics of Decision Trees

Two functionally equivalent decision trees

index	1	2	3	4	5	6	7	8
height h [m]	0.1	0.2	1.8	0.2	2.1	1.7	0.1	1.6
legs l	0	2	2	4	4	2	4	2
class	fish	bird	human	cat	horse	human	cat	human



Basics of Decision Trees

Both decision trees realize the same classifier function but they yield different information gains in each step, depending on the order in which the features are considered. Given the training data Z =(X, Y), the probability for class k is

$$p(y = k) = \frac{|\{y \in Y \mid y = k\}|}{|Y|}$$

so the entropy at the root node is

$$H(Z) = -\sum_{k=1}^{c} p(y = k) \log_2 p(y = k)$$

$$= -\sum_{k=1}^{c} \frac{|\{y \in Y \mid y = k\}|}{|Y|} \log_2 \frac{|\{y \in Y \mid y = k\}|}{|Y|}$$

Basics of Decision Trees

At the root node we consider feature $x^{(j)}$, $j \in \{1, \ldots, p\}$, with the discrete range $x^{(j)} \in \{1, \ldots, v_j\}$, $v_j \in \{1, 2, \ldots\}$. The probability that $x^{(j)} = k$, $k = 1, \ldots, v_j$, is

$$p(x^{(j)} = k) = \frac{|\{x \in X \mid x^{(j)} = k\}|}{|X|}$$

and if $x^{(j)} = k$, then the new entropy is $H(Z \mid x^{(j)} = k)$. So considering feature $x^{(j)}$ yields an information gain of $H(Z)$ minus the expected value of $H(Z \mid x^{(j)} = k)$.

$$g_j = H(Z) - \sum_{k=1}^{v_j} p(x^{(j)} = k) \cdot H(Z \mid x^{(j)} = k)$$

$$= H(Z) - \sum_{k=1}^{v_j} \frac{|\{x \in X \mid x^{(j)} = k\}|}{|X|} H(Z \mid x^{(j)} = k)$$

Basics of Decision Trees

At each node, the decision tree that is optimal with respect to maximum entropy picks the feature that yields the highest information gain. For our example consider the data sets X and Y given in Table. The data set contains one fish, one bird, one horse, three humans, and two cats, so the entropy is

index	1	2	3	4	5	6	7	8
height h [m]	0.1	0.2	1.8	0.2	2.1	1.7	0.1	1.6
legs l	0	2	2	4	4	2	4	2
class	fish	bird	human	cat	horse	human	cat	human

Basics of Decision Trees

$$H(Z) = -\frac{1}{8}\log_2\frac{1}{8} - \frac{1}{8}\log_2\frac{1}{8} - \frac{3}{8}\log_2\frac{3}{8} - \frac{2}{8}\log_2\frac{2}{8} - \frac{1}{8}\log_2\frac{1}{8}$$
$$\approx 2.1556 \text{ bit}$$

We first consider the decision tree in the top view of Fig. , where the feature $l \in \{0, 2, 4\}$ is considered first. The corresponding conditional entropies are

$$H(Z \mid l = 0) = -1\log_2 1 = 0$$
$$H(Z \mid l = 2) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} \approx 0.8113 \text{ bit}$$
$$H(Z \mid l = 4) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} \approx 0.9183 \text{ bit}$$

Basics of Decision Trees

So at the root node the information gain for feature l is

$$g_l = H(Z) - \frac{1}{8}H(Z \mid l = 0) - \frac{4}{8}H(Z \mid l = 2) - \frac{3}{8}H(Z \mid l = 4)$$
$$\approx 1.4056 \text{ bit}$$

Next we consider the decision tree in the bottom view of Fig. , where the feature $h \in \{< 1m, > 1m\}$ is considered first. The corresponding conditional entropies are

$$H(Z \mid h < 1m) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{2}{4}\log_2\frac{2}{4} \approx 1.5 \text{ bit}$$
$$H(Z \mid h > 1m) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} \approx 0.8113 \text{ bit}$$

So at the root node the information gain for feature h is

$$g_h = H(Z) - \frac{4}{8}H(Z \mid h < 1m) - \frac{4}{8}H(Z \mid h > 1m)$$
$$\approx 1 \text{ bit}$$

At the root node, the information gain of feature l is higher than the information gain of feature h, $g_l > g_h$, so in our example the optimal decision tree is the one shown in the *top* view of Fig.

Basics of Decision Trees

$$H(Z) = -\frac{1}{8}\log_2\frac{1}{8} - \frac{1}{8}\log_2\frac{1}{8} - \frac{3}{8}\log_2\frac{3}{8} - \frac{2}{8}\log_2\frac{2}{8} - \frac{1}{8}\log_2\frac{1}{8}$$
$$\approx 2.1556 \text{ bit}$$

index	1	2	3	4	5	6	7	8
height h [m]	0.1	0.2	1.8	0.2	2.1	1.7	0.1	1.6
legs l	0	2	2	4	4	2	4	2
class	fish	bird	human	cat	horse	human	cat	human

Basics of Decision Trees

We first consider the decision tree in the top view of Fig. , where the feature $l \in \{0, 2, 4\}$ is considered first. The corresponding conditional entropies are

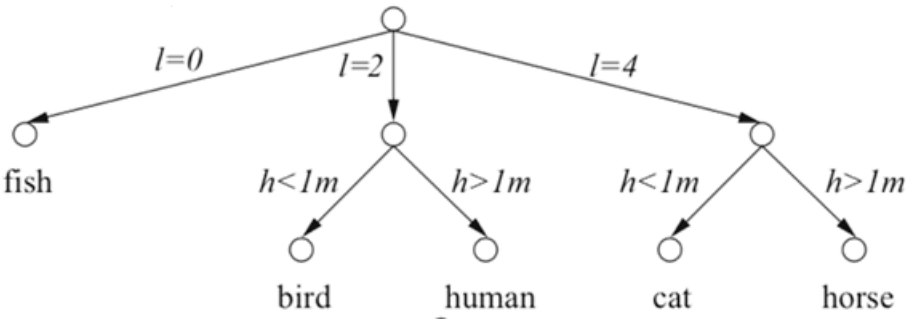
$$H(Z \mid l = 0) = -1 \log_2 1 = 0$$

$$H(Z \mid l = 2) = -\frac{1}{4} \log_2 \frac{1}{4} - \frac{3}{4} \log_2 \frac{3}{4} \approx 0.8113 \text{ bit}$$

$$H(Z \mid l = 4) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} \approx 0.9183 \text{ bit}$$

So at the root node the information gain for feature l is

$$g_l = H(Z) - \frac{1}{8} H(Z \mid l = 0) - \frac{4}{8} H(Z \mid l = 2) - \frac{3}{8} H(Z \mid l = 4)$$
$$\approx 1.4056 \text{ bit}$$



index	1	2	3	4	5	6	7	8
height h [m]	0.1	0.2	1.8	0.2	2.1	1.7	0.1	1.6
legs l	0	2	2	4	4	2	4	2
class	fish	bird	human	cat	horse	human	cat	human

Basics of Decision Trees

So at the root node the information gain for feature l is

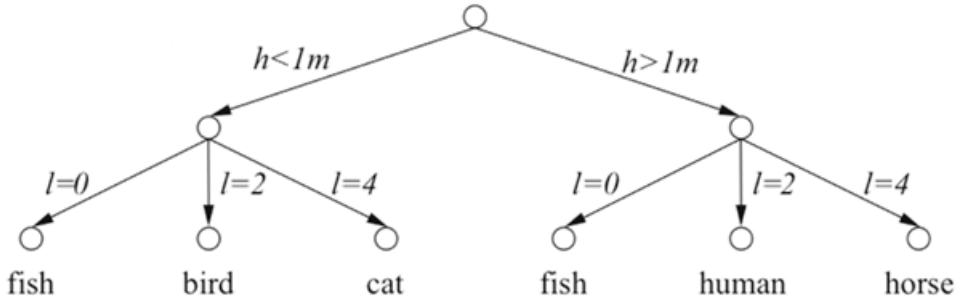
$$g_l = H(Z) - \frac{1}{8}H(Z \mid l = 0) - \frac{4}{8}H(Z \mid l = 2) - \frac{3}{8}H(Z \mid l = 4)$$
$$\approx 1.4056 \text{ bit}$$

Next we consider the decision tree in the bottom view of Fig. , where the feature $h \in \{< 1m, > 1m\}$ is considered first. The corresponding conditional entropies are

$$H(Z \mid h < 1m) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{2}{4}\log_2\frac{2}{4} \approx 1.5 \text{ bit}$$
$$H(Z \mid h > 1m) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} \approx 0.8113 \text{ bit}$$

So at the root node the information gain for feature h is

$$g_h = H(Z) - \frac{4}{8}H(Z \mid h < 1m) - \frac{4}{8}H(Z \mid h > 1m)$$
$$\approx 1 \text{ bit}$$



At the root node, the information gain of feature l is higher than the information gain of feature h, $g_l > g_h$, so in our example the optimal decision tree is the one shown in the *top* view of Fig.

index	1	2	3	4	5	6	7	8
height h [m]	0.1	0.2	1.8	0.2	2.1	1.7	0.1	1.6
legs l	0	2	2	4	4	2	4	2
class	fish	bird	human	cat	horse	human	cat	human

Basics of Decision Trees

$$H(Z) = -\frac{1}{8}\log_2\frac{1}{8} - \frac{1}{8}\log_2\frac{1}{8} - \frac{3}{8}\log_2\frac{3}{8} - \frac{2}{8}\log_2\frac{2}{8} - \frac{1}{8}\log_2\frac{1}{8}$$
$$\approx 2.1556 \text{ bit}$$

We first consider the decision tree in the top view of Fig. , where the feature $l \in \{0, 2, 4\}$ is considered first. The corresponding conditional entropies are

$$H(Z \mid l = 0) = -1\log_2 1 = 0$$
$$H(Z \mid l = 2) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} \approx 0.8113 \text{ bit}$$
$$H(Z \mid l = 4) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} \approx 0.9183 \text{ bit}$$

So at the root node the information gain for feature l is

$$g_l = H(Z) - \frac{1}{8}H(Z \mid l = 0) - \frac{4}{8}H(Z \mid l = 2) - \frac{3}{8}H(Z \mid l = 4)$$
$$\approx 1.4056 \text{ bit}$$

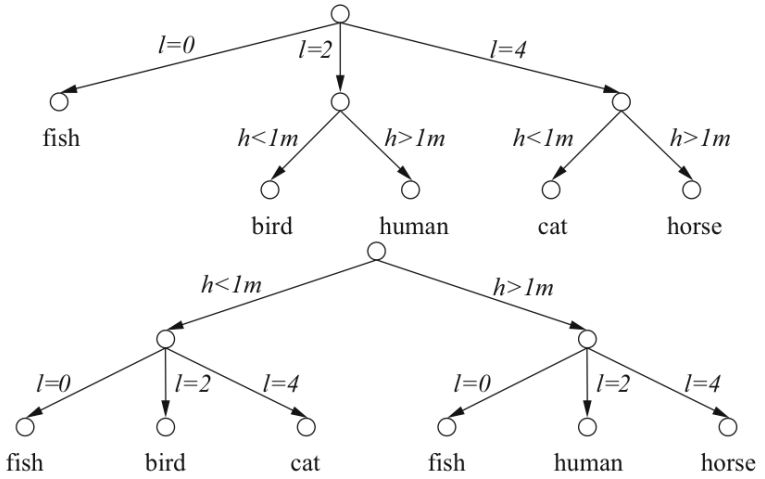
Next we consider the decision tree in the bottom view of Fig. , where the feature $h \in \{< 1m, > 1m\}$ is considered first. The corresponding conditional entropies are

$$H(Z \mid h < 1m) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{2}{4}\log_2\frac{2}{4} \approx 1.5 \text{ bit}$$
$$H(Z \mid h > 1m) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} \approx 0.8113 \text{ bit}$$

So at the root node the information gain for feature h is

$$g_h = H(Z) - \frac{4}{8}H(Z \mid h < 1m) - \frac{4}{8}H(Z \mid h > 1m)$$
$$\approx 1 \text{ bit}$$

At the root node, the information gain of feature l is higher than the information gain of feature h, $g_l > g_h$, so in our example the optimal decision tree is the one shown in the *top* view of Fig.



index	1	2	3	4	5	6	7	8
height h [m]	0.1	0.2	1.8	0.2	2.1	1.7	0.1	1.6
legs l	0	2	2	4	4	2	4	2
class	fish	bird	human	cat	horse	human	cat	human

Prediction via Stratification of the Feature Space

The process of building a regression tree, roughly speaking, has the following two steps.

1. We divide the predictor space—that is, 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, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Division of predictor space

- 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 R1,...,Rj that minimize the RSS, given by

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

- Where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Division of predictor space

- Unfortunately, it is 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.

- The approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region) 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, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Division of predictor space

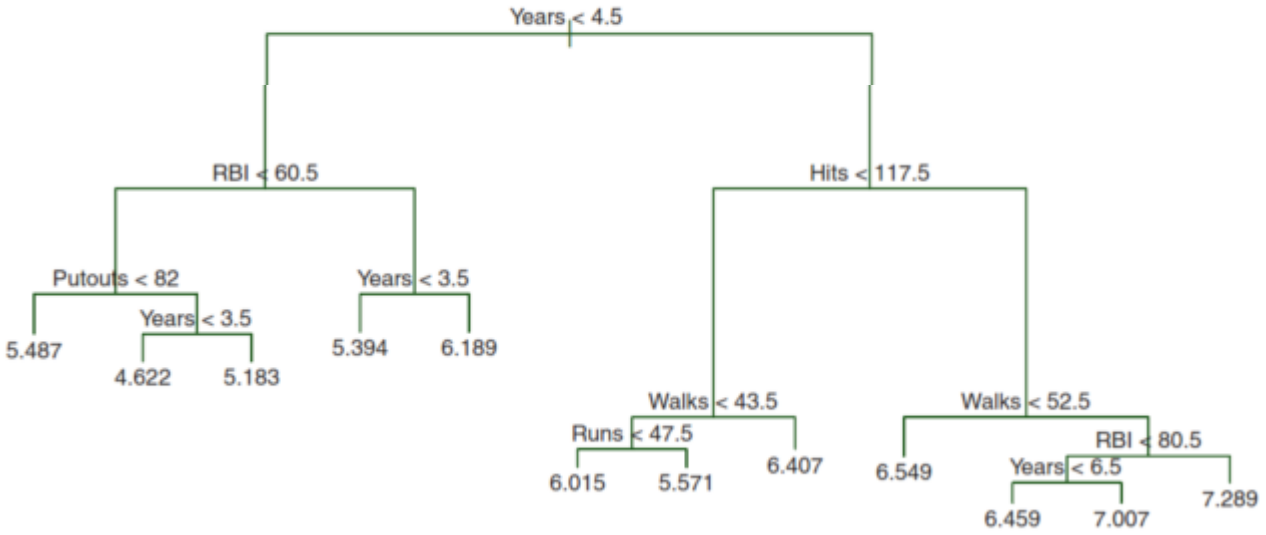
- We first select the predictor Xj and the cutpoint s such that splitting the predictor space into the regions {X|Xj<s} and {X|X>= s} leads to the greatest possible reduction in RSS.
- We seek the value of j and s that minimize the equation

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

- Next, we repeat the process within each of the resulting regions

Regression Tree

- For regression tree, the predicted response for an observation is given by the mean response of the training observations that belong to the same terminal node.



Over fitting of data

- The previously described process is likely to overfit the data, leading to poor test set performance.
- This is because the resulting tree might be too complex.
- A smaller tree with fewer splits (that is, fewer regions R1,...,Rj) might lead to lower variance and better interpretation at the cost of a little bias.

Over fitting of data

- The over fitting can be addressed by many strategies:

1. Build 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 could be too short-sighted.
2. a better strategy is to grow a very large tree T0, and then prune it back in order to obtain a subtree.

Tree Pruning

- Our goal is to select a subtree that leads to the lowest test error rate.
- However, estimating the cross-validation error for every possible subtree would be too cumbersome.

Cost complexity pruning

- This is weakest link pruning, which rather than every possible subtree, only considers a sequence of trees indexed by a nonnegative tuning parameter α.
- For each value of α there corresponds a subtree which minimizes

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

Here |T | indicates the number of terminal nodes of the tree

Cost complexity pruning

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

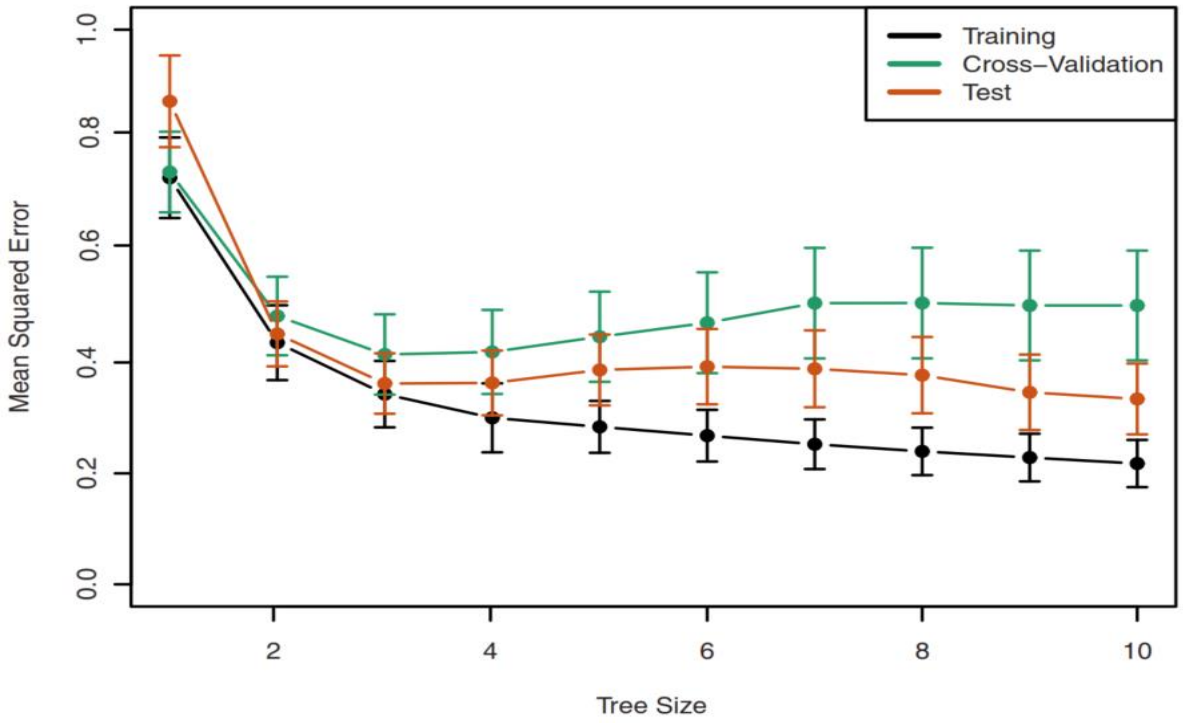
(a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

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

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

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

Cost complexity pruning



Regression tree analysis for the Hitters data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.

Classification Trees

- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs (mode class for the region).

Classification Trees

- In the classification setting, RSS cannot be used as a criterion for making the binary splits. A natural alternative to RSS is the classification error rate.
- The classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max(P_{mk})$$

- Here, Pmk represents the proportion of training observations in the mth region that are from the kth class.

Gini index

- It turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.
- The **Gini index**

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

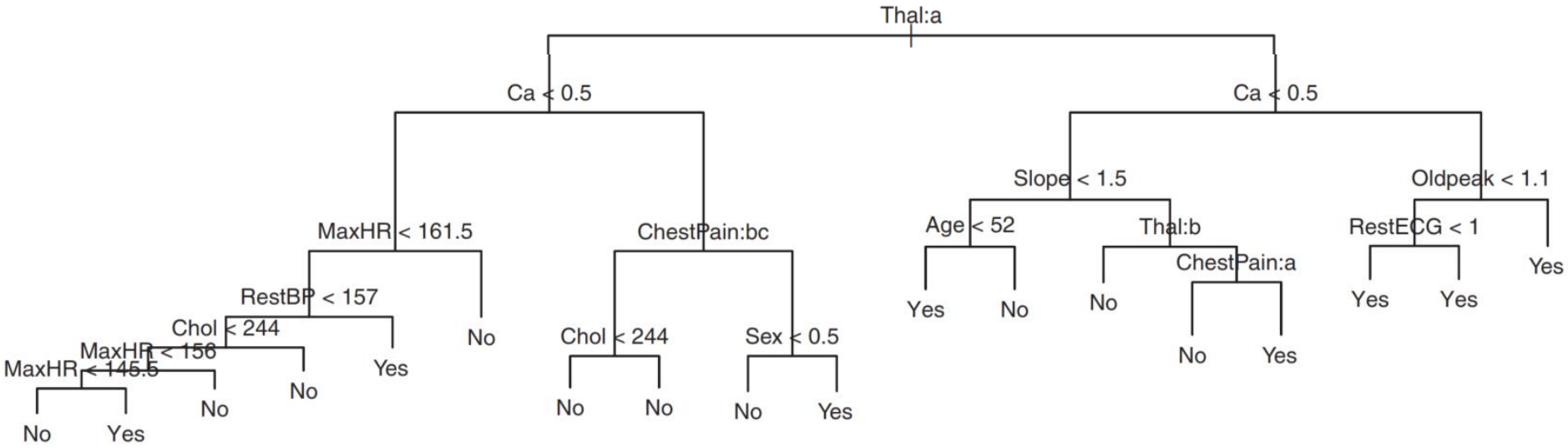
- This is a measure of total variance across the K classes. It provides a measure of node purity—a small value indicates that a node contains predominantly observations from a single class.

Cross-Entropy

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

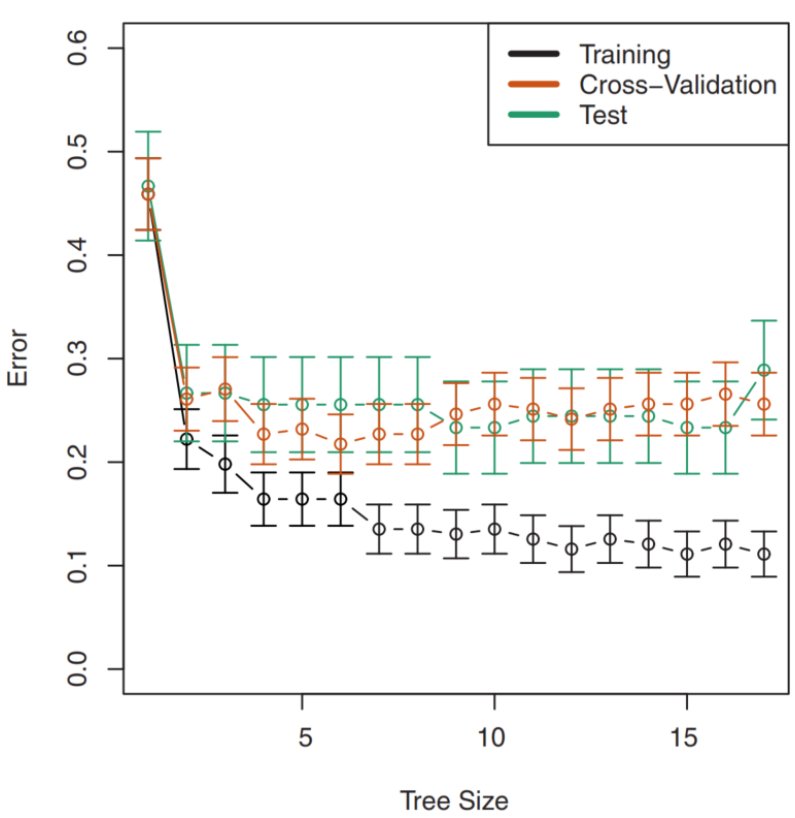
- One can show that the cross-entropy will take on a value near zero if the ˆp mk's are all near zero or near one. Therefore, like the Gini index, the cross-entropy will take on a small value if the mth node is pure.

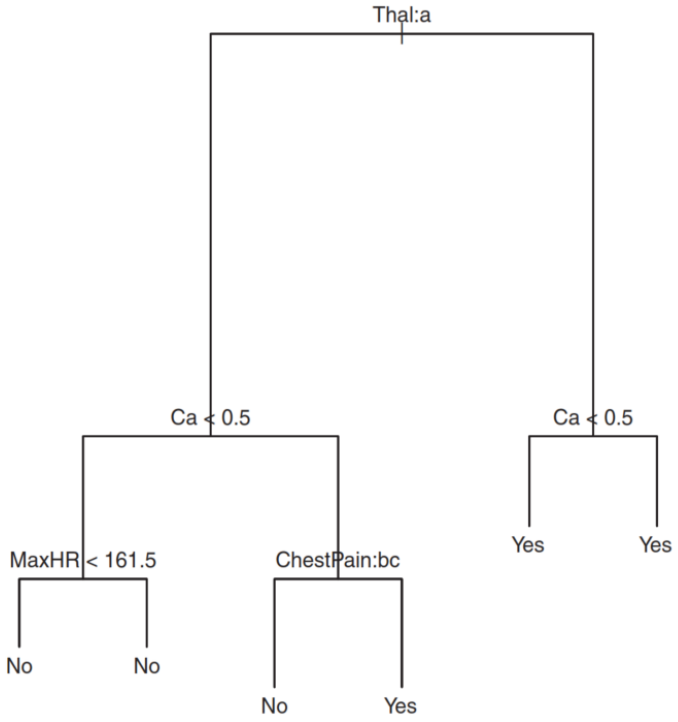
Classification Tree



An example on the Heart data set. These data contain a binary outcome HD for 303 patients who presented with chest pain. An outcome value of Yes indicates the presence of heart disease based on an angiographic test, while No means no heart disease. There are 13 predictors including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements.

Classification Tree





- Left: Cross-validation error, training, and test error, for different sizes of the pruned tree.
- Right: The pruned tree corresponding to the minimal cross-validation error.

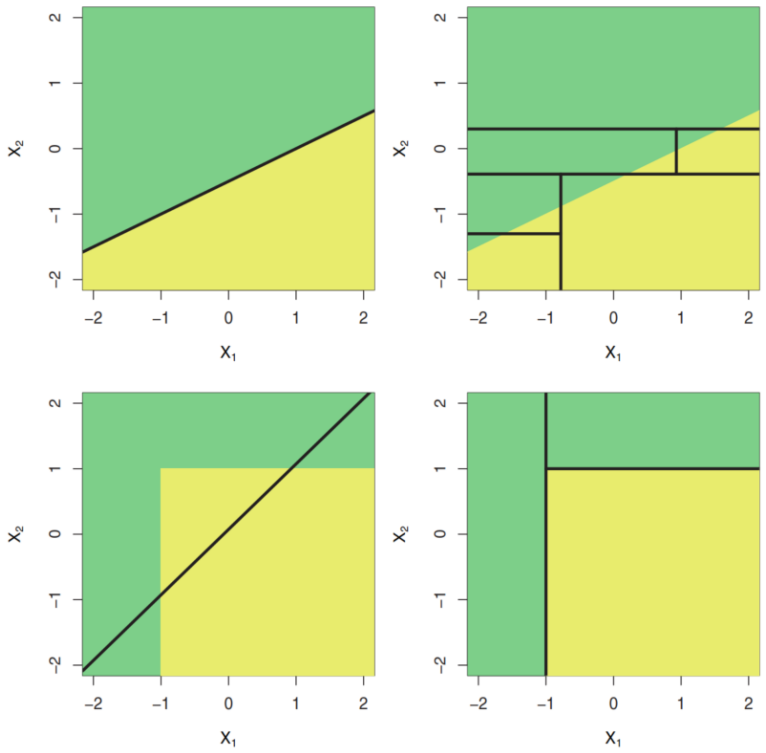
Trees Versus Linear Models

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

Linear Regression Model

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

Regression tree Model



Advantage of trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.

Disadvantages of Trees

- Do not have the same level of predictive accuracy as some of the other regression and classification approaches.

- Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

Bagging, random forests, boosting

- Bagging, random forests, and boosting use trees as building blocks to construct more powerful prediction models.

Bagging regression trees

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

- In bagging regression trees, we simply construct B regression trees using B bootstrapped training sets, and average the resulting predictions. These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias. Averaging these B trees reduces the variance.
- Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.
- The number of trees B is not a critical parameter with bagging; using a very large value of B will not lead to overfitting. In practice we use a value of B sufficiently large that the error has settled down. Using B = 100 is sufficient to achieve good performance in this example.

Bagging Classification trees

- For a given test observation, we can record the class predicted by each of the B trees, and take a **majority vote**: the overall prediction is the most commonly occurring class among the B predictions.

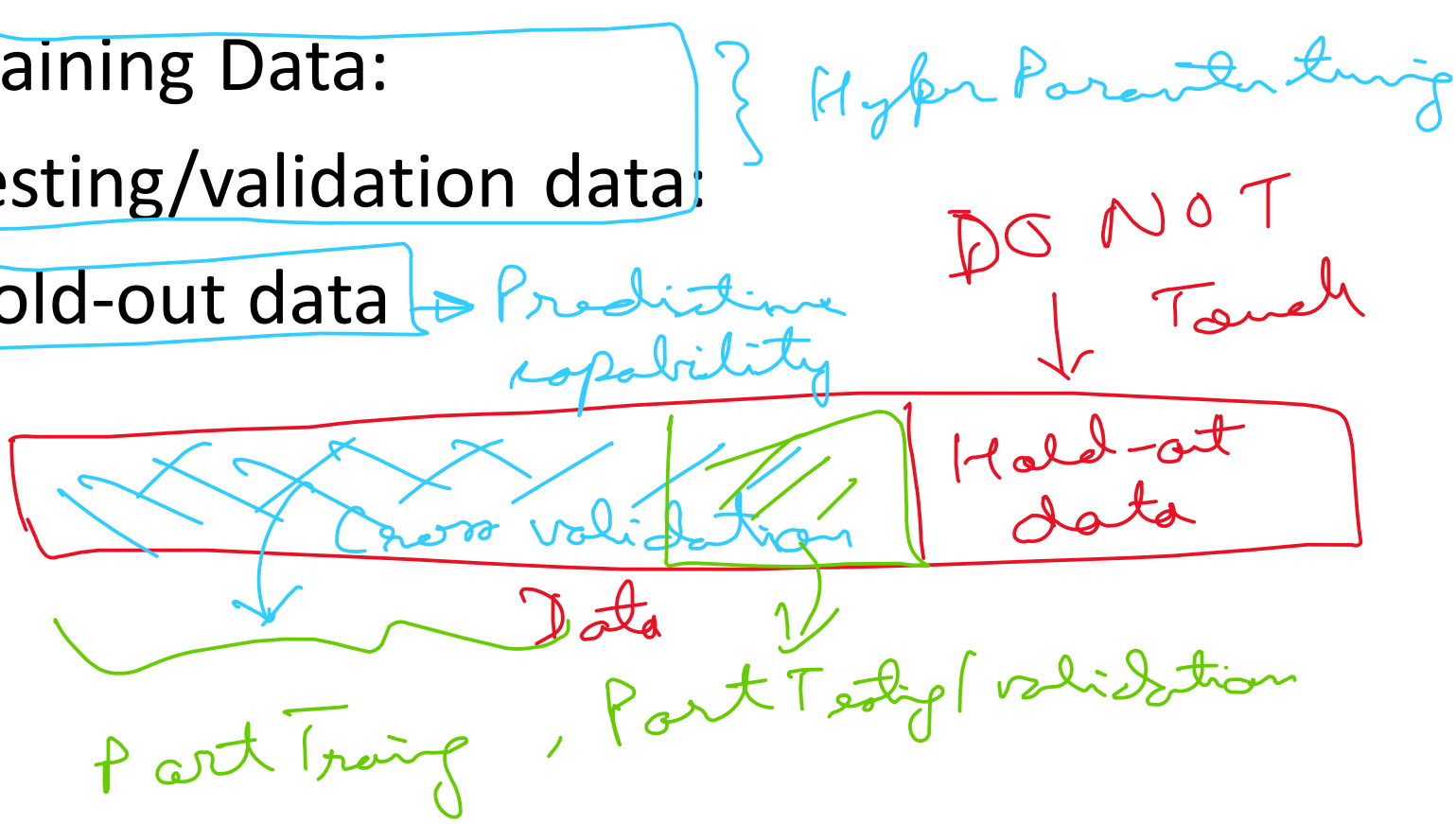
Out-of-Bag Error Estimation

- In bagging trees are repeatedly fit to bootstrapped subsets (around 2/3 of the observations) of the observations.

- The remaining 1/3 of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.

Training, Testing/validation and Hold-out data

- Training Data:
- Testing/validation data:
- Hold-out data



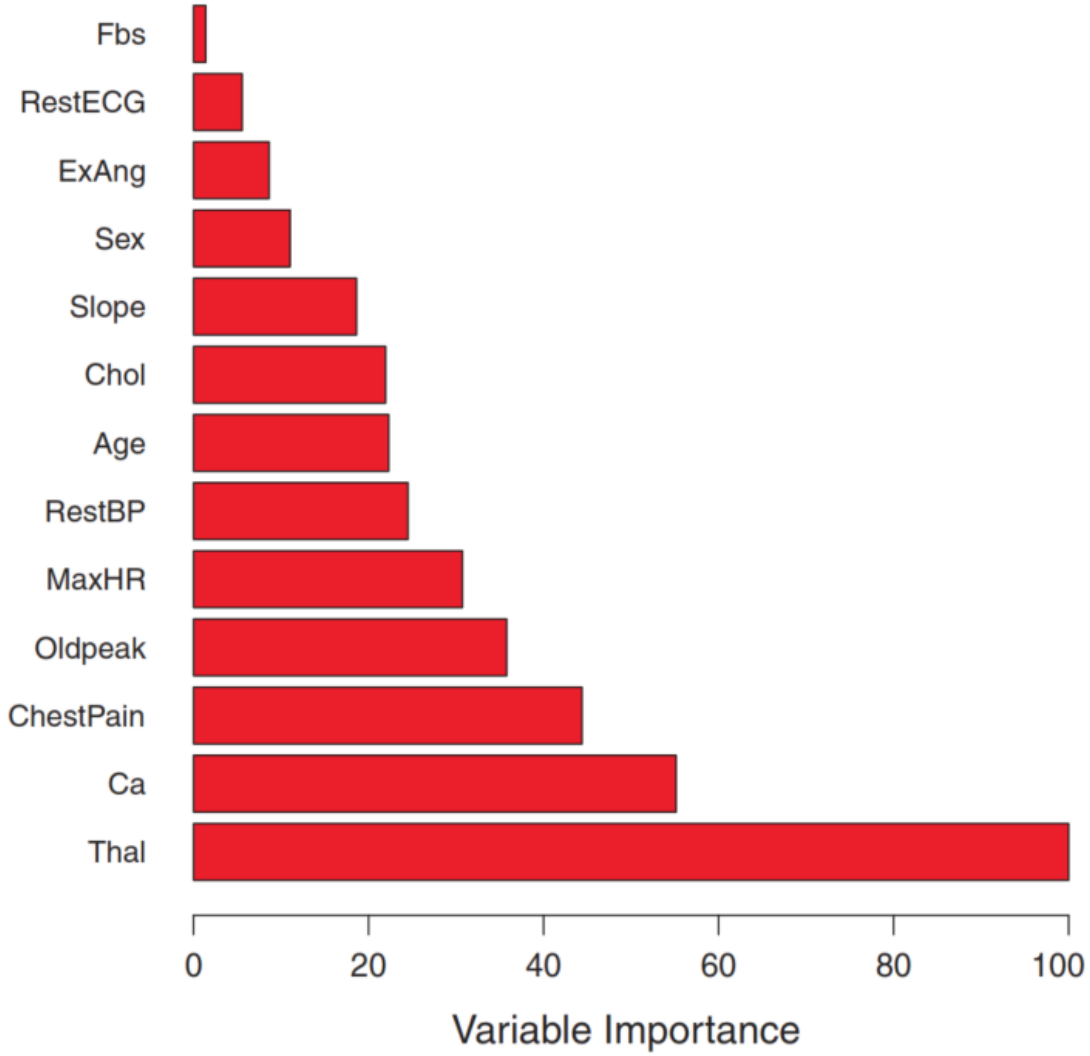
Random Forests

- Random forests a small tweak in bagged trees decorrelates the trees
- each time a split in a tree is considered, a random sample of *m* (~sqrt(p)) predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.

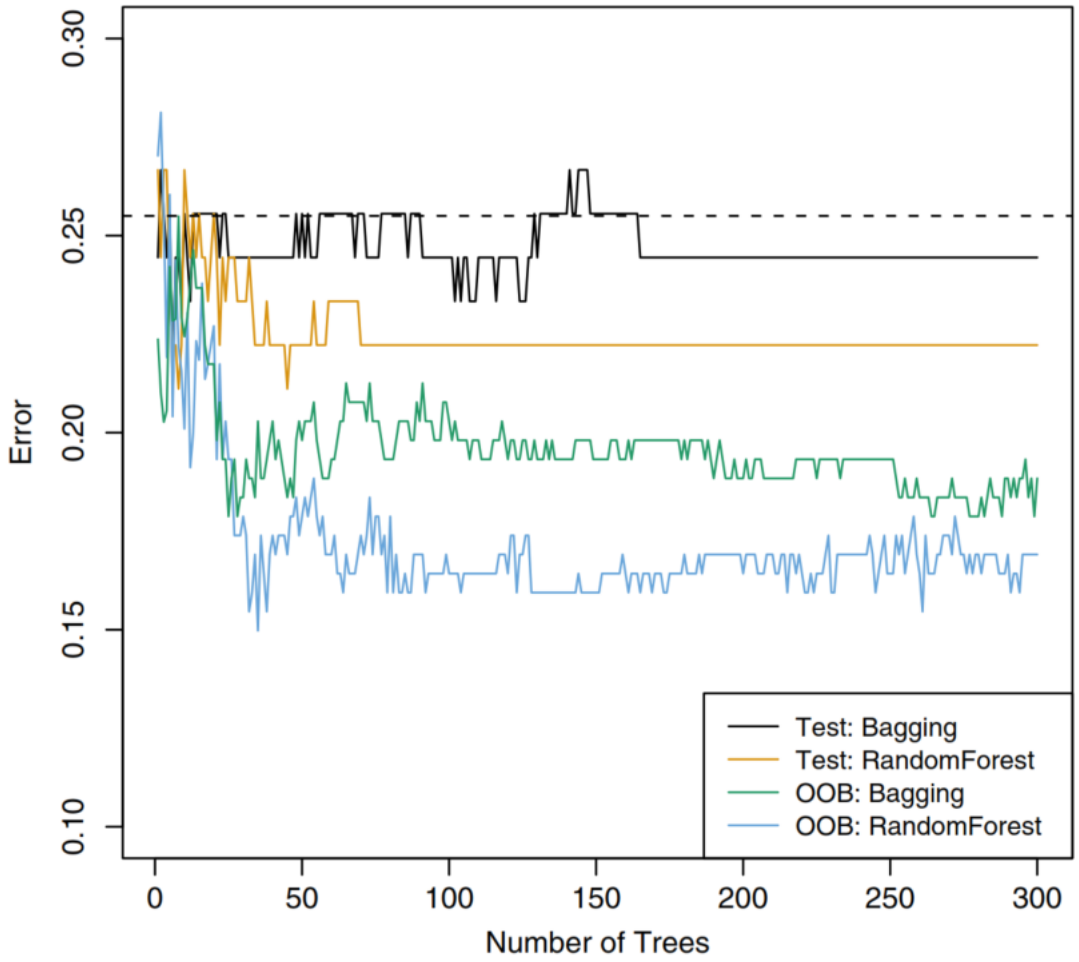
Random Forests

- Hence, in Random Forest, the algorithm is not even allowed to consider a majority of the available predictors
- This may sound crazy, but it has a clever rationale.
- Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split.
- Consequently, all of the bagged trees will be highly correlated, and averaging many highly correlated quantities does not lead to a large reduction in variance.
- Random forests overcome this problem by forcing each split to consider only a subset of the predictors

Random Forest



Random Forest vs Bagging



Boosting

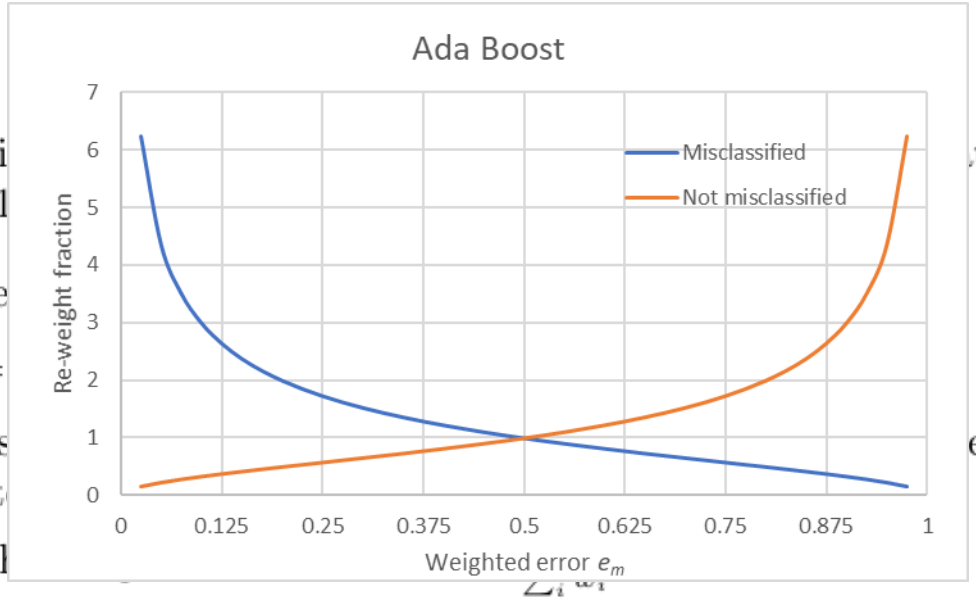
The idea of boosting is to combine the models (typically called weak learners in this context) in a more principled manner.

AdaBoost

There are many flavors of boosting. We will discuss one of the most popular versions, known as AdaBoost (short for adaptive boosting), which is a method for binary classification. Its developers won the prestigious Godel Prize for this work.

AdaBoost

Algorithm



We present the algori taset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1$

1. Initialize the we
2. Repeat for $m =$
   (a) Build a clas ess the data are weighted according t
   (b) Compute tl
   (c) Re-weight the training points as

$$w_i \leftarrow w_i \cdot \begin{cases} \sqrt{\frac{1-e_m}{e_m}} & \text{if misclassified by } G_m \\ \sqrt{\frac{e_m}{1-e_m}} & \text{otherwise} \end{cases}$$

   (d) Optional: normalize the weights w_i to sum to 1.

Nearest Neighbor Classifier

A much simpler classifier method is the nearest neighbor classifier which assigns an object with a given feature vector to the class of the training object with the most similar feature vector. More formally, for a given feature vector x the nearest neighbor classifier yields class y_k if

$$\|x - x_k\| = \min_{j=1,\ldots,n} \|x - x_j\|$$

where k:k is a suitable dissimilarity measure, for example the Euclidean or the Mahalanobis distance. In case of multiple minima one of the minima may be chosen randomly.

The resulting class boundary is piecewise linear along edges of a so–called Voronoi diagram. For noisy data or overlapping classes the nearest neighbor classifier sometimes yields bad results near the noise data or the class borders. This can be avoided by the k-nearest neighbor classifier which not only considers the nearest neighbor but the k ϵ (2,.. n} nearest neighbors and yields the most frequent class of these neighbors. For two classes, ties can be avoided by selecting odd values for k. Don't confuse the number k of nearest neighbors with the object index k