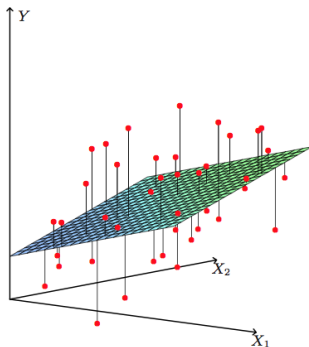


# Statistical Methods for Data Mining

Kuangnan Fang

Xiamen University Email: [xmufkn@xmu.edu.cn](mailto:xmufkn@xmu.edu.cn)



# Tree-based Methods

- Here we describe *tree-based* methods for regression and classification.
- These involve *stratifying* or *segmenting* 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.

## Pros and Cons

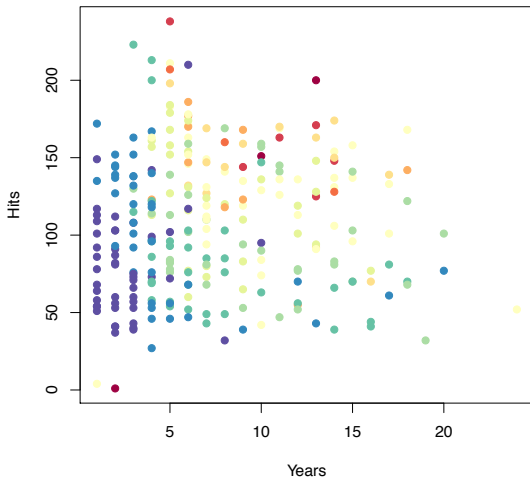
- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss *bagging*, *random forests*, and *boosting*. These methods grow multiple trees which are then 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.

# The Basics of Decision Trees

- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.

## Baseball salary data: how would you stratify it?

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



## Decision tree for these data

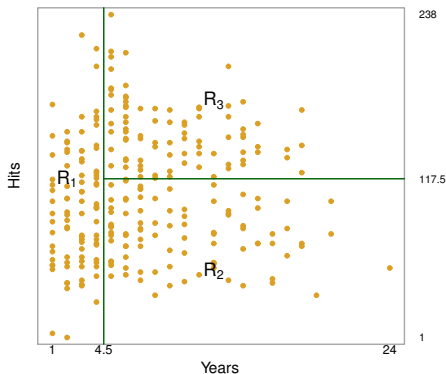


## Details of previous figure

- 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  $X_j < t_k$ ) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to  $X_j \geq t_k$ . 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.

## Results

- Overall, the tree stratifies or segments the players into three regions of predictor space:  $R_1 = \{X \mid \text{Years} < 4.5\}$ ,  $R_2 = \{X \mid \text{Years} \geq 4.5, \text{Hits} < 117.5\}$ , and  $R_3 = \{X \mid \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$ .



# Terminology for Trees

- In keeping with the *tree* analogy, the regions  $R_1$ ,  $R_2$ , and  $R_3$  are known as *terminal nodes*
- 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*
- In the hitters tree, the two internal nodes are indicated by the text **Years**<4.5 and **Hits**<117.5.

## Interpretation of Results

- **Years** is the most important factor in determining **Salary**, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of **Hits** that he made in the previous year seems to play little role in his **Salary**.
- But among players who have been in the major leagues for five or more years, the number of **Hits** made in the previous year does affect **Salary**, and players who made more **Hits** last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

## Details of the tree-building process

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

## More details of the tree-building process

- 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 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.

## Details— Continued

- We first select the predictor  $X_j$  and the cutpoint  $s$  such that splitting the predictor space into the regions  $\{X|X_j < s\}$  and  $\{X|X_j \geq s\}$  leads to the greatest possible reduction in RSS.
- Next, 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.
- 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; for instance, we may continue until no region contains more than five observations.

# Regression Trees

- data  $(x_i, y_i)$  for  $i = 1, 2, \dots, N$ , with  $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$ .
- Suppose first that we have a partition into  $M$  regions  $R_1, R_2, \dots, R_M$ , and we model the response as a constant  $c_m$  in each region:

$$f(x) = \sum_{m=1}^M c_m I(x \in R_m)$$

- If we adopt as our criterion minimization of the sum of squares  $\sum (y_i f(x_i))^2$ , it is easy to see that the best  $\hat{c}_m$  is just the average of  $y_i$  in region  $R_m$

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m)$$

## Regression Trees

- Starting with all of the data, consider a splitting variable  $j$  and split point  $s$ , and define the pair of half-planes

$$R_1(j, s) = X | X_j \leq s \text{ and } R_2(j, s) = X | X_j > s$$

- Then we seek the splitting variable  $j$  and split point  $s$  that solve

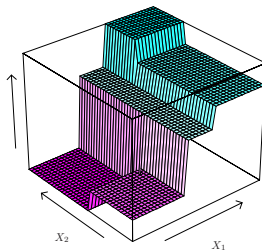
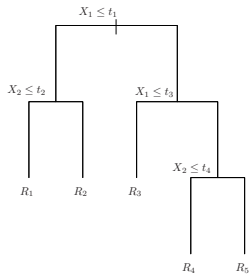
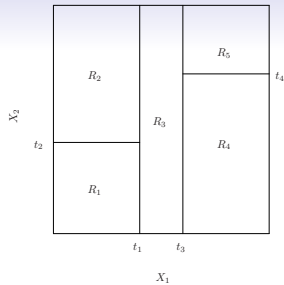
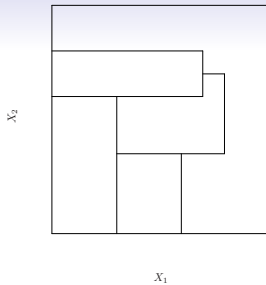
$$\min_{j, s} \left[ \min_{c_1} \sum_{x_i \in R_1(j, s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j, s)} (y_i - c_2)^2 \right].$$

For any choice  $j$  and  $s$ , the inner minimization is solved by

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j, s)) \quad \text{and} \quad \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j, s)).$$

# Predictions

- 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 of this approach is shown in the next slide.



## 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. *Why?*

## 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. *Why?*
- A smaller tree with fewer splits (that is, fewer regions  $R_1, \dots, R_J$ ) might lead to lower variance and better interpretation at the cost of a little bias.
- 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— continued

- A better strategy is to grow a very large tree  $T_0$ , and then *prune* it back in order to obtain a *subtree*
- *Cost complexity pruning* — also known as *weakest link pruning* — is used to do this
- we consider a sequence of trees indexed by a nonnegative tuning parameter  $\alpha$ . For each value of  $\alpha$  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. Here  $|T|$  indicates the number of terminal nodes of the tree  $T$ ,  $R_m$  is the rectangle (i.e. the subset of predictor space) corresponding to the  $m$ th terminal node, and  $\hat{y}_{R_m}$  is the mean of the training observations in  $R_m$ .

## 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  $\hat{\alpha}$ .

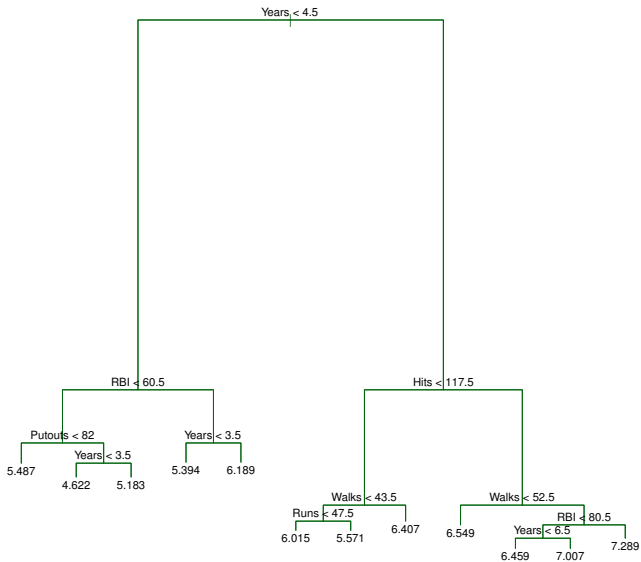
## Summary: tree algorithm

1. Use recursive binary splitting to grow a large tree 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  $\alpha$ .
3. Use K-fold cross-validation to choose  $\alpha$ . For each  $k = 1, \dots, K$ :
  - 3.1 Repeat Steps 1 and 2 on the  $\frac{K-1}{K}$ th fraction of the training data, excluding the  $k$ th fold.
  - 3.2 Evaluate the mean squared prediction error on the data in the left-out  $k$ th fold, as a function of  $\alpha$ .Average the results, and pick  $\alpha$  to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of  $\alpha$ .

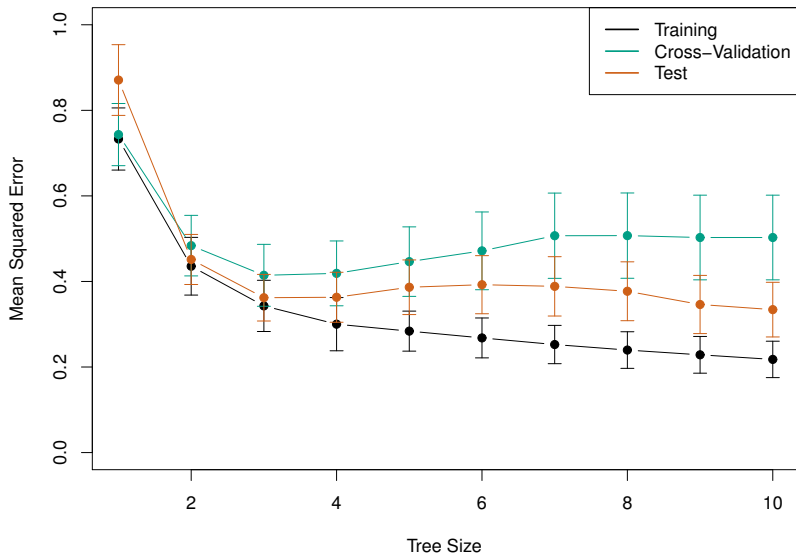
## Baseball example continued

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied  $\alpha$  in in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of  $\alpha$ .

## Baseball example continued



## Baseball example continued



# Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- 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.

## Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- 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*. this is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

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

- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

## Gini index and Deviance

- The *Gini index* is defined by

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

a measure of total variance across the  $K$  classes. The Gini index takes on a small value if all of the  $\hat{p}_{mk}$ 's are close to zero or one.

- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.

## Gini index and Deviance

- The *Gini index* is defined by

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

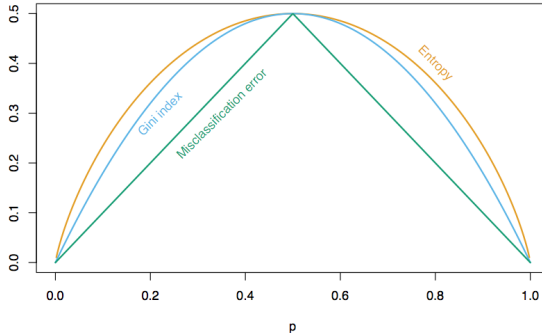
a measure of total variance across the  $K$  classes. The Gini index takes on a small value if all of the  $\hat{p}_{mk}$ 's are close to zero or one.

- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is *cross-entropy*, given by

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

- It turns out that the Gini index and the cross-entropy are very similar numerically.

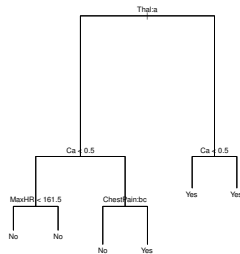
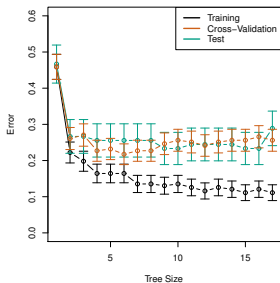
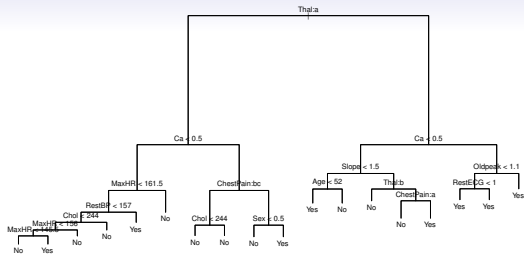
# Classification Trees



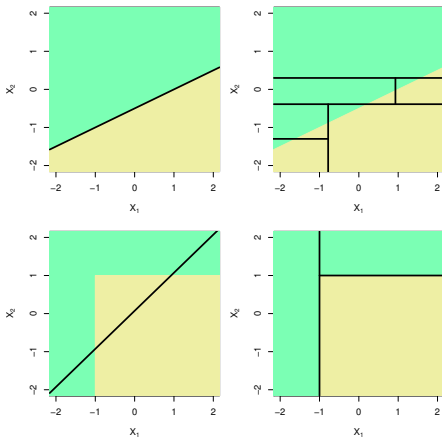
**FIGURE 9.3.** Node impurity measures for two-class classification, as a function of the proportion  $p$  in class 2. Cross-entropy has been scaled to pass through  $(0.5, 0.5)$ .

## Example: heart data

- 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.
- Cross-validation yields a tree with six terminal nodes. See next figure.



## Trees Versus Linear Models



Top Row: True linear boundary; Bottom row: true non-linear boundary.

Left column: linear model; Right column: tree-based model

## Advantages and Disadvantages 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.
- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

## More for Decision Trees

- Why Binary Splits?
- Linear Combination Splits. Rather than restricting splits to be of the form  $X_j \leq s$ , one can allow splits along linear combinations of the form  $a_j X_j \leq s$ . The weights  $a_j$  and split point  $s$  are optimized to minimize the relevant criterion (such as the Gini index). While this can improve the predictive power of the tree, it can hurt interpretability.
- Instability of Trees. One major problem with trees is their high variance. Often a small change in the data can result in a very different series of splits, making interpretation somewhat precarious. The major reason for this instability is the hierarchical nature of the process: the effect of an error in the top split is propagated down to all of the splits below it.
- Lack of Smoothness.
- Difficulty in Capturing Additive Structure.  
$$Y = c_1 I(X_1 < t_1) + c_2 I(X_2 < t_2) + \text{where } \text{is zero-mean noise.}$$

# Bagging

- *Bootstrap aggregation*, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of  $n$  independent observations  $Z_1, \dots, Z_n$ , each with variance  $\sigma^2$ , the variance of the mean  $\bar{Z}$  of the observations is given by  $\sigma^2/n$ .
- In other words, *averaging a set of observations reduces variance*. Of course, this is not practical because we generally do not have access to multiple training sets.

## Bagging— continued

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate  $B$  different bootstrapped training data sets. We then train our method on the  $b$ th bootstrapped training set in order to get  $\hat{f}^{*b}(x)$ , the prediction at a point  $x$ . We then average all the predictions to obtain

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

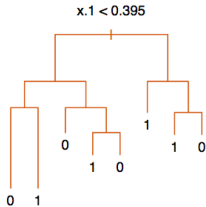
This is called *bagging*.

## Bagging classification trees

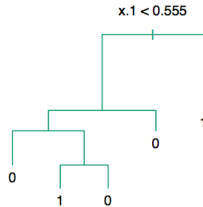
- The above prescription applied to regression trees
- For classification trees: for each test observation, we record the class predicted by each of the  $B$  trees, and take a *majority vote*: the overall prediction is the most commonly occurring class among the  $B$  predictions.

# bagging

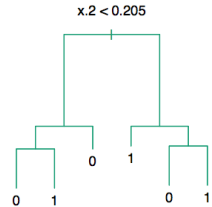
**Original Tree**



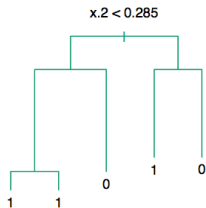
**b = 1**



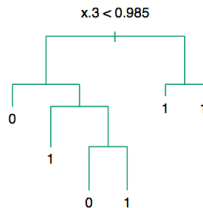
**b = 2**



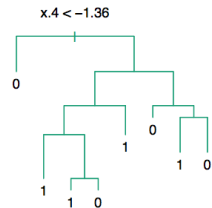
**b = 3**



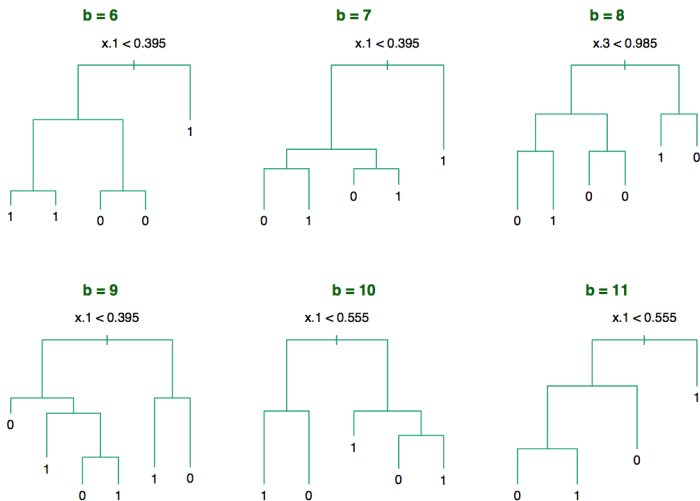
**b = 4**



**b = 5**

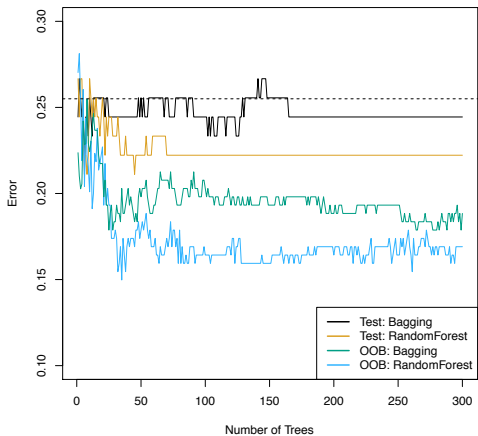


# bagging



**FIGURE 8.9.** Bagging trees on simulated dataset. The top left panel shows the original tree. Eleven trees grown on bootstrap samples are shown. For each tree, the top split is annotated.

# Bagging the heart data



## Details of previous figure

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

## Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the *out-of-bag* (OOB) observations.
- We can predict the response for the  $i$ th observation using each of the trees in which that observation was OOB. This will yield around  $B/3$  predictions for the  $i$ th observation, which we average.
- This estimate is essentially the LOO cross-validation error for bagging, if  $B$  is large.

## Random Forests

- *Random forests* provide an improvement over bagged trees by way of a small tweak that *decorrelates* the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, *a random selection of  $m$  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.
- A fresh selection of  $m$  predictors is taken at each split, and typically we choose  $m \approx \sqrt{p}$  — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).

# Random Forest

---

**Algorithm 15.1** *Random Forest for Regression or Classification.*

---

1. For  $b = 1$  to  $B$ :
  - (a) Draw a bootstrap sample  $\mathbf{Z}^*$  of size  $N$  from the training data.
  - (b) Grow a random-forest tree  $T_b$  to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{min}$  is reached.
    - i. Select  $m$  variables at random from the  $p$  variables.
    - ii. Pick the best variable/split-point among the  $m$ .
    - iii. Split the node into two daughter nodes.
2. Output the ensemble of trees  $\{T_b\}_1^B$ .

To make a prediction at a new point  $x$ :

*Regression:*  $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$ .

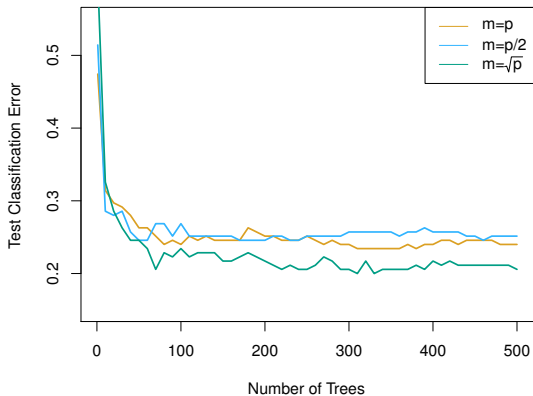
*Classification:* Let  $\hat{C}_b(x)$  be the class prediction of the  $b$ th random-forest tree. Then  $\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$ .

---

## Example: gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables  $m$ .

## Results: gene expression data



## Details of previous figure

- Results from random forests for the fifteen-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

- Consider two-class problem,  $Y \in \{-1, 1\}$ , Given a vector of predictor variables  $X$ , a classifier  $G(X)$  produces a prediction taking one of the two values  $\{1, -1\}$
- Given a vector of predictor variables  $X$ , a classifier  $G(X)$  produces a prediction taking one of the two values  $\{1, -1\}$

$$\bar{err} = \frac{1}{N} \sum_{i=1}^N I(y_i \neq G(x_i))$$

and the expected error rate on future predictions is  $E_{XY} I(Y \neq G(X))$

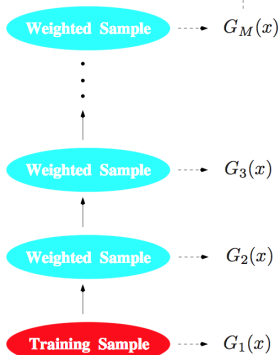
- The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers  $G_m(x), m = 1, 2, \dots, M$ .

# Boosting

- The predictions from all of them are then combined through a weighted majority vote to produce the final prediction:

$$G(x) = \text{sign} \left( \sum_{m=1}^M G_m(x) \right)$$

$$G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$$



# Boosting

---

**Algorithm 10.1** *AdaBoost.M1*.

---

1. Initialize the observation weights  $w_i = 1/N$ ,  $i = 1, 2, \dots, N$ .
  2. For  $m = 1$  to  $M$ :
    - (a) Fit a classifier  $G_m(x)$  to the training data using weights  $w_i$ .
    - (b) Compute
$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
    - (c) Compute  $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$ .
    - (d) Set  $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$ ,  $i = 1, 2, \dots, N$ .
  3. Output  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ .
-

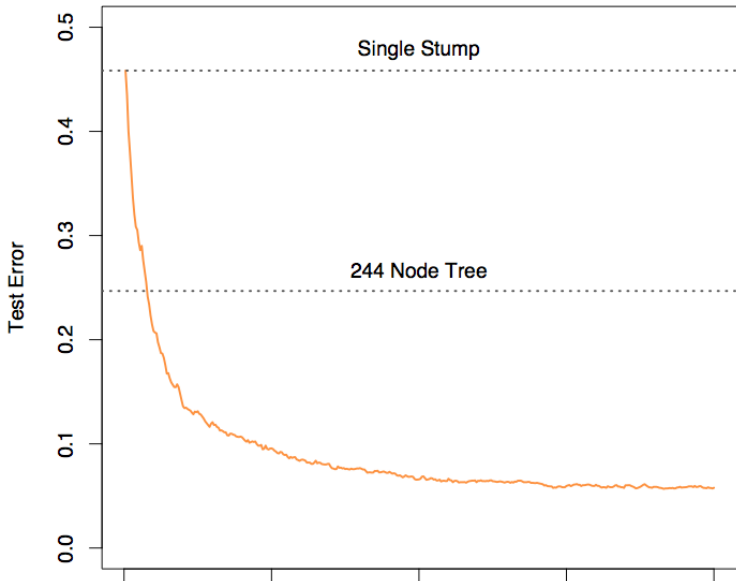
# Boosting

The power of AdaBoost to dramatically increase the performance of even a very weak classifier is illustrated in Figure 10.2. The features  $X_1, \dots, X_{10}$  are standard independent Gaussian, and the deterministic target  $Y$  is defined by

$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5), \\ -1 & \text{otherwise.} \end{cases} \quad (10.2)$$

Here  $\chi_{10}^2(0.5) = 9.34$  is the median of a chi-squared random variable with 10 degrees of freedom (sum of squares of 10 standard Gaussians). There are 2000 training cases, with approximately 1000 cases in each class, and 10,000 test observations. Here the weak classifier is just a “stump”: a two terminal-node classification tree. Applying this classifier alone to the training data set yields a very poor test set error rate of 45.8%, compared to 50% for

# Boosting



## Boosting Fits an Additive Model

- Boosting is a way of fitting an additive expansion in a set of elementary basis functions. Here the basis functions are the individual classifiers  $G_m(x)$ . More generally, basis function expansions take the form

$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m)$$

where  $\beta_m, m = 1, 2, \dots, M$  are the expansion coefficients, and  $b(x; \gamma) \in R$  are usually simple functions of the multivariate argument  $x$ , characterized by a set of parameters  $\gamma$ .

# Boosting Fits an Additive Model

Typically these models are fit by minimizing a loss function averaged over the training data, such as the squared-error or a likelihood-based loss function,

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^N L \left( y_i, \sum_{m=1}^M \beta_m b(x_i; \gamma_m) \right). \quad (10.4)$$

For many loss functions  $L(y, f(x))$  and/or basis functions  $b(x; \gamma)$ , this requires computationally intensive numerical optimization techniques. However, a simple alternative often can be found when it is feasible to rapidly solve the subproblem of fitting just a single basis function,

$$\min_{\beta, \gamma} \sum_{i=1}^N L(y_i, \beta b(x_i; \gamma)). \quad (10.5)$$

# Boosting Fits an Additive Model

---

**Algorithm 10.2** *Forward Stagewise Additive Modeling.*

---

1. Initialize  $f_0(x) = 0$ .
2. For  $m = 1$  to  $M$ :
  - (a) Compute

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

- (b) Set  $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$ .
-

# Forward Stagewise Additive Modeling

- For squared-error loss

$$L(y, f(x)) = (y - f(x))^2$$

on has

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2, \end{aligned}$$

- Thus, for squared-error loss, the term  $\beta_m b(x; \gamma_m)$  that best fits the current residuals is added to the expansion at each step.

# Exponential Loss and AdaBoost

- We now show that AdaBoost.M1 is equivalent to forward stagewise additive modeling using the loss function

$$L(y, f(x)) = \exp(-yf(x))$$

For AdaBoost the basis functions are the individual classifiers  $G_m(x) \in \{-1, 1\}$ . Using the exponential loss function, one must solve

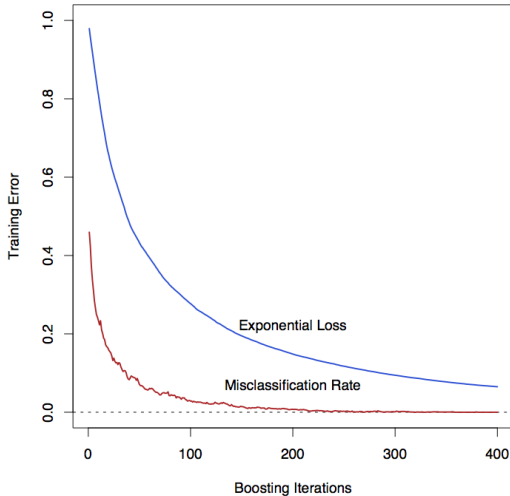
$$(\beta_m, G_m) = \arg \min_{\beta, G} \sum_{i=1}^N \exp[-y_i(f_{m-1}(x_i) + \beta G(x_i))]$$

for the classifier  $G_m$  and corresponding coefficient  $\beta_m$  to be added at each step. This can be expressed as

$$(\beta_m, G_m) = \arg \min_{\beta, G} \sum_{i=1}^N w_i^{(m)} \exp(-\beta y_i G(x_i)) \quad (10.9)$$

with  $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$ . Since each  $w_i^{(m)}$  depends neither on  $\beta$  nor  $G(x)$ , it can be regarded as a weight that is applied to each observation. This weight depends on  $f_{m-1}(x_i)$ , and so the individual weight values change with each iteration  $m$ .

# Boosting



**FIGURE 10.3.** Simulated data, boosting with stumps: misclassification error rate on the training set, and average exponential loss:  $(1/N) \sum_{i=1}^N \exp(-y_i f(x_i))$ . After about 250 iterations, the misclassification error is zero, while the exponential loss continues to decrease.

# Why Exponential Loss

The AdaBoost.M1 algorithm was originally motivated from a very different perspective than presented in the previous section. Its equivalence to forward stagewise additive modeling based on exponential loss was only discovered five years after its inception. By studying the properties of the exponential loss criterion, one can gain insight into the procedure and discover ways it might be improved.

The principal attraction of exponential loss in the context of additive modeling is computational; it leads to the simple modular reweighting AdaBoost algorithm. However, it is of interest to inquire about its statistical properties. What does it estimate and how well is it being estimated? The first question is answered by seeking its population minimizer.

It is easy to show (Friedman et al., 2000) that

$$f^*(x) = \arg \min_{f(x)} E_{Y|x}(e^{-Yf(x)}) = \frac{1}{2} \log \frac{\Pr(Y = 1|x)}{\Pr(Y = -1|x)}, \quad (10.16)$$

# Why Exponential Loss

or equivalently

$$\Pr(Y = 1|x) = \frac{1}{1 + e^{-2f^*(x)}}.$$

Thus, the additive expansion produced by AdaBoost is estimating one-half the log-odds of  $P(Y = 1|x)$ . This justifies using its sign as the classification rule in (10.1).

Another loss criterion with the same population minimizer is the binomial negative log-likelihood or *deviance* (also known as cross-entropy), interpreting  $f$  as the logit transform. Let

$$p(x) = \Pr(Y = 1 | x) = \frac{e^{f(x)}}{e^{-f(x)} + e^{f(x)}} = \frac{1}{1 + e^{-2f(x)}} \quad (10.17)$$

and define  $Y' = (Y + 1)/2 \in \{0, 1\}$ . Then the binomial log-likelihood loss function is

$$l(Y, p(x)) = Y' \log p(x) + (1 - Y') \log(1 - p(x)),$$

# Why Exponential Loss

or equivalently the deviance is

$$-l(Y, f(x)) = \log \left( 1 + e^{-2Yf(x)} \right). \quad (10.18)$$

Since the population maximizer of log-likelihood is at the true probabilities  $p(x) = \Pr(Y = 1 | x)$ , we see from (10.17) that the population minimizers of the deviance  $E_{Y|x}[-l(Y, f(x))]$  and  $E_{Y|x}[e^{-Yf(x)}]$  are the same. Thus, using either criterion leads to the same solution at the population level. Note that  $e^{-Yf}$  itself is not a proper log-likelihood, since it is not the logarithm of any probability mass function for a binary random variable  $Y \in \{-1, 1\}$ .

# Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown *sequentially*: each tree is grown using information from previously grown trees.

## Boosting algorithm for regression trees

1. Set  $\hat{f}(x) = 0$  and  $r_i = y_i$  for all  $i$  in the training set.
2. For  $b = 1, 2, \dots, B$ , repeat:
  - 2.1 Fit a tree  $\hat{f}^b$  with  $d$  splits ( $d + 1$  terminal nodes) to the training data  $(X, r)$ .
  - 2.2 Update  $\hat{f}$  by adding in a shrunk version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

- 2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x).$$

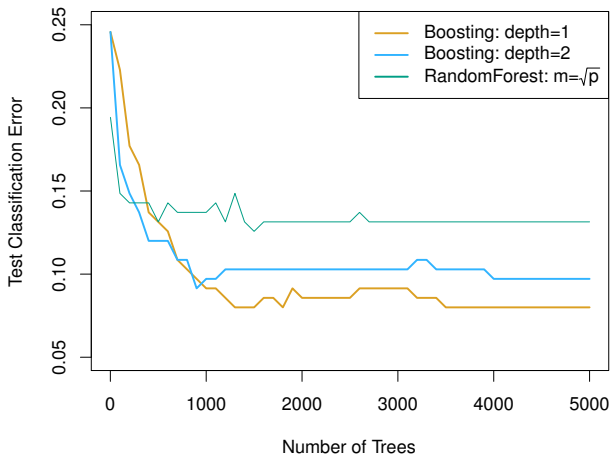
## What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to *fitting the data hard* and potentially overfitting, the boosting approach instead *learns slowly*.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter  $d$  in the algorithm.
- By fitting small trees to the residuals, we slowly improve  $\hat{f}$  in areas where it does not perform well. The shrinkage parameter  $\lambda$  slows the process down even further, allowing more and different shaped trees to attack the residuals.

## Boosting for classification

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here, nor do we in the text book.
- Students can learn about the details in *Elements of Statistical Learning, chapter 10*.
- The R package `gbm` (gradient boosted models) handles a variety of regression and classification problems.

## Gene expression data continued



## Details of previous figure

- Results from performing boosting and random forests on the fifteen-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%.

## Tuning parameters for boosting

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  $B$ .

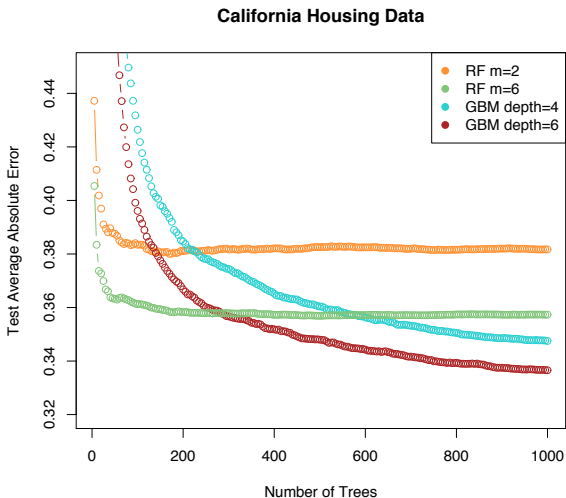
## Tuning parameters for boosting

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*  $\lambda$ , a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small  $\lambda$  can require using a very large value of  $B$  in order to achieve good performance.

## Tuning parameters for boosting

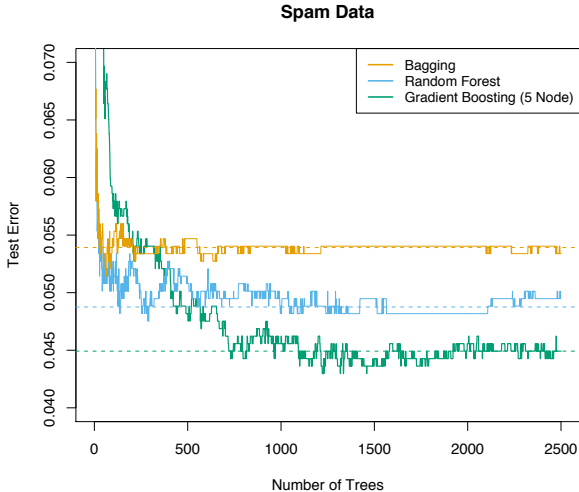
1. The *number of trees*  $B$ . Unlike bagging and random forests, boosting can overfit if  $B$  is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select  $B$ .
2. The *shrinkage parameter*  $\lambda$ , a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small  $\lambda$  can require using a very large value of  $B$  in order to achieve good performance.
3. The *number of splits*  $d$  in each tree, which controls the complexity of the boosted ensemble. Often  $d = 1$  works well, in which case each tree is a *stump*, consisting of a single split and resulting in an additive model. More generally  $d$  is the *interaction depth*, and controls the interaction order of the boosted model, since  $d$  splits can involve at most  $d$  variables.

## Another regression example



from *Elements of Statistical Learning*, chapter 15.

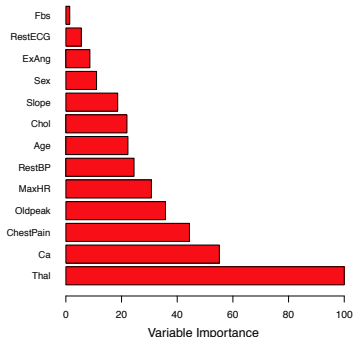
# Another classification example



from *Elements of Statistical Learning, chapter 15.*

## Variable importance measure

- For bagged/RF regression trees, we 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.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all  $B$  trees.



Variable importance plot  
for the **Heart** data

## Summary

- Decision trees are simple and interpretable models for regression and classification
- However they are often not competitive with other methods in terms of prediction accuracy
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods— random forests and boosting—are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.