Section 11Tree methods

- **Tree-based method for regression**
- **Tree-based method for classification**
- **Bagging**
- **Random Forest**
- **Boosting**

➢ Overview:

- **Decisions trees:** splitting each variable sequentially, creating rectangular regions.
  - **CART** refers to **C**lassification **A**nd **R**egression **T**rees.
  - Making fitting/prediction locally at each region.
  - It is intuitive and easy to implement, may have good interpretation.
  - Lower prediction accuracy in general (weak learners).
- The **Boosting problem** (Kearns & Valiant 1988): Can a set of weak learners create a single strong learner.

Bagging, random forests and boosting ... make fitting/prediction based on a number of trees.

Bagging and **Boosting** are general methodologies, not just limited to trees.

➢ Regression trees (A case study):

We first consider regression trees through an example of predicting Baseball players' salaries.

This dataset was originally taken from the StatLib library maintained at Carnegie Mellon University.

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	PutOuts	Assists	Errors	Salary
1	315	81	7	24	38	39	14	3449	835	69	321	414	375	632	43	10	475.0
2	479	130	18	66	72	76	3	1624	457	63	224	266	263	880	82	14	480.0
3	496	141	20	65	78	37	11	5628	1575	225	828	838	354	200	11	3	500.0
4	321	87	10	39	42	30	2	396	101	12	48	46	33	805	40	4	91.5
5	594	169	4	74	51	35	11	4408	1133	19	501	336	194	282	421	25	750.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

AtBat Number of times at bat in 1986
Hits Number of hits in 1986
HmRun Number of home runs in 1986
Runs Number of runs in 1986
RBI Number of runs batted in in 1986
Walks Number of walks in 1986
Years Number of years in the major leagues
CAtBat Number of times at bat during his career
CHits Number of hits during his career

CHmRun Number of home runs during his career
CRuns Number of runs during his career
CRBI Number of runs batted in during his career
CWalks Number of walks during his career
PutOuts Number of put outs in 1986
Assists Number of assists in 1986
Errors Number of errors in 1986
Salary 1987 annual salary on opening day in thousands of dollars

- **Predictors/Inputs/Covariates:**

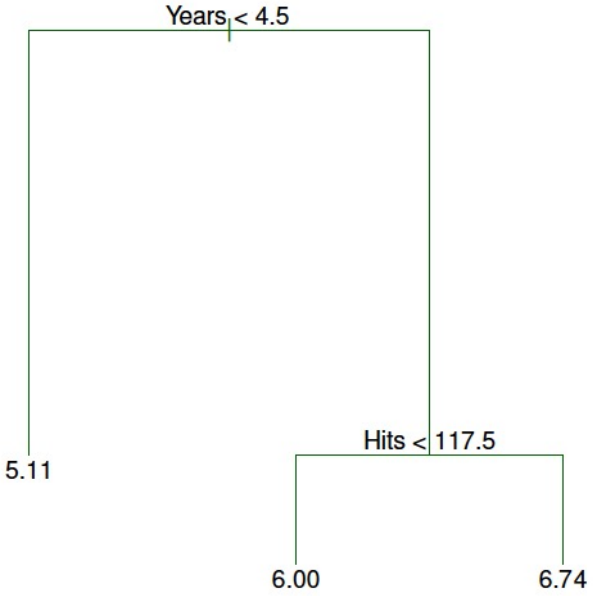
1. Years (the number of years that he has played in the major leagues)

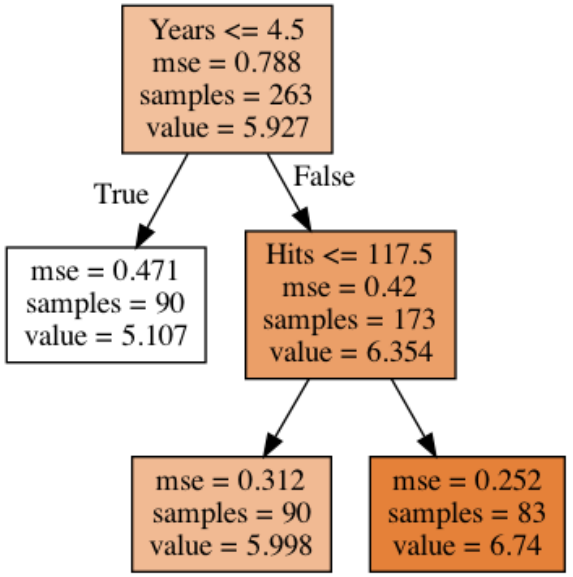
2. Hits (the number of hits that he made in the previous year).

- **Response/Outputs: Salary**.

Preparing data: remove the observations with missing data and log-transformed the Salary (preventing heavy right-skewness)

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.





- The number in each **terminal node (leaf)** is the **mean** of the response for the observations that fall there.

  In the example, the tree has two internal nodes and three terminal nodes, or leaves.

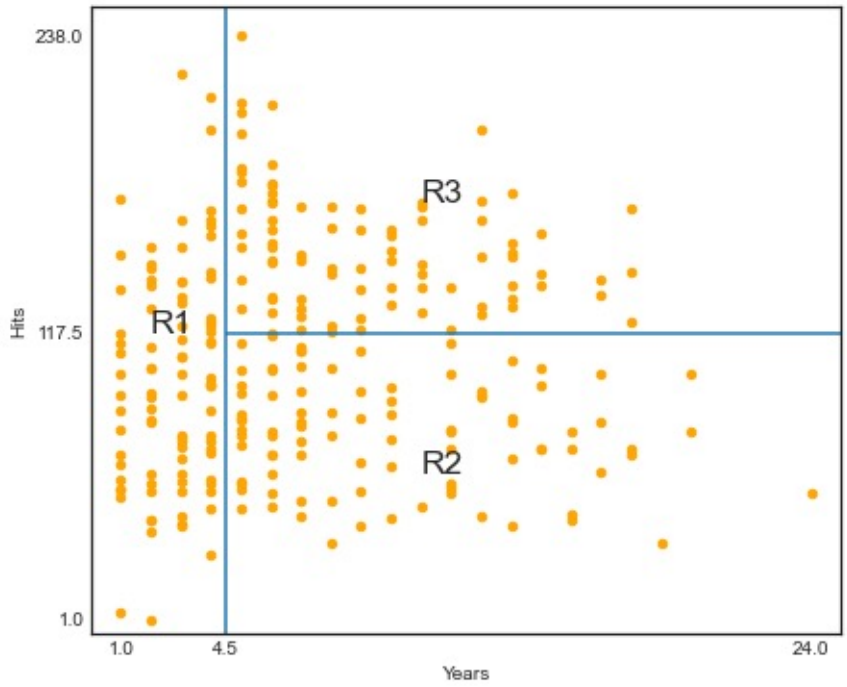
- The lines connecting nodes are called **branches**.
- Trees are typically drawn upside down.

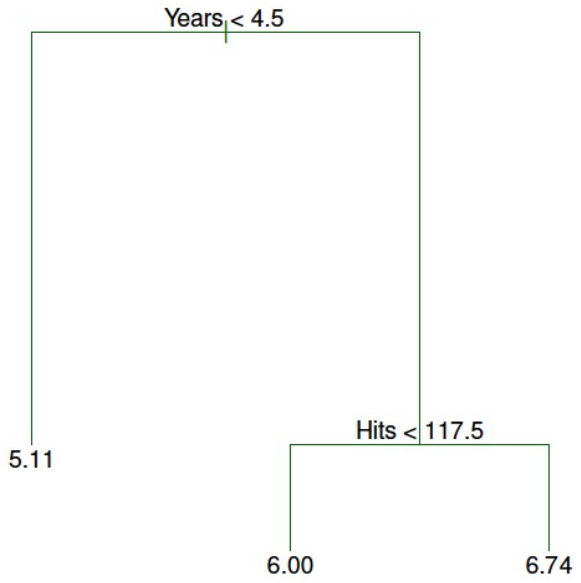
- Each **internal node** represents a splitting.

- At a given internal node,
  the label of the form $X_j < t_k$ indicates the left-hand branch is from that split,
  the right-hand branch corresponds to $X_j \geq t_k$.

  In the example, the two internal nodes are indexed by Year < 4.5 and Hits < 117.5.

Region partitions from trees





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

On Regions R_1, R_2, R_3, the mean-log-salary are 5.11, 6 and 6.74.

Our prediction for any players in R_1, R_2 and R_3 are, respectively

$$1{,}000e^{5.107} = \$165{,}174;\ \ 1000e^{5.999} = \$402{,}834,\ \ and\ \ \ 1{,}000e^{6.740} = \$845{,}346.$$

- Trees involve a series of splitting of the data, each time by one variable.
- The series of actions taken place sequentially creates a tree-like results.
- As in tree figure, the terminal nodes are the three indexed by the numbers, which represent the regions $R_1, R_2\ and\ R_3$. These regions constitute the final partition of the data.

Two step towards prediction:

1. Run the **splitting** according to input values sequentially, and obtain final partition of the data in regions $R_1, \dots, R_J$.

2. For any new observation with covariates $\vec{x}$ in region R_k, we predict its response by the **average** of the responses of the data points in region R_k.

➢ How to split?

Suppose we wish to partition a region R. In other words, we wish to separate the data in region R into two parts, data $R_1 \; and \; R_2$, according to input values.

Question: What would be the optimal or efficient split in some sense?

Only two flexibility in the split:

1. Choice of the input variable to split.
2. The cut point of the split of that chose input.

Imagine that this is the final split of R: $R_1 \; and \; R_2$ would be leaves. We would use the mean response of data in $R_1 \; and \; R_2$ to predict the response of any new/old observations.

We wish our choice of $R_1 \; and \; R_2$ would be optimal in the sense of achieving **minimum prediction error** on the training data in region R.

➢ Recursive binary splitting

A **greedy algorithm** (greedy means it is optimal at the current step):

1. For $j = 1, \ldots, p$ and all real value cutpoint s,

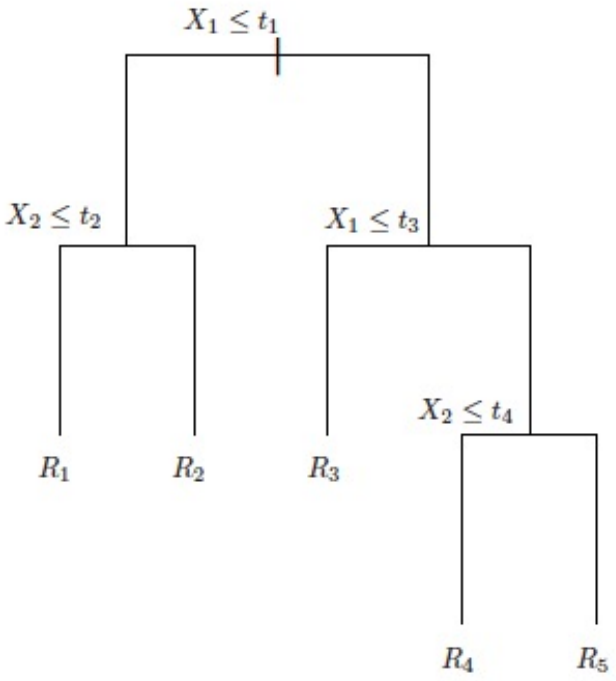
- let $R_1(j,s) = \{\vec{x} \in R \mid x_j < s\}$ and $R_1(j,s) = \{\vec{x} \in R \mid x_j \geq s\}$.
- let $\bar{y}_1 \ and \ \bar{y}_2$ be the mean response of all observations in $R_1(j,s)$ and $R_2(j,s)$.

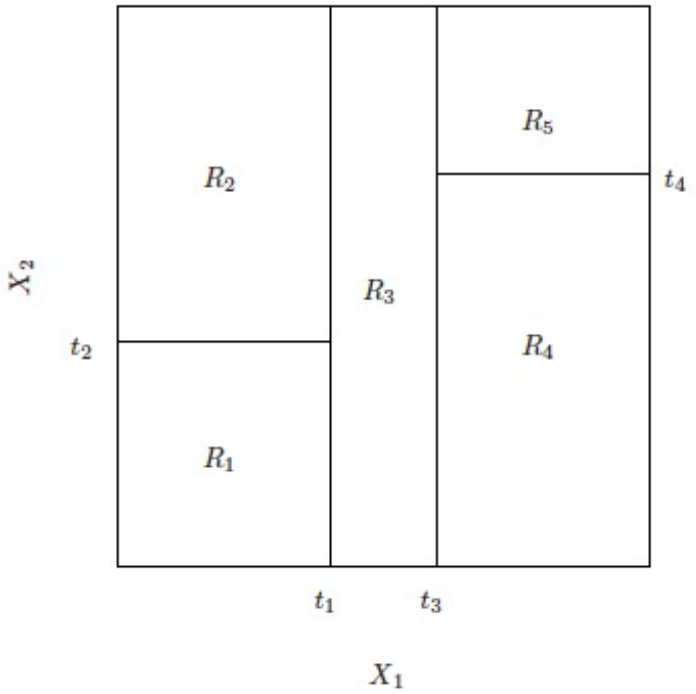
2. Consider the following **RSS prediction error**:

$$RSS_{new} = \sum_{i \in R_1(j,s)} (y_i - \bar{y}_1)^2 + \sum_{i \in R_2(j,s)} (y_i - \bar{y}_2)^2$$

3. Choose the split which has the smallest prediction error. This split is the optimal one, denoted as $R_1 \ and \ R_2$.

4. Continue the split till the final partition.

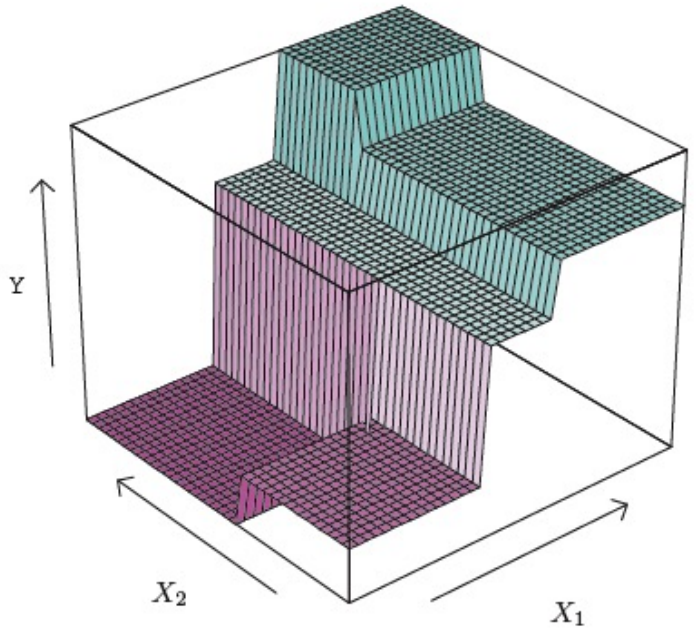




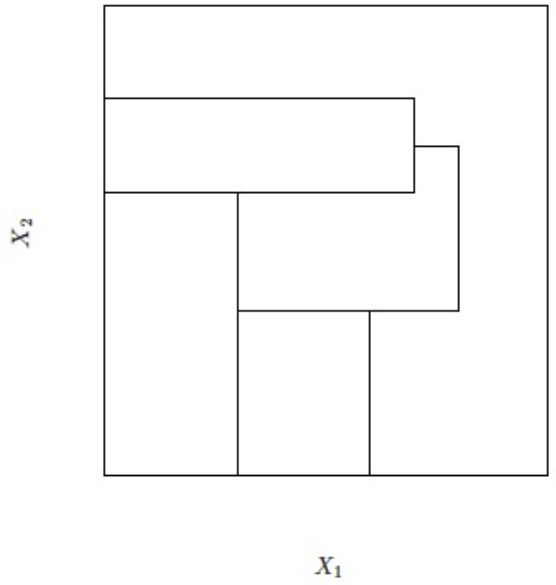
Top Left: A tree corresponding to the partition in the top right panel.

Top right: The output of recursive binary splitting on a two-dimensional example.

Right: A perspective plot of the prediction surface corresponding to that tree.



Example: A partition of two-dimensional feature space that could **not** result from recursive binary splitting.



➢ When to stop the splitting?

- If too many steps of splitting: many leaves, too complex model, small bias but large variance, may overfit.

- If too few steps of splitting: few leaves, too simple model, large bias but small variance, may underfit.

One natural idea: When splitting R into R_1 and R_2, consider the RSS before the split

$$RSS_{old} = \sum_{i \in R} (y_i - \bar{y})^2$$

With the optimal split, the **reduction of RSS**

$$RSS_{old} - RSS_{new}$$

We can pre-choose a threshold, h, and decide the worthiness of the split.

- If the reduction is smaller than h, we do not do it, and stop right there; then R is one terminal node (a leave).
- If the reduction is greater than h, we make the split, and continue with next step.

- The idea is seemingly reasonable, but is too near-sighted.
- Only look at the effect of the current split.
- It is possible that even if the current split is not effective, the future splits could be effective and, maybe, very effective.

➢ Tree pruning

1. Grow a very large tree T_0.
2. Prune the tree back to obtain a subtree.
3. **Objective:** find the subtree that has the best test error.
4. Use *cross-validation* to examine the test errors for a sequence (parametrized by α) of subtrees during the growth/pruning, instead of all possible subtrees which is too large a model space.

➢ Cost complexity pruning (weakest link pruning)

- Let T_0 be the original (large) tree.
- Let T be any subtree.
- Use $|T_0|$ and $|T|$ to denote their **numbers of terminal nodes**, which represent **complexity**.

Consider **"Loss + Penalty":** $$\sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \bar{y}_m)^2 + \alpha |T|$$

Here, R_m are the terminal nodes of the subtree T, and α is tuning parameter.

Denote the minimized subtree as T_α.

- If $\alpha = 0$, no penalty the optimal tree is the original T_0.
- If $\alpha = \infty$, the tree has no split at all. The predictor is just the **mean** $\bar{y}$.
- The larger the α, the more penalty for model complexity.

- Use **cross-validation** to find the best α to minimize the test error.
- There exists efficient algorithm to compute the entire sequence of T next.

➢ The Algorithm for (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 determine best 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 k-th fold of the training data.

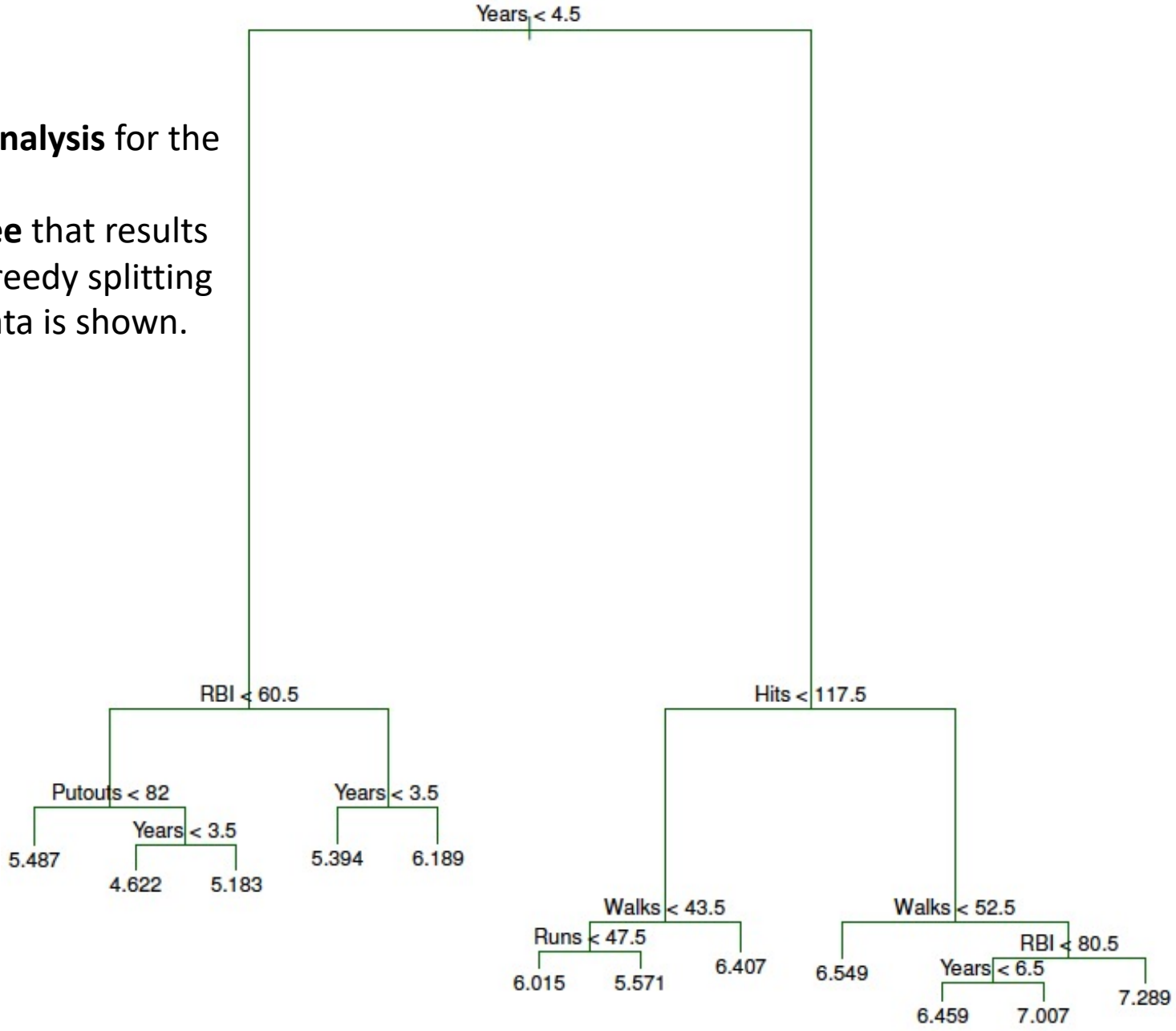
   (b) Evaluate the mean squared prediction error on the data in the left-out k-th fold, as a function of α.

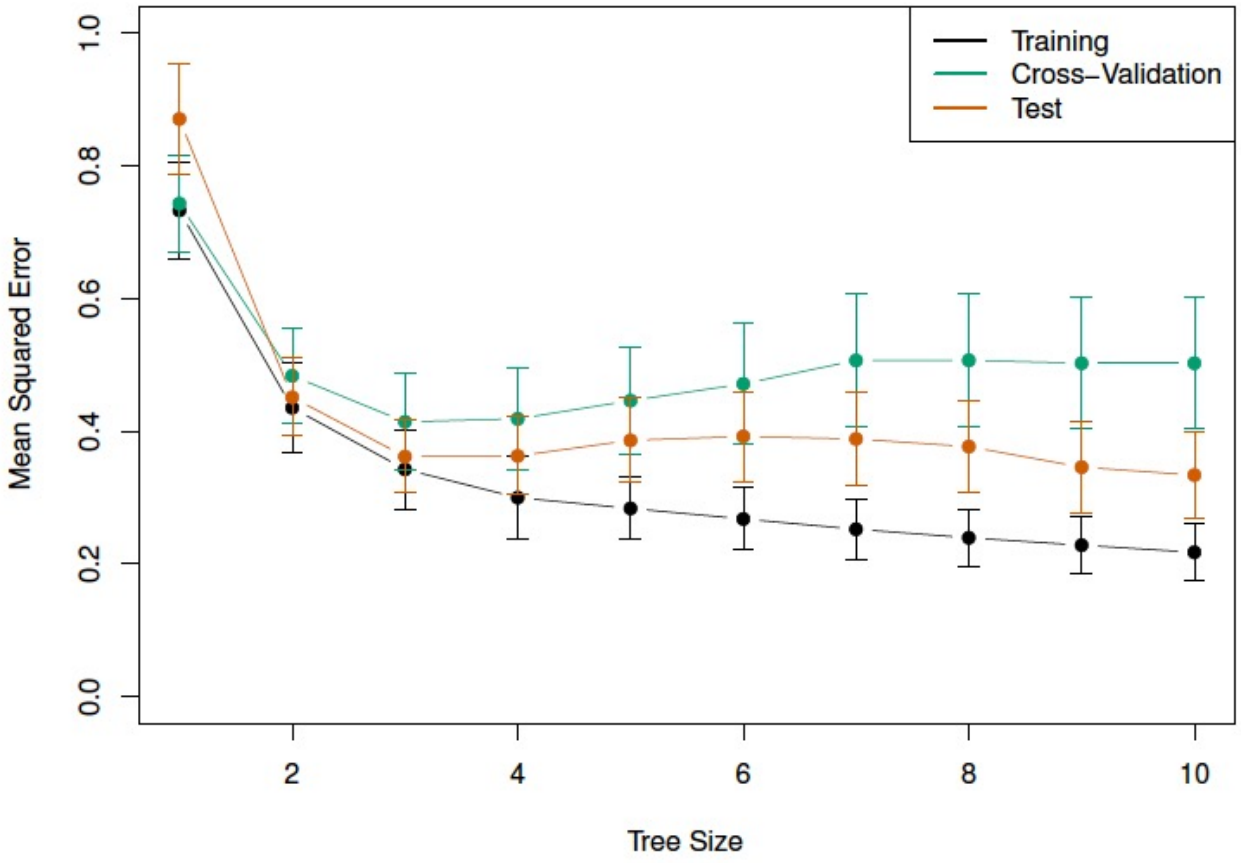
   (c) Average the results for each value of α, and pick α to minimize the average error.

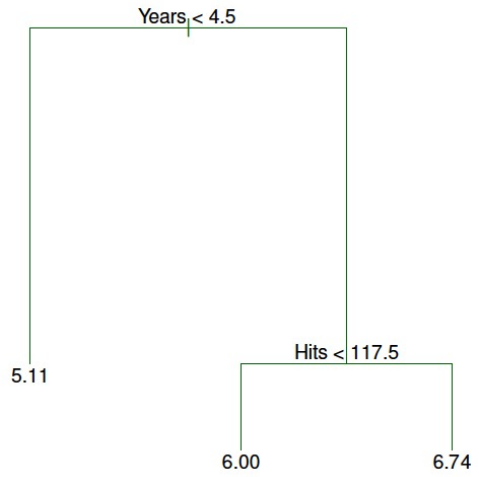
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Example:

- **Regression tree analysis** for the Hitters data.
- The **unpruned tree** that results from top-down greedy splitting on the training data is shown.







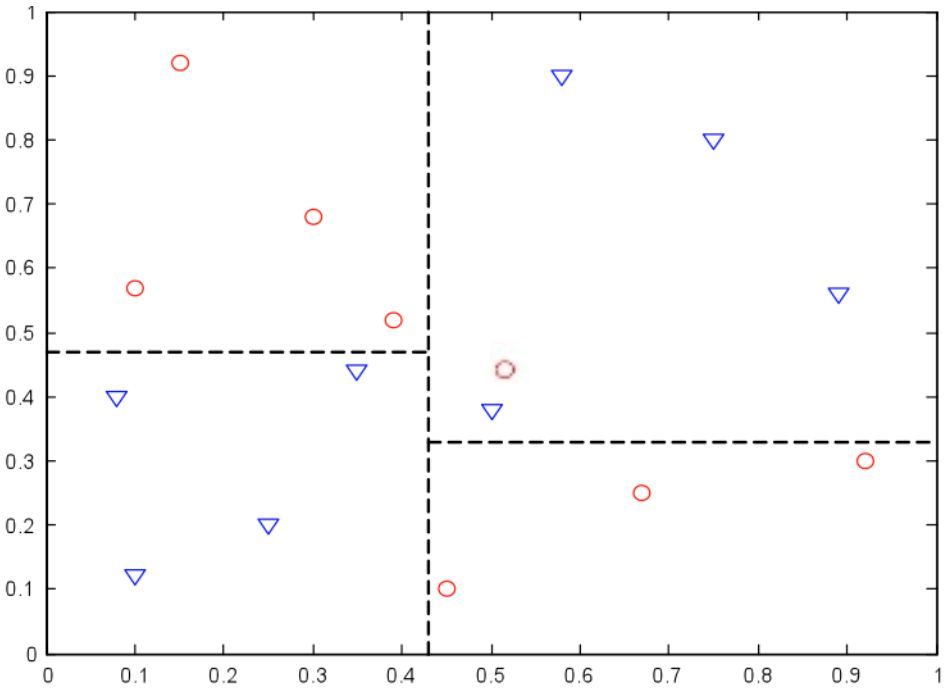
- **Regression tree analysis** for the Hitters data.
- The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree.
- The CV error is a reasonable approximation of the test error
- **Standard error bands** are displayed.
- The minimum cross-validation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

➢ Classification trees

- Regression has **numerical** responses; and classification has **qualitative** responses.
- Recall that for regression trees, we chose to obtain the greatest reduction of RSS. RSS is using sum of squares to measure the error.
- We predict that each observation belongs to the **most commonly occurring class** of training observations in the region to which it belongs.
- For classification trees, one can follow the same line of procedure as that of regression trees, but using different error measurements (e.g., **classification error rate, Gini index, entropy**) that are more appropriate for classification.

Classification error rates

- For a region R, let p_k be the **percentage** of observations in this region that belong to class k.
- We assign any new observation $\vec{x}$ in region R as from the class with largest p_k, which is the called **the most commonly occurring class** in training data.



➢ The impurity measure

The following three approaches are commonly used when pruning the tree.

1. The **classification error rate** (for a region R) is

$$E = 1 - \max_k p_k$$

2. The **Gini index** is

$$G = \sum_{k=1}^{K} p_k(1 - p_k)$$

3. The **cross-entropy** is

$$D = -\sum_{k=1}^{K} p_k \log(p_k)$$

- If R is nearly pure, most of the observations are from one class, then the *Gini-index and cross-entropy* would take smaller values than *classification error rate.*

  For example, in a region R with 3 classes:

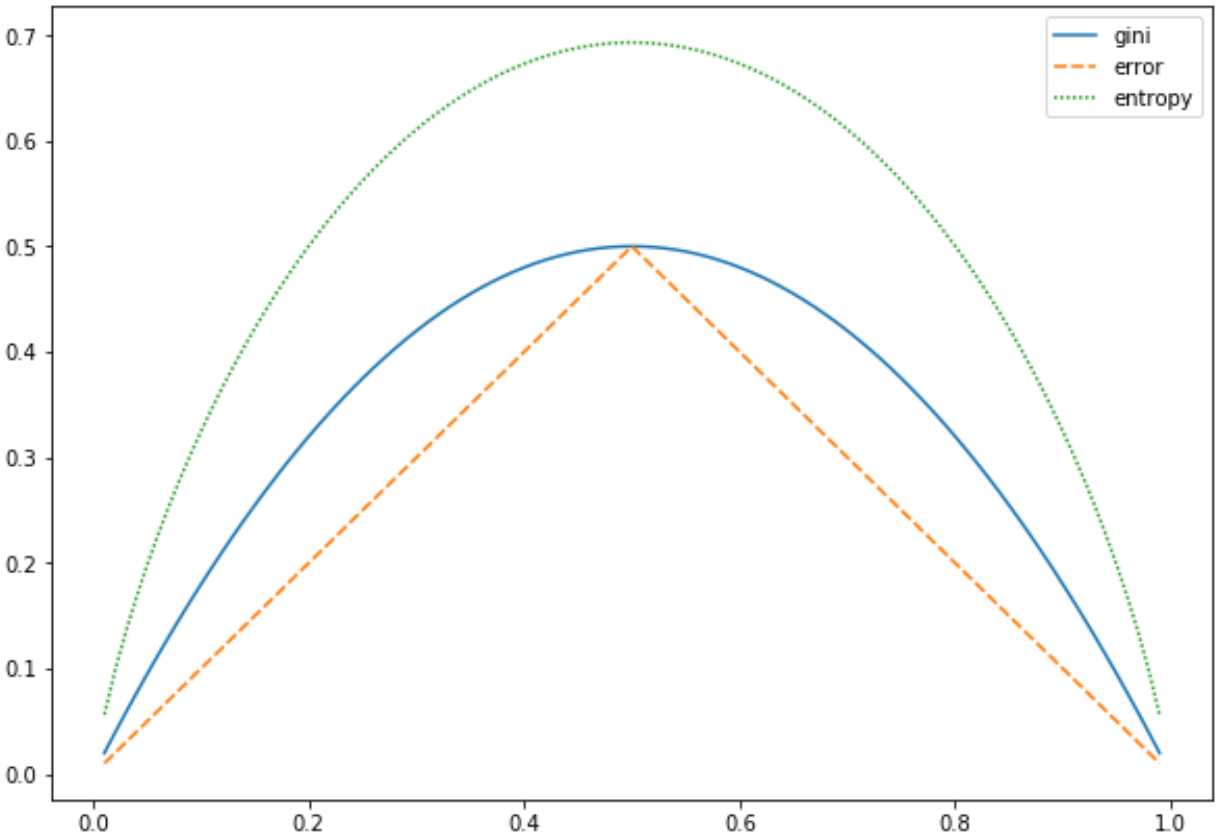
$$p_1 = [0.5, 0.25, 0.25] \Longrightarrow E = 0.5;\ G = 0.625;\ D = 1.0397$$

$$p_2 = [0.5, 0.4;\ 0.1] \Longrightarrow E = 0.5;\ G = 0.580;\ D = 0.9433$$

- *Gini-index and cross-entropy* are more sensitive to node purity. A small value indicates that a node contains predominantly observations from a single class.

- To evaluate the quality of a particular split, the *Gini-index* and *cross-entropy* are more popularly used as error measurement criteria than *classification error rate*. (E can't distinguish p_1 and p_2 above, while G, D are more informative for p_2)

- The *classification error rate* is preferable if only prediction accuracy of the final pruned tree is the goal.


gini
error
entropy
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
1.0

➢ A case study: Heart data.

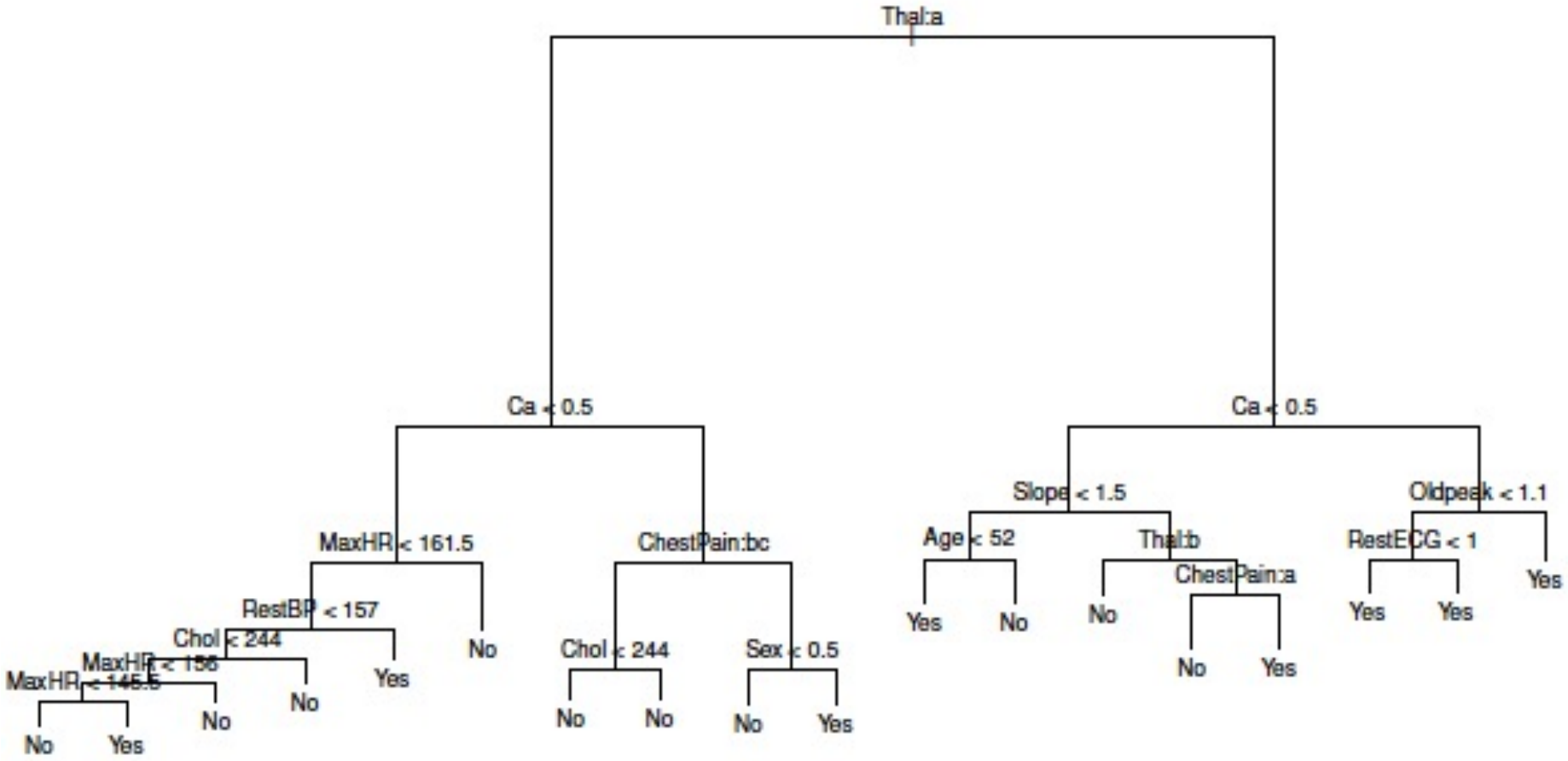
	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
1	63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
2	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
3	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
4	37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
5	41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

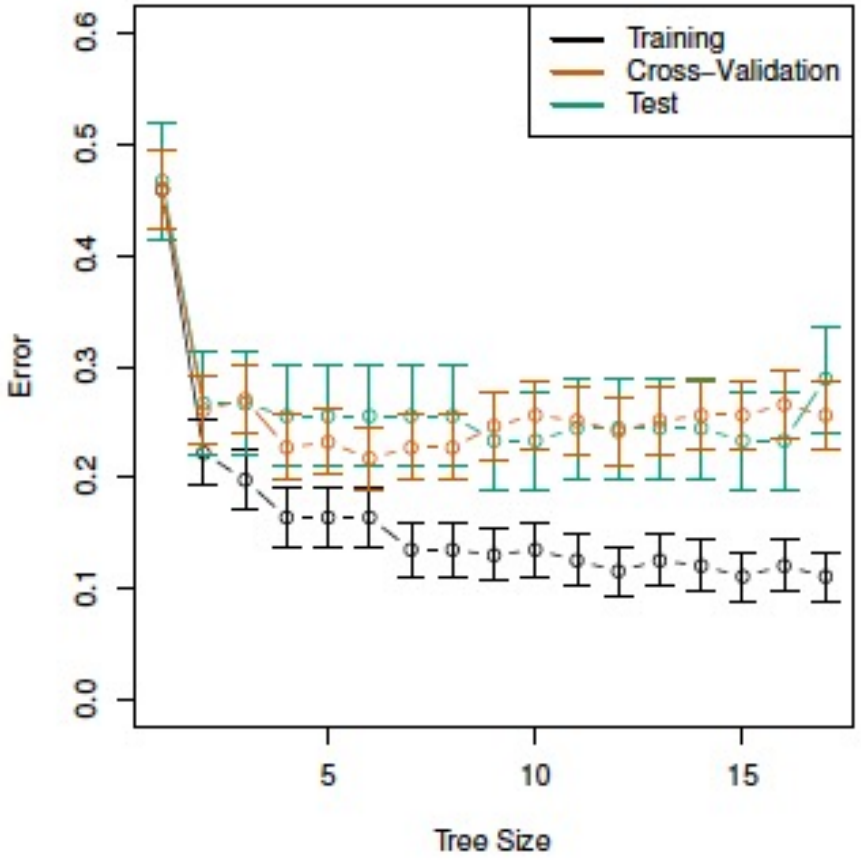
Predictors/Inputs

1.age
2.sex
3.chest pain type (4 values)
4.resting blood pressure
5.serum cholestoral in mg/dl
6.fasting blood sugar > 120 mg/dl
7.resting electrocardiographic results (values 0,1,2)
8.maximum heart rate achieved
9.exercise induced angina
10.oldpeak = ST depression induced by exercise relative to rest
11.the slope of the peak exercise ST segment
12.number of major vessels (0-3) colored by flourosopy
13.thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

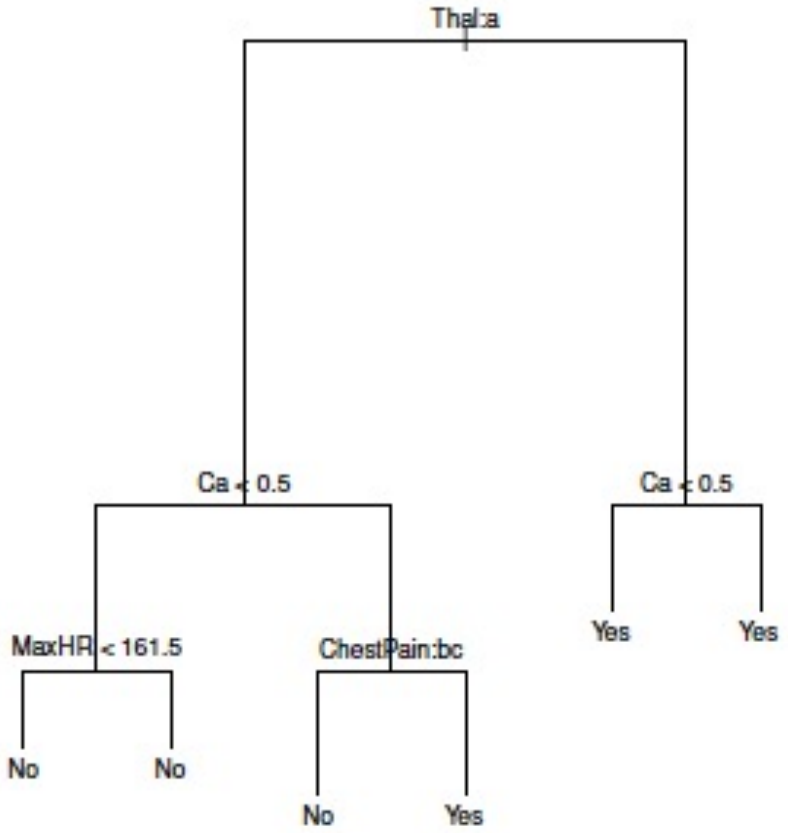
Outputs: AHD: atherosclerotic heart disease.

An unpruned tree.





Cross-validation error, training, and test error, for different sizes of the pruned tree.



The pruned tree corresponding to the minimal cross-validation error.

➢ Trees vs. Linear models

For a **regression model:** $y = h(\vec{x}) + \epsilon$

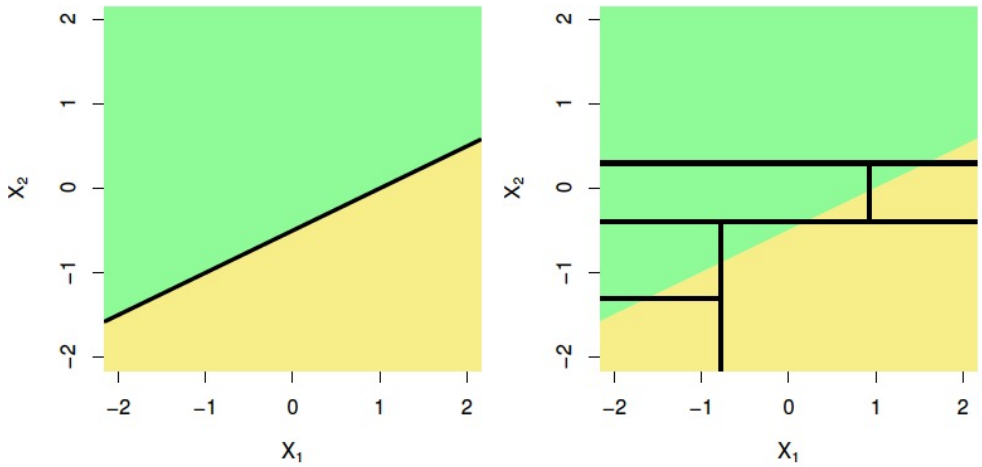
- **Linear model** assumes

$$h(\vec{x}) = \vec{\theta}^T \vec{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$$

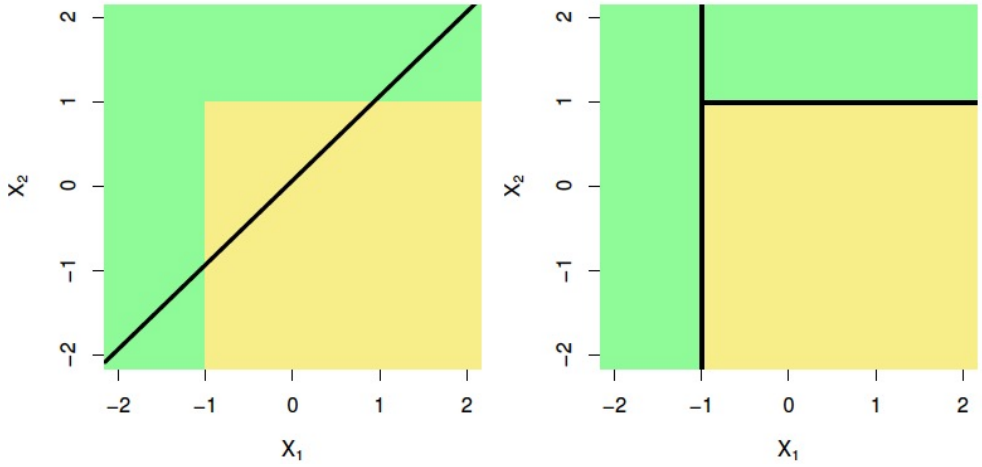
- **Regression trees** assume

$$h(\vec{x}) = \sum_{m=1}^{M} c_m \, \mathbb{I}(\vec{x} \in R_m)$$

If the underlying relation is close to linear, linear model is better.
Otherwise, regression trees are generally better.



A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).



Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).

Advantages 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

- Trees generally do not have the same level of **predictive accuracy** as some of the other regression and classification approaches seen in this book.
- 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.

- However, by aggregating many decision trees, using methods like **bagging**, **random forests, and boosting**, the predictive performance of trees can be substantially improved. We introduce these concepts next.

➢ Bagging (Bootstrap Aggregating)

- **Bagging** is a **general averaging technique** to **reduce variance** of a learning method.
- **Decision tree** is generally a high variance method. (Apply the method based on different data based on same sampling scheme would lead to very different result.)
- **Average** of i.i.d. random variables would have a reduced variance $\frac{\sigma^2}{n}$.

Model $y_i = h(\vec{x}_i) + \epsilon_i$ for $i = 1,2, \dots, n$

Suppose a statistical learning method gives $\hat{h}$ based on the training data $\left(\vec{x}^{(i)}, y^{(i)}\right)$ for $i = 1 \dots n$. For example,

- Linear model: $\hat{h}(\vec{x}) = \vec{\theta}^T \vec{x}$
- K-NN: $\hat{h}(\vec{x}) = \sum_{j=1}^{n/K} \bar{y}_j$ with least distance to K-cluster partition.
- Decision tree: $\hat{h}(\vec{x}) = \sum_{j=1}^{J} \bar{y}_{R_j}$ with rectangular partition.

➢ The procedure of Bagging.

Data $\left(\vec{x}^{(i)}, y^{(i)}\right)$ for $i = 1 \ldots n$ and a learning method $\hat{h}$

- Draw a **bootstrap** sample from the data, and compute $\hat{h}_1^*$ based on this set of bootstrap sample.
- Draw **another bootstrap** sample from the data, and compute $\hat{h}_2^*$ based on this set of bootstrap sample.
- …
- Repeat B times, obtain $\hat{h}_1^*, \ldots, \hat{h}_B^*$
- Produce the **learning method with bagging** as

$$\hat{h}_{bagging} = \frac{1}{B}\sum_{i=1}^{B} \hat{h}_i^*$$

➢ Remarks on Bagging method.

1. Bagging is **general-purpose**.
2. It works best for **high variance low bias** learning methods.
3. When the trees are grown deep, and are not pruned, each individual tree has high variance, but low bias.
4. **Averaging** these trees **reduces the variance**.
5. If the response is qualitative, we can take the **majority vote** (not averaging) of the predicted class based on all learning methods based on bootstrap samples.

➢ Out-of-Bag (OOB) Error Estimation

- Estimation of **test error** for the bagged model.
- For **each** bootstrap sample, observation i is bootstrap sampled with probability $\left(1 - \frac{1}{n}\right)^n \approx 1/e$.
- For **each** bootstrap sample, the number of observations **not** taken into this bootstrap sample is $n\left(1 - \frac{1}{n}\right)^n \approx n/e$. These are referred to as **out-of-bag (OOB) observations.**
- For **totally** B bootstrap samples, about B/e times, the bootstrap sample does not contain observation i.
- For each observation $z_i = \left(\vec{x}^{(i)}, y^{(i)}\right)$, construct its **bagged predictor** by averaging (for regression) or taking majority vote (for classification) of only those trees corresponding to bootstrap samples in which z_i did **not** appear, denoted as $\hat{h}^*_{-i}(\vec{x}^{(i)})$

- The **OOB Mean Squares Error (MSE)** is

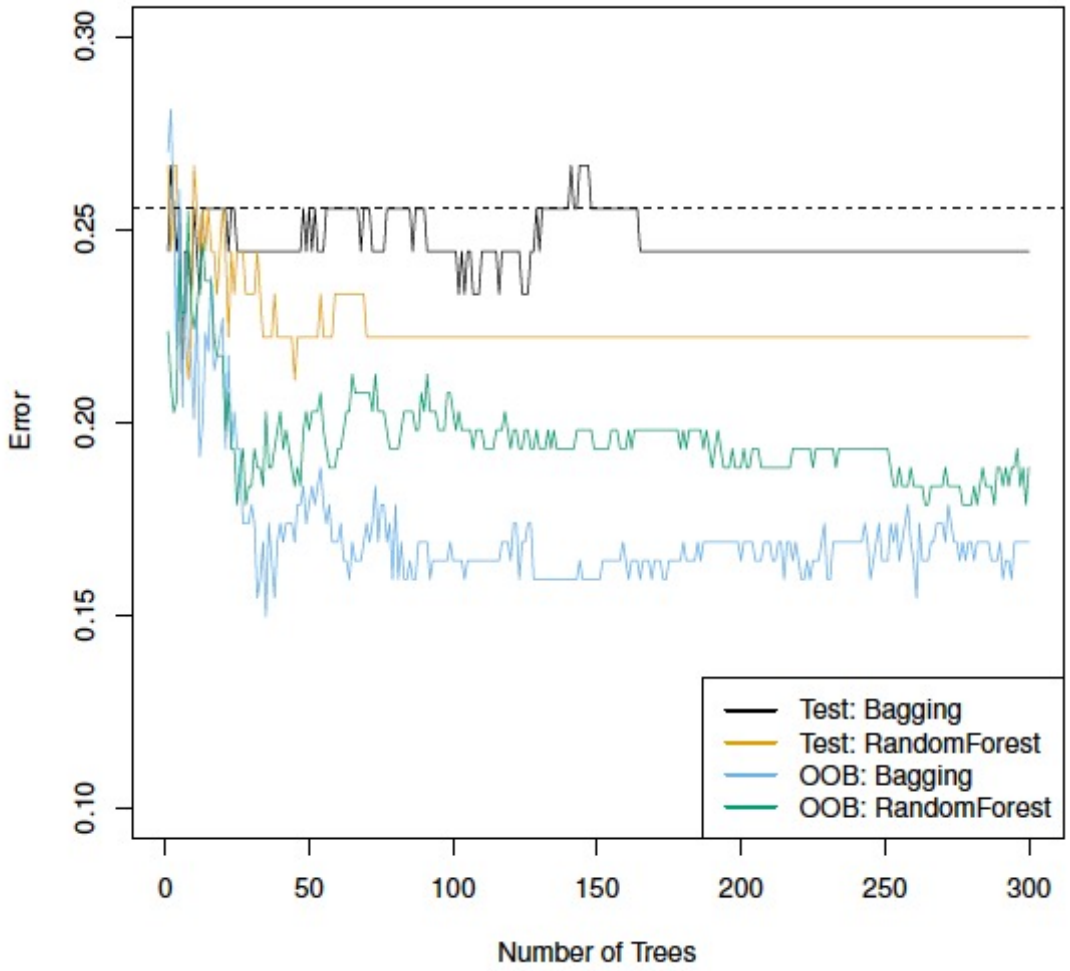
$$\sum_{i=1}^{n} \left(y^{(i)} - \hat{h}^{*}_{-i} \left(\vec{x}^{(i)} \right) \right)^2$$

- The **OOB classification error** is

$$\sum_{i=1}^{n} I \left(y^{(i)} \notin \hat{h}^{*}_{-i} \left(\vec{x}^{(i)} \right) \right)$$

- 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 fit not using that observation.

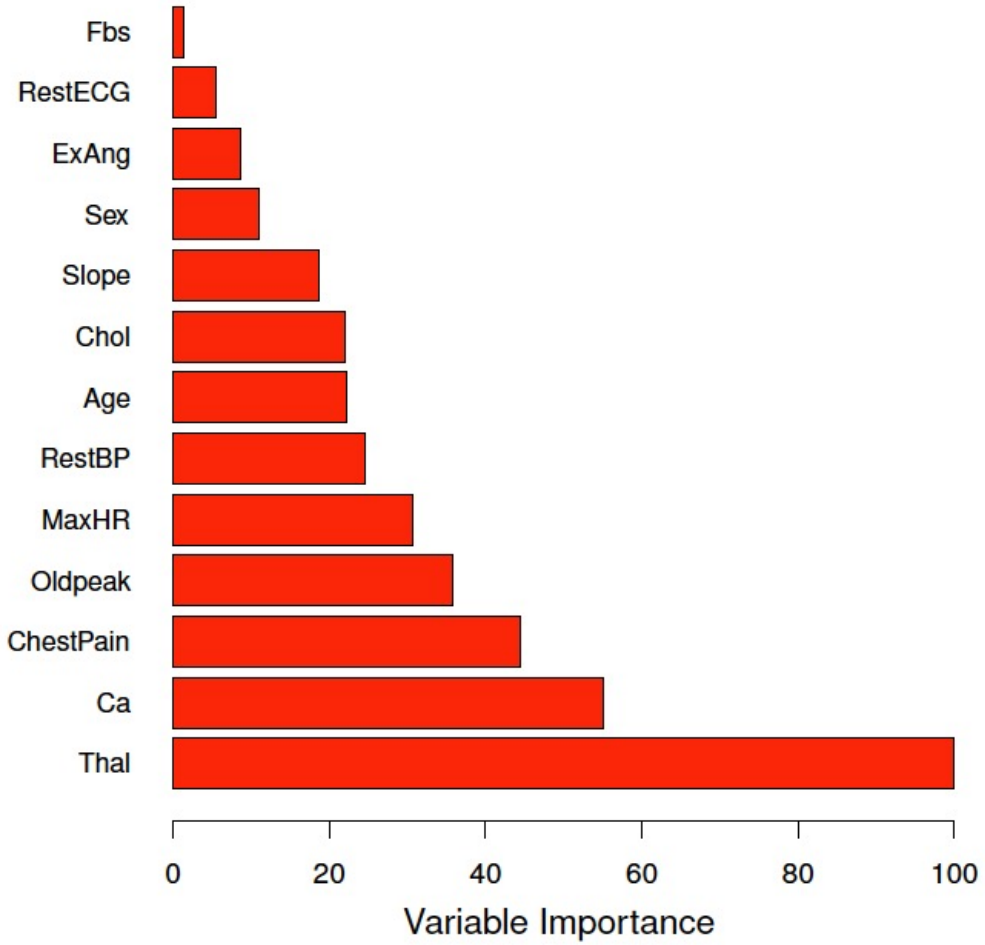
- It can be shown that with B sufficiently large, **OOB error** is virtually equivalent to **leave-one-out** cross-validation error.



Bagging and **random forest** results for the Heart data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The **dashed line** indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.

➢ Variable importance measures

- **Bagging** improves **prediction accuracy** at the expense of **interpretability**.
- An overall **summary of the importance of each predictor** using the RSS (for bagging regression trees) or the Gini index (for bagging classification trees).
- **Bagging regression trees**, we can record the **total amount that the RSS** is **decreased** due to splits over a given predictor, averaged over all B trees.
- A large value indicates an important predictor.
- **Bagging classification trees**, we can add up the **total amount that the Gini index** is decreased by splits over a given predictor, averaged over all B trees.



- 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.

➢ Motivation of Random Forest

An average of B i.i.d random variables, each with variance σ^2, has variance $\frac{1}{B}\sigma^2$

❖ **Question:** What if **not independent** but **correlated** identical random variables?

- If the variables are simply i.d. (identically distributed but not necessarily independent) with positive pairwise correlation ρ, the variance of the average is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

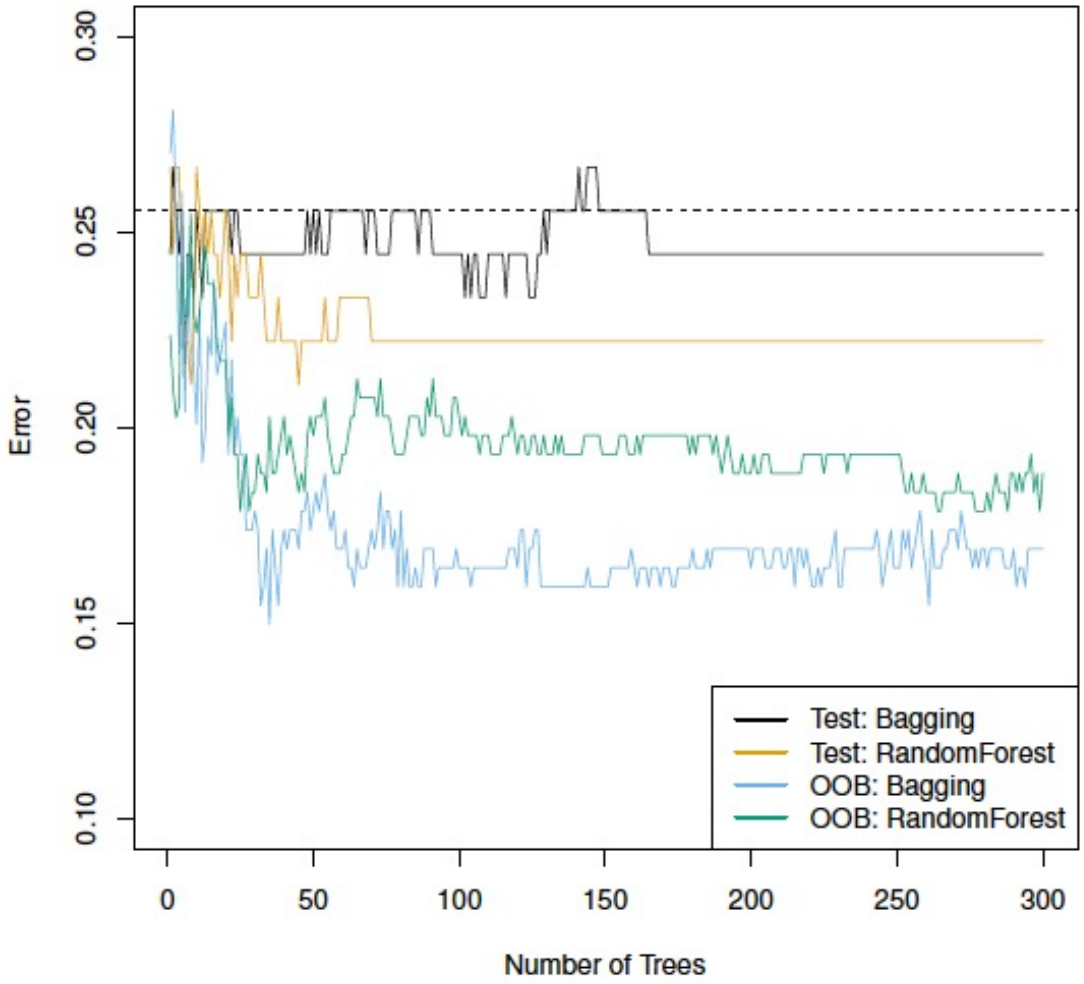
- The idea of random forests is to improve the variance reduction of bagging by **reducing the correlation between trees**, without increasing the variance too much.

Random forests is the same as bagging decision trees, except ...

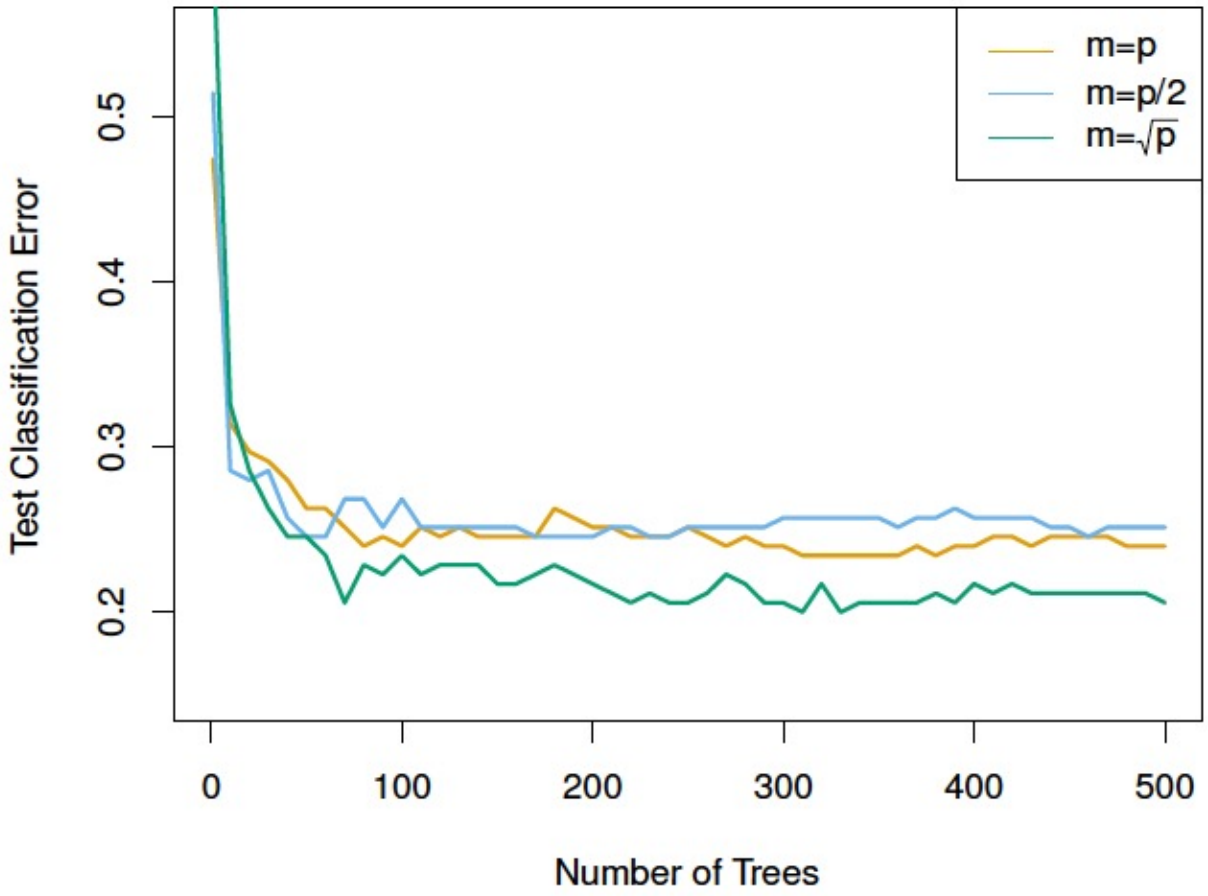
- When building these decision trees, **each time** a split in a tree is considered, a **random sample of m predictors is chosen** as split candidates from the full set of p predictors. Typically, $m \approx \sqrt{p}$.

➢ Random Forest (improved bagging tree method)

- **Goal:** Avoid all trees are too similar to each other.
- Too similar trees are too highly correlated, average highly correlated trees cannot achieve large amount of variance reduction. **Extreme case:** If all trees are the same, average of them is still the same one.
- Every step of random forest , the split is constrained on a small number m and **randomly** selected inputs.
- Random forest produces less correlated trees.
- Random forest reduces to bagging if $m = p$.
- Averaging uncorrelated or low-correlated trees can achieve large amount of **variance reduction**.



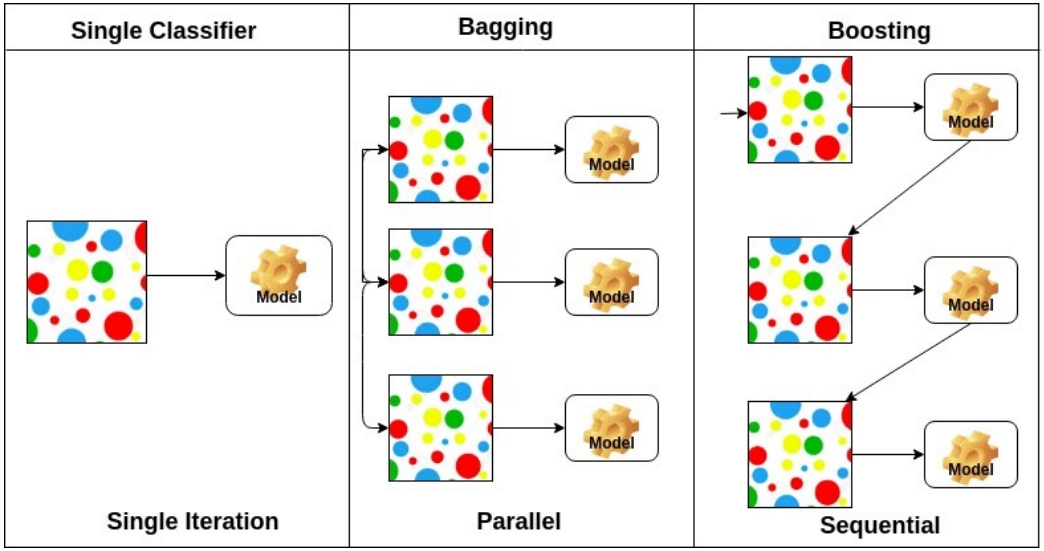
Bagging and **random forest** results for the Heart data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The **dashed line** indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.



Results from **random forests** for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a **different value of m**, the number of predictors available for splitting at each interior tree node. Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

➢ Boosting

- **General purpose** for improving learning methods **by combining many weaker learners** in attempt to produce a strong learner.
- Like bagging, boosting involves combining a large number of weaker learners.
- The weaker learners are created **sequentially** (no bootstrap involved).
- **Bagging** create large variance and possibly overfit bootstrap learners and try to reduce their variance by averaging.
- **Boosting** create weak learners sequentially and slowly (to avoid overfit).



➢ Algorithm: Boosting for Regression Trees

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

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

   - Fit a tree $\hat{h}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   - Update $\hat{h}$ by adding in a shrunken version of the new tree:

$$\hat{h}(\vec{x}) = \hat{h}(\vec{x}) + \lambda \hat{h}^b(\vec{x})$$

   - Update the residuals,

$$r_i = r_i - \lambda \hat{h}^b(x_i)$$

3. Output the **boosted model**,

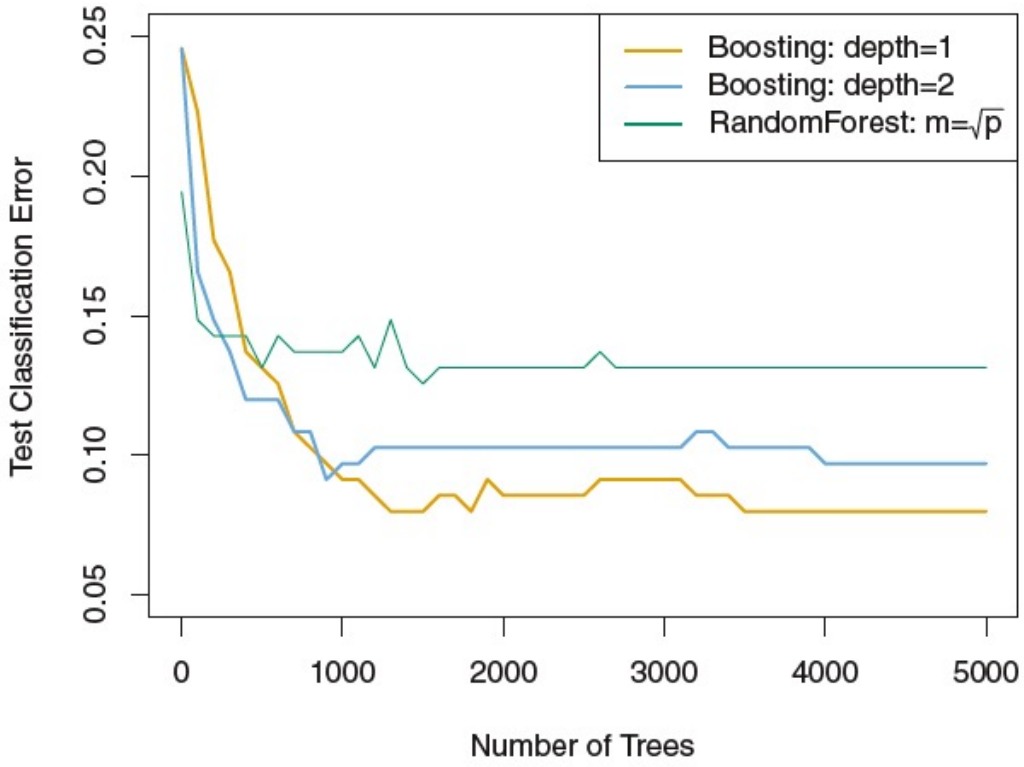
$$\hat{h}(\vec{x}) = \sum_{b=1}^{B} \lambda \hat{h}^b(\vec{x})$$

Boosting has 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. We use cross-validation to select B.

2. The shrinkage parameter λ, a small positive number.

3. The number d of splits in each tree, which controls the complexity of the boosted ensemble.



Results from performing boosting and random forests on the 15-class gene expression data set in order to predict cancer versus normal. The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant. The test error rate for a single tree is 24%.

➢ Adaboost

1. **Initialize** the **data weights** $\{w_i\}$ by setting $w_i^{(1)} = 1/N$ for i = $1, \ldots, N$.

2. For $m = 1, \ldots, M$:

   (a) Fit a classifier $f_m(\vec{c})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{i=1}^{N} w_i^{(m)} \, \mathbb{I}(f_m(\vec{x}^{(i)}) \neq y^{(i)}$$

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \, \mathbb{I}(f_m(\vec{x}^{(i)}) \neq y^{(i)}}{\sum_{i=1}^{N} w_i^{(m)}}$$

   and then use these to evaluate

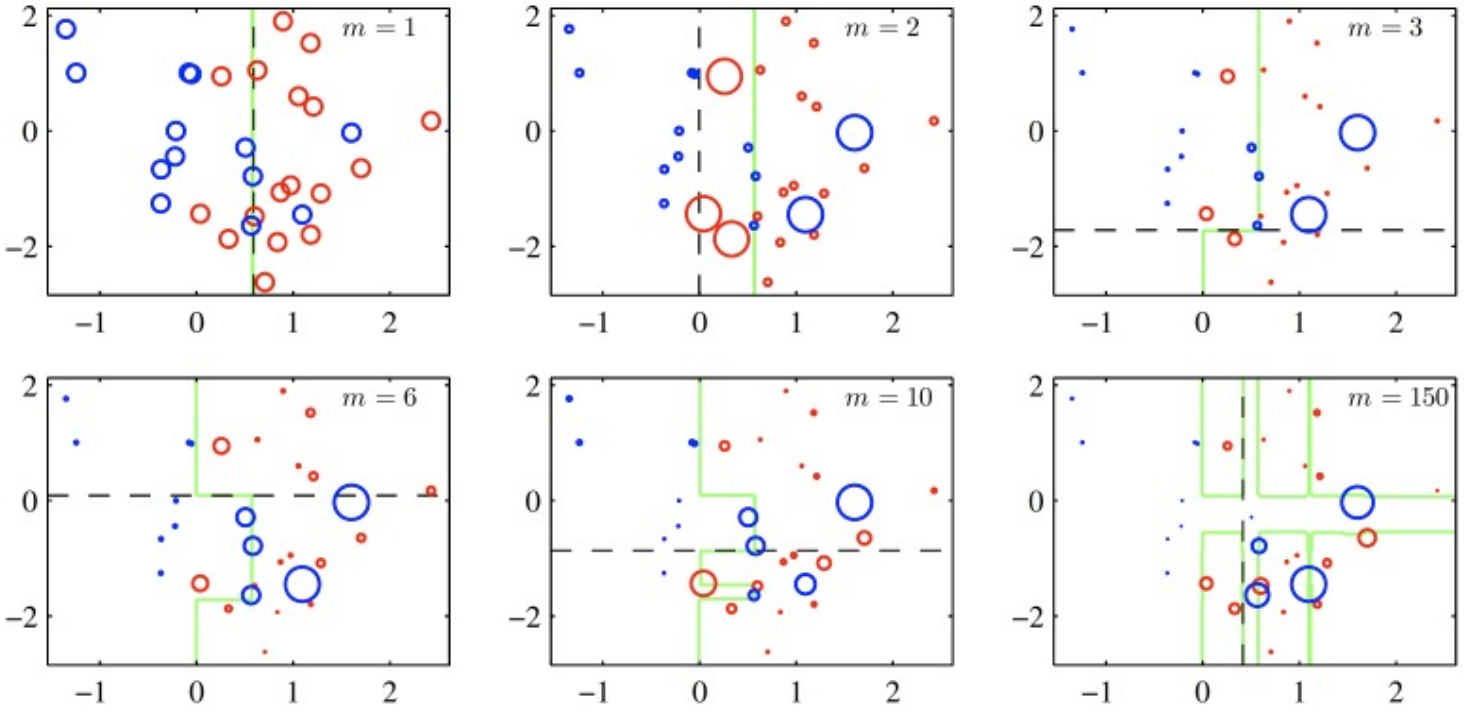
$$\alpha_m = \log\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}$$

   (c) Update the data weights

$$w_i^{(m+1)} = w_i^{(m+1)} \exp\{\alpha_m \mathbb{I}(f_m(\vec{x}^{(i)}) \neq y^{(i)})\}$$

3. Make prediction by

$$F_M(\vec{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m f_m(\vec{x})\right).$$



Each figure shows the number m of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the $m = 1$ base learner are given greater weight when training $the\ m = 2$ base learner.

Summary: Statistical view of boosting methods

Boosting methods have three important properties that contribute to their success:

1. They fit (by coordinate descent) an additive model in a flexible set of basis functions.
2. They use a suitable loss function for the fitting process.
3. They regularize by forward stagewise fitting; with shrinkage this mimics an L_1 (Lasso) penalty on the weights.