Lecture 08. Tree-Based Methods (Chapter 8 and Section 5.2)

Ping Yu

HKU Business School
The University of Hong Kong

Introduction

- In Lecture 7, we discuss how to estimate $f(X) = E[Y|X]$ nonparametrically using polynomials, steps functions, splines or local regression when $X \in \mathbb{R}^1$; when $X \in \mathbb{R}^p$, we assume either $f(X) = \beta_0(X_p) + \beta_1(X_p)X_1 + \cdots + \beta_{p-1}(X_p)X_{p-1}$ or $f(X) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p)$, i.e., the nonparametric dimension is 1.
- Here we describe some general nonparametric methods – tree-based methods for regression and classification.
- Depending on the predicted outcome is discrete or continuous, it is called classification tree and regression tree, respectively, and combined as so-called Classification And Regression Tree (CART).
- These involve stratifying or segmenting the predictor space into a number of simple regions, and using the mean or mode response in the training region to which the test point belongs for prediction.
  - Different from step basis functions in Lecture 7, the stratifying regions are adaptively determined rather than predetermined.
- 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, such as those in Lectures 4 and 7, in terms of prediction accuracy.
- Hence we also discuss bagging, random forests (RF), boosting, and Bayesian additive regression trees (BART).[1]
- 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 in interpretation.
- These are examples of the ensemble method, which combines many simple "building block" models or weak learners to obtain a single more powerful model.

[1](**) These are general approaches which can be applied to many other statistical learning methods for regression or classification. We introduce them here because they are particularly useful and frequently used in the context of decision trees.

The Basics of Decision Trees (Section 8.1)

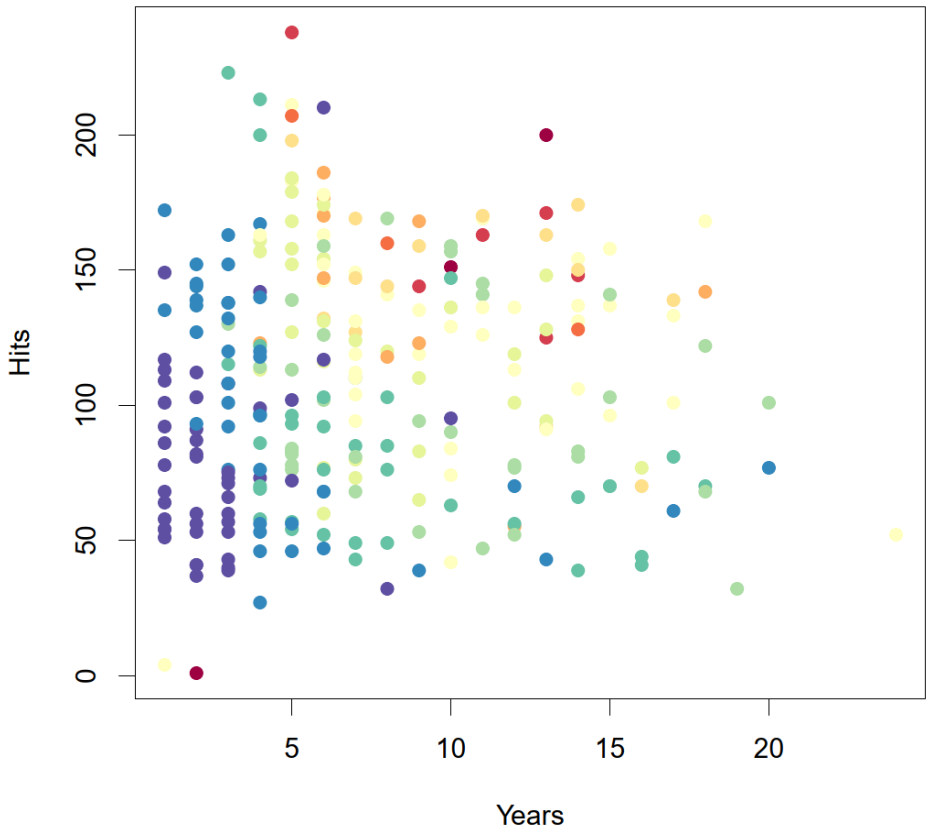
History of CART, Bagging and RF

Leo Breiman (1928-2005, UC-Berkeley)

- Breiman, L., J.H. Friedman, R.A. Olshen, and C.J. Stone, 1984, *Classification and Regression Trees*, Boca Raton, Fla.: Chapman & Hall/CRC.
- Breiman, L., 1996, Bagging Predictors, *Machine Learning*, 24, 123-140.
- Breiman, L., 2001, Random Forests, *Machine Learning*, 45, 5-32.

Regression Trees

- Using the Hitters data set, we try to predict a baseball player's Salary (in log thousands) based on Years and Hits (i.e., tenure and performance).



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

Decision Tree for These Data

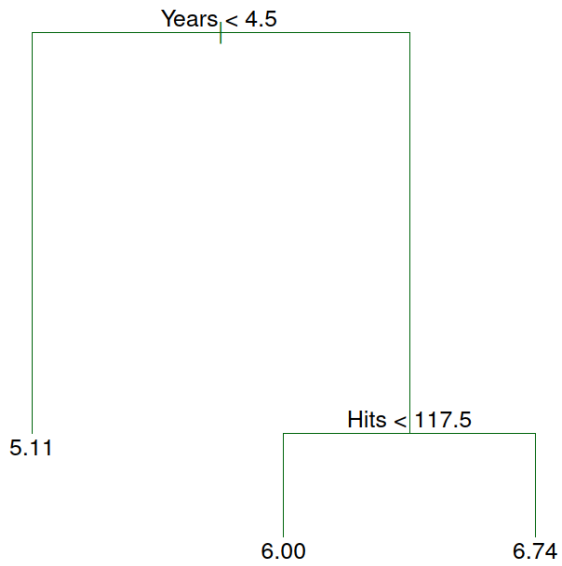


FIGURE 8.1. *For the* `Hitters` *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form* $X_j < t_k$*) indicates 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.*

- Years and Hits are integers, so the splitting points are the midpoints between two adjacent values.

Results

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

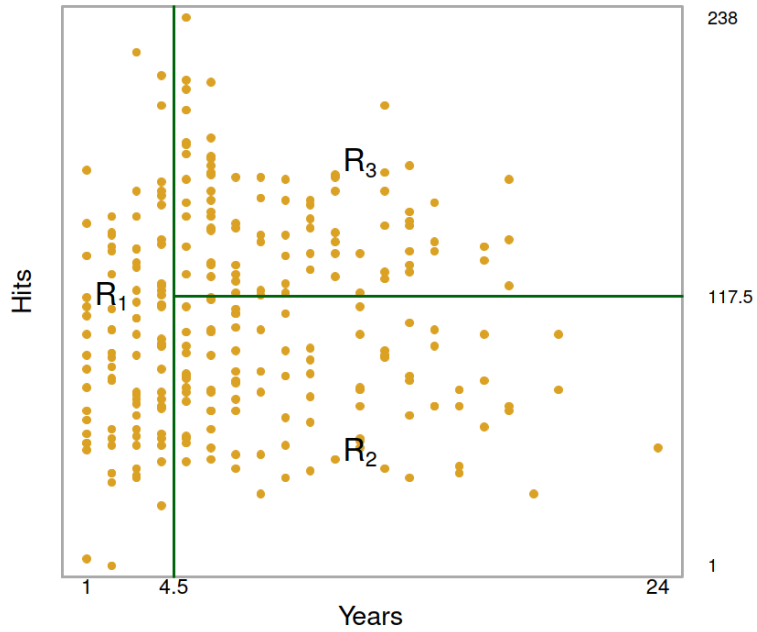


FIGURE 8.2. *The three-region partition for the* `Hitters` *data set from the regression tree illustrated in Figure 8.1.*

- Think of the differences from the step functions: stratified regions, interactions, and parameter restrictions. [Exercise]

Terminology for Trees

- In keeping with the tree analogy, the regions R_1, R_2, and R_3 are known as terminal nodes, or leaves.
- 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 (or interior) nodes.
- In the hitters tree, the two internal nodes are indicated by the text Years$<$ 4.5 and Hits$<$ 117.5.
- If the tree has only two terminal nodes, i.e., split only once, it is called a stump or best-first induced binary tree without pruning.
- The segments of the trees that connect the nodes (either terminal or internal) are called branches.

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 $X_1, X_2, \cdots, X_p$ – into J distinct and non-overlapping regions, $R_1, R_2, \cdots, R_J$.
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.

- 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 $R_1, R_2, \cdots, R_J$ that minimize the RSS, given by

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

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

Continued

- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $R_1(j,s) := \{X|X_j < s\}$ and $R_2(j,s) := \{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
  - Mathematically, we are minimizing

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

  with respect to both j and s.
- 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.

- How to predict the response for a given test observation? Use the mean of the training observations in the region to which that test observation belongs.

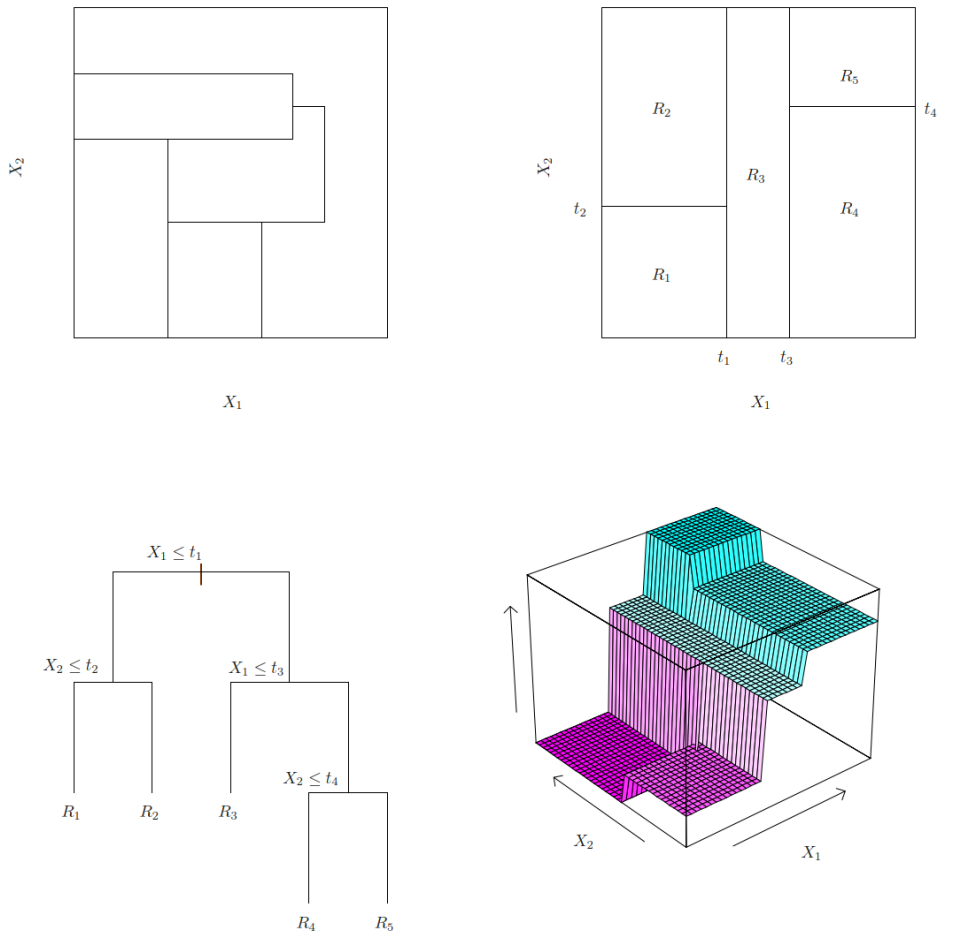


FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A perspective plot of the prediction surface corresponding to that tree.*

- The partitioning scheme in regression trees is obtained through nested parallel axis splitting.

(**) More Details

- The tree splitting rules are invariant to monotone transformations of the X components, so there is no need to standardize X_j's as in ridge regression.
- For binary or ordinal variables, the cut-points can be defined by the collection of all possible values. Unordered categorical variables with q levels are generally expanded into q binary indicators of these levels, or consider all possible category subsets.
  - If a binary predictor were used in a splitting rule, then it would no longer be available for splitting rules at nodes below it.

Tree Pruning

- 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 $R_1, R_2, \cdots, R_J$) 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-signed: 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.
- 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 is used to do this. Rather than considering every possible subtree (too many such subtrees!), we consider a sequence of trees indexed by a nonnegative tuning parameter α.

Continued

- For each value of α there corresponds a subtree $T_\alpha \subset T_0$ 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, where $|T|$ indicates the number of terminal nodes of the tree T.
  - How? Use weakest link pruning: successively collapse the internal (non-terminal) node that produces the smallest per-node increase in $\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2$, and continue until produce the single-node (root) tree; this sequence of subtrees must contain T_α.
- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
  - As α increases from 0, we prune T_0 in a nested and predictable fashion.
- We select an optimal value $\hat{\alpha}$ using cross-validation (CV).
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$ [see Algorithm 8.1].
- Note that the order of pruning the tree is not the reverse order of building the tree.

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

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

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

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

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

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

Baseball Example Continued

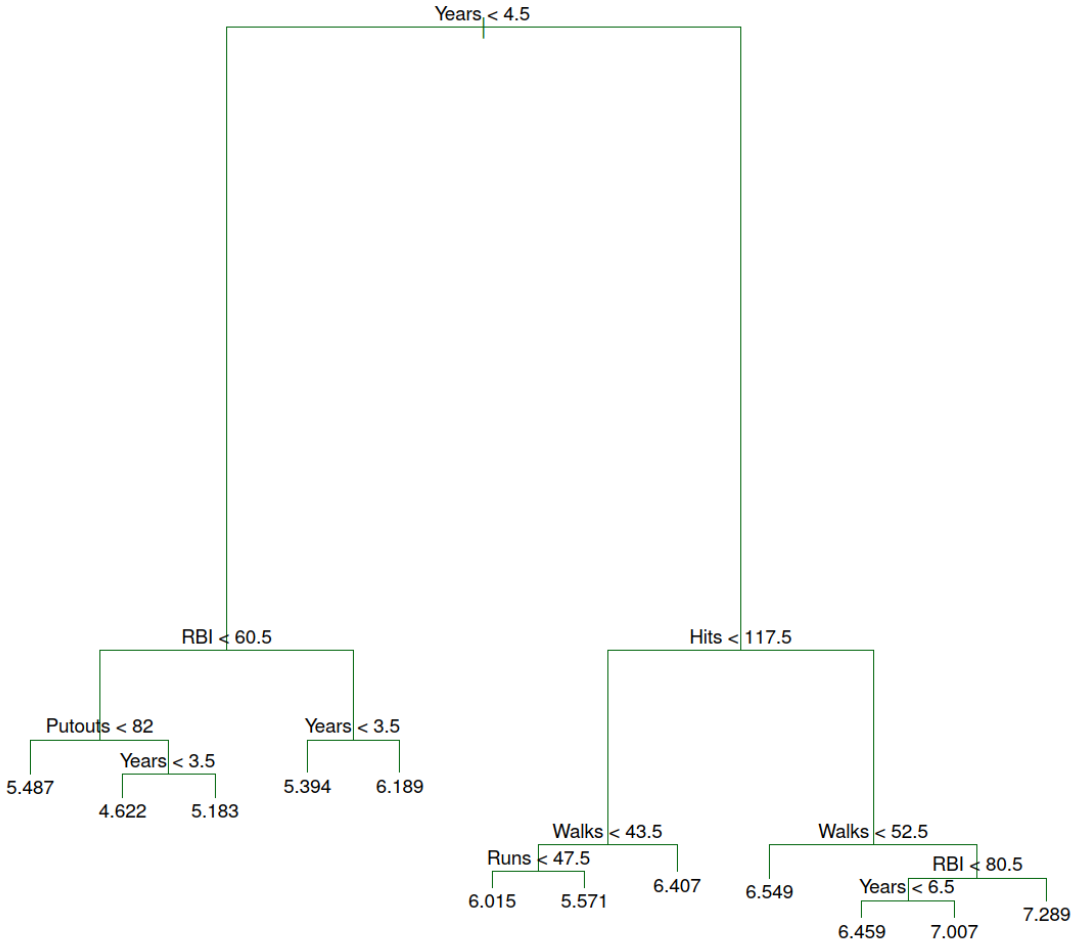


FIGURE 8.4. *Regression tree analysis for the* `Hitters` *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*

- $n_{\text{train}} = 132$, and $p = 9$.

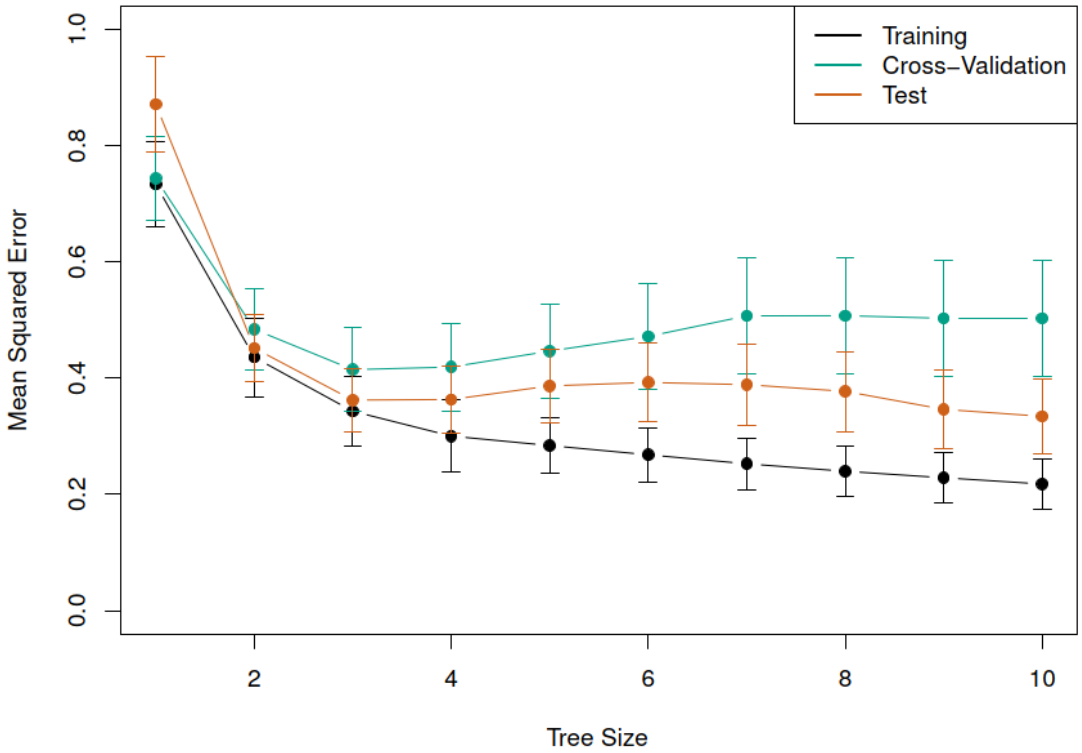


FIGURE 8.5. *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

- $n_{\text{test}} = 131$, and six-fold CV; x-axis is $|T|$ rather than α because the one-to-one correspondence between $|T|$ and α in the original tree.

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, which is simply the fraction of the training observations in that region that do not belong to the most commonly occurring class (i.e., the mode of y_i in that region).
- Define $\hat{p}_{mk} = \frac{1}{n_m} \sum_{x_i \in R_m} I(y_i = k)$ as the proportion of training observations in the mth region that are from the kth class, where $n_m = \sum_{i=1}^{n} I(x_i \in R_m)$, and $k(m) = \arg\max_k \hat{p}_{mk}$. Then

$$E = \frac{1}{n_m} \sum_{x_i \in R_m} I(y_i \neq k(m)) = 1 - \max_k (\hat{p}_{mk}).$$

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

Gini Index and Entropy

- The Gini index is defined as

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

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.
- The Gini index is thus referred to as a measure of node purity – a small value indicates that a node contains predominantly observations from a single class.

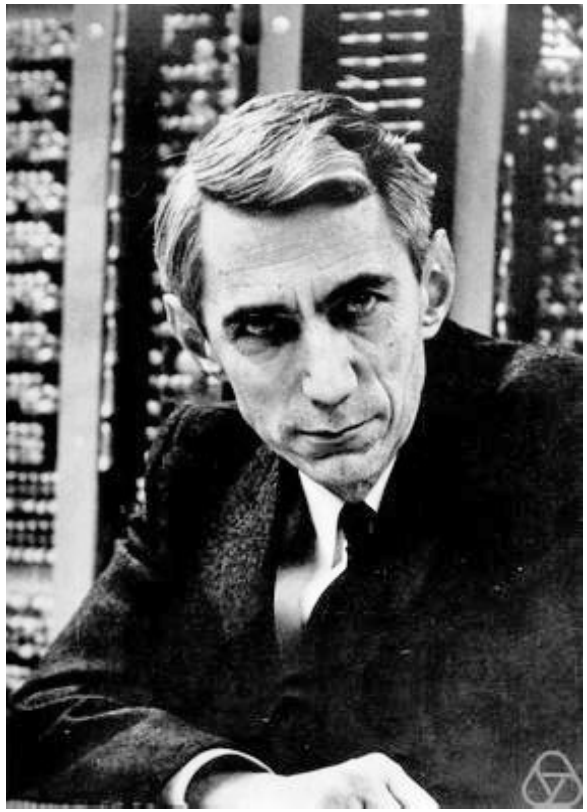
- An alternative to the Gini index is entropy :

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

where the base for log can be 2 (measured in bits), e (nats) or 10 (dits).
- (*) $\log \frac{1}{\hat{p}_{mk}}$ measures the surprise of outcome k, e.g., if $\hat{p}_{mk} = 1$, then $\log \frac{1}{\hat{p}_{mk}} = 0$ – not surprising at all, and if $\hat{p}_{mk} \to 0$, then $\log \frac{1}{\hat{p}_{mk}} \to \infty$ – very surprising. So entropy is the average level of "surprise" ("information"[2] or "uncertainty") inherent to the system's possible outcomes. For more discussions on entropy, see Appendix A.

[2] "information" and "surprise" are the same: a certain prediction when $\hat{p}_{mk}$ is small is very informative.

History of Entropy



Claudem E. Shannon (1916-2001, MIT), the father of information theory

- Entropy in the information theory was introduced by Claude Shannonn in his 1948 landmark paper "*A Mathematical Theory of Communication*", and measures the disorder of a random system (in our case, a random variable or a dataset); D with base 2 in log gives the average (genuine) information in the unit of bits of a random system.

Continued

- It turns out that the Gini index and entropy are very similar numerically. [figure here for $K = 2$]
- Gini index and entropy are also more sensitive to variations in the two child-node probabilities $\left\{\hat{p}_{m_L k(m_L)}, \hat{p}_{m_R k(m_R)}\right\}$, so are recommended to grow the tree. [see the next[2] slide]
- They are differentiable, so more amenable to numerical optimization than misclassification error.
- Parallel to regression tree, where

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 = \sum_{m=1}^{|T|} n_m \left[\frac{1}{n_m} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 \right],$$

we should weight $Q_{m_L}(T)$ and $Q_{m_R}(T)$ by n_{m_L} and n_{m_R} when splitting node m, i.e., we try to find the splitting that minimizes

$$n_{m_L} Q_{m_L}(T) + n_{m_R} Q_{m_R}(T),$$

where $Q_{m_L}(T)$ and $Q_{m_R}(T)$ are either of the three measures of impurity in the left and right branches.
- Equivalently, we use weighted $Q_{m_L}(T)$ and $Q_{m_R}(T)$, $w_{m_L} Q_{m_L}(T) + w_{m_R} Q_{m_R}(T)$, where $w_{m_L} = n_{m_L}/n_m$, and $w_{m_R} = 1 - w_{m_L}$.

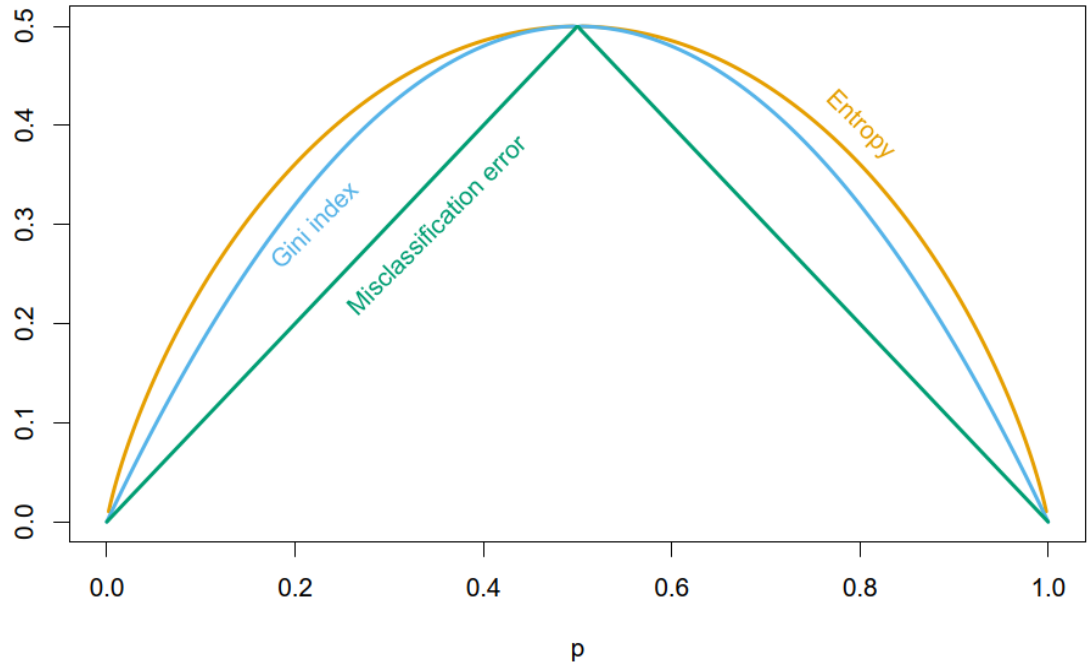
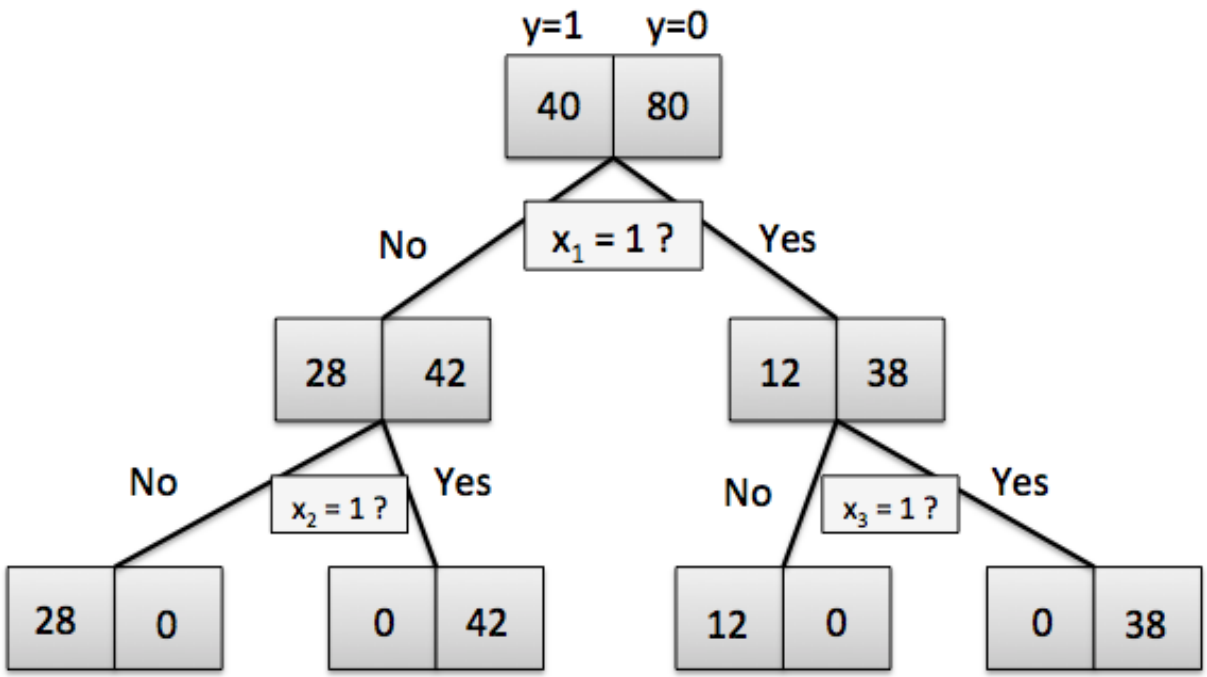


Figure: Node impurity measures for two-class classification, as a function of the proportion p in class 1. Entropy has been scaled to pass through $(0.5, 0.5)$.

- Since $p_2 = 1 - p_1$, $G = p_1(1-p_1) + p_2(1-p_2) = 2p_1(1-p_1)$ is symmetric about $p_1 = 0.5$, with $G = 0.5$ when $p_1 = 0.5$ and $G = 0$ when $p_1 = 0$.
- $D = -p_1 \log p_1 - p_2 \log p_2$ is also symmetric with $D = \log_2 2 = 1$ when $p_1 = 0.5$ and $D = 0$ when $p_1 = 0$ (by the convention $0 \cdot \log 0 = 0$); i.e., when $p_1 = 0.5$, the system has the maximal disorder. This also indicates that entropy can be used to quantify the similarity or difference in the probabilities of the outcomes.

(**) Why Growing Classification Trees via Gini Index/Entropy Instead of the Misclassification Error?

- We take entropy for illustration since the analysis for Gini index is similar.
- In the following example, x_1, x_2 and x_3 are three binary features used for growing the tree.
- The questions is whether the first splitting would be conducted (implicitly, x_1 is the best splitter under either criterion); if YES, then the ensued splittings would generate pure terminal nodes, the dream of any classification.



(**) Continued

- If misclassification error is used, then

$$\Delta E = \frac{40}{120} - \left(\frac{70}{120} \times \frac{28}{70} + \frac{50}{120} \times \frac{12}{50} \right) = 0,$$

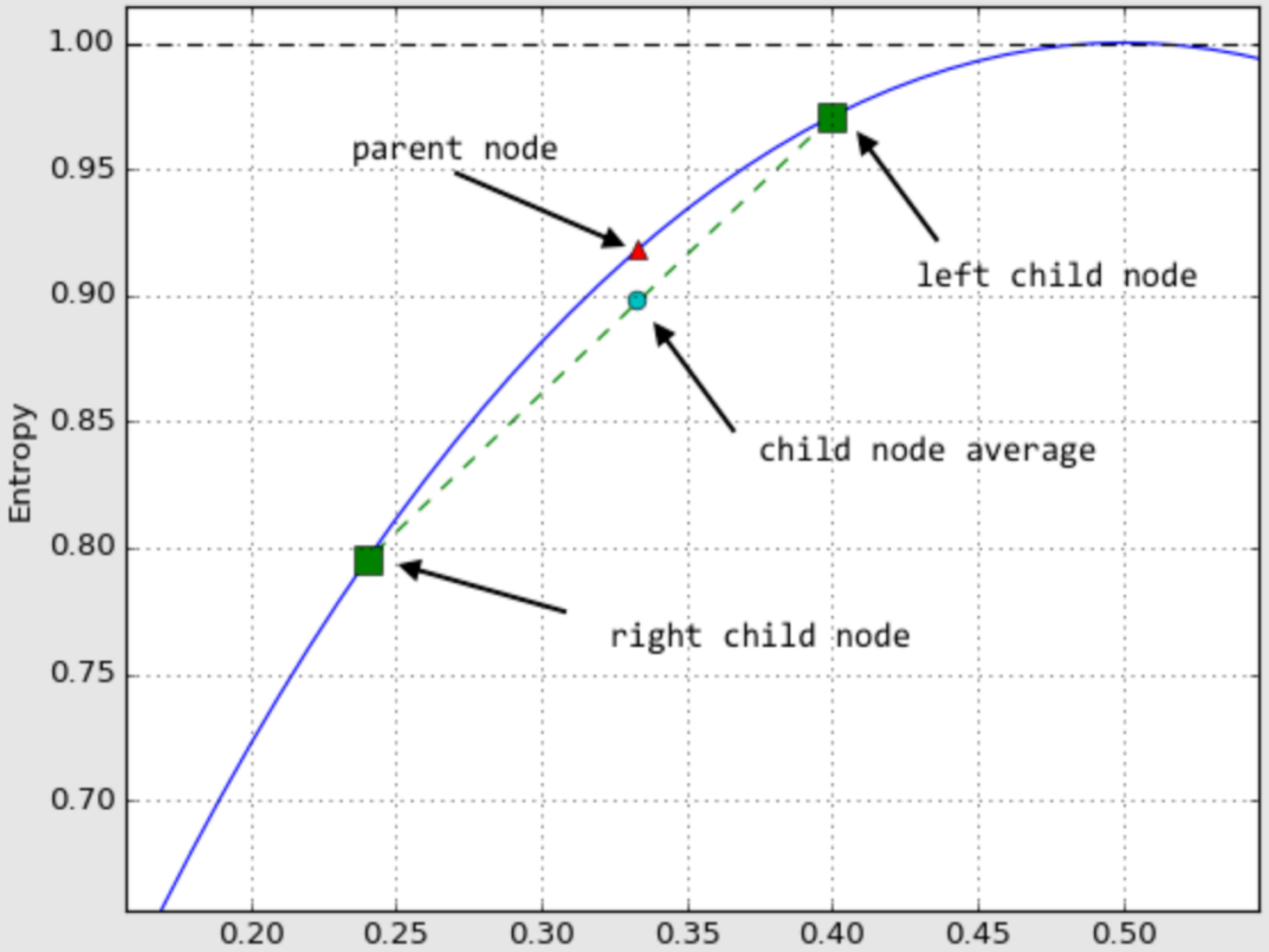
  so the first splitting would not be conducted.
- If entropy is used, then

$$\Delta D = \left(\frac{40}{120} \log \frac{120}{40} + \frac{80}{120} \log \frac{120}{80} \right) - \left[\frac{70}{120} \times \left(\frac{28}{70} \log \frac{70}{28} + \frac{42}{70} \log \frac{70}{42} \right) + \frac{50}{125} \times \left(\frac{12}{50} \log \frac{50}{12} + \frac{38}{50} \log \frac{50}{38} \right) \right] = 0.02 > 0,$$

  so the first splitting would be conducted.
- Why can this happen? Entropy is concave in p_1 while misclassification error is linear in p_1 when $p_1 \leq 0.5$.
- In this example, $E_m = w_{m_L} E_{m_L} + w_{m_R} E_{m_R}$, while $D_m = D(E_m) > w_{m_L} D(E_{m_L}) + w_{m_R} D(E_{m_R}) = w_{m_L} D_{m_L} + w_{m_R} D_{m_R}$, where

$$D(E_m) = -E_m \log E_m - (1 - E_m) \log (1 - E_m),$$

$E_{m_L} = \frac{28}{70}$, $E_{m_R} = \frac{12}{50}$, $w_{m_L} = \frac{70}{120}$, and $w_{m_R} = \frac{50}{120}$. [figure here]



- x-axis is the misspecification error.

Pruning the Tree

- Typically, the misclassification error rate rather than Gini index or entropy is used to prune the tree if prediction accuracy of the final pruned tree is the goal, i.e., use

$$\sum_{m=1}^{|T|} n_m E_m(T) + \alpha |T|$$

to prune a tree.

- Note that $\sum_{m=1}^{|T|} n_m E_m(T)$ is the total number of missclassifications in all terminal nodes.

- In other words, we use different criteria to grow a tree and prune a tree!

[Example] Heart Data

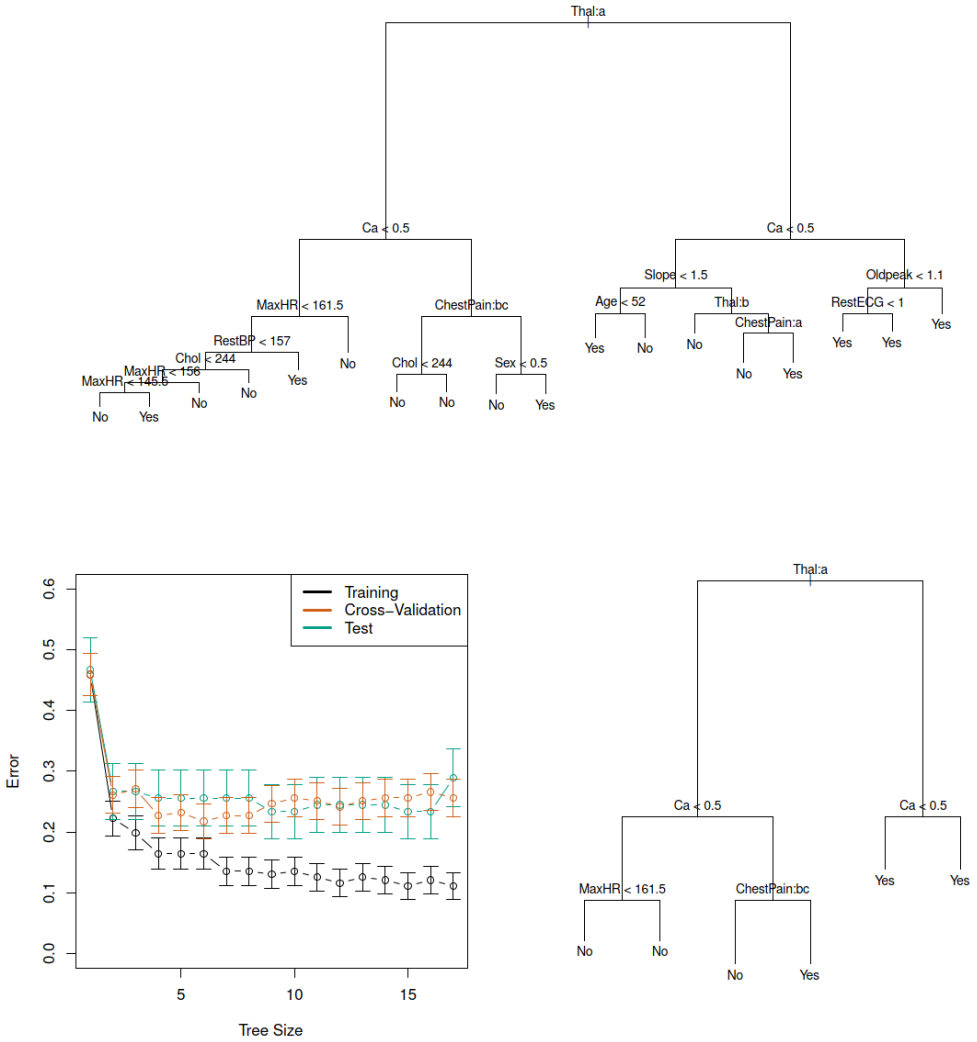


FIGURE 8.6. `Heart` *data.* Top: *The unpruned tree.* Bottom Left: *Cross-validation error, training, and test error, for different sizes of the pruned tree.* Bottom Right: *The pruned tree corresponding to the minimal cross-validation error.*

Comments

- $n = 303$, and $p = 13$; the CV $|T|$ is 6.
- The two nodes in each splitting need not have different predictions.
  - The split is performed because it can increase node purity, which is important for prediction.
  - The misclassification error rate need not decrease in both nodes.
  - For example, 16 of 20 observations in one node is 1, and the right child node contains 9 1's, and the left contains 7 1's and 4 0's. $E_f = \frac{4}{20} < \frac{4}{11} = E_L$, but $20 \times \frac{16}{20} \times \frac{4}{20} > 11 \times \frac{7}{11} \times \frac{4}{11} + 9 \times 0$.[3]
- Pure nodes are not split further since the objective function is zero for any splitting.
- X_j can be qualitative, e.g., Sex, Thal and ChestPain. [see Figure 8.6]

[3] Here, we can see the importance of weighting $Q_{m_L}(T)$ and $Q_{m_R}(T)$ by n_{m_L} and n_{m_R}, respectively. Otherwise, $\frac{16}{20} \times \frac{4}{20} < \frac{7}{11} \times \frac{4}{11} + 0$. Essentially, we are weighting the impurity of each child-node by its probability $p_{mL} = \frac{n_{mL}}{n_m}$ and $p_{mR} = \frac{n_{mR}}{n_m}$; otherwise, we may split off a single data point (which is pure) to reduce the impurity of the parent node. Similarly, in regression tree, we are actually using the weighted variance as the impurity measure.

Trees Versus Linear Models

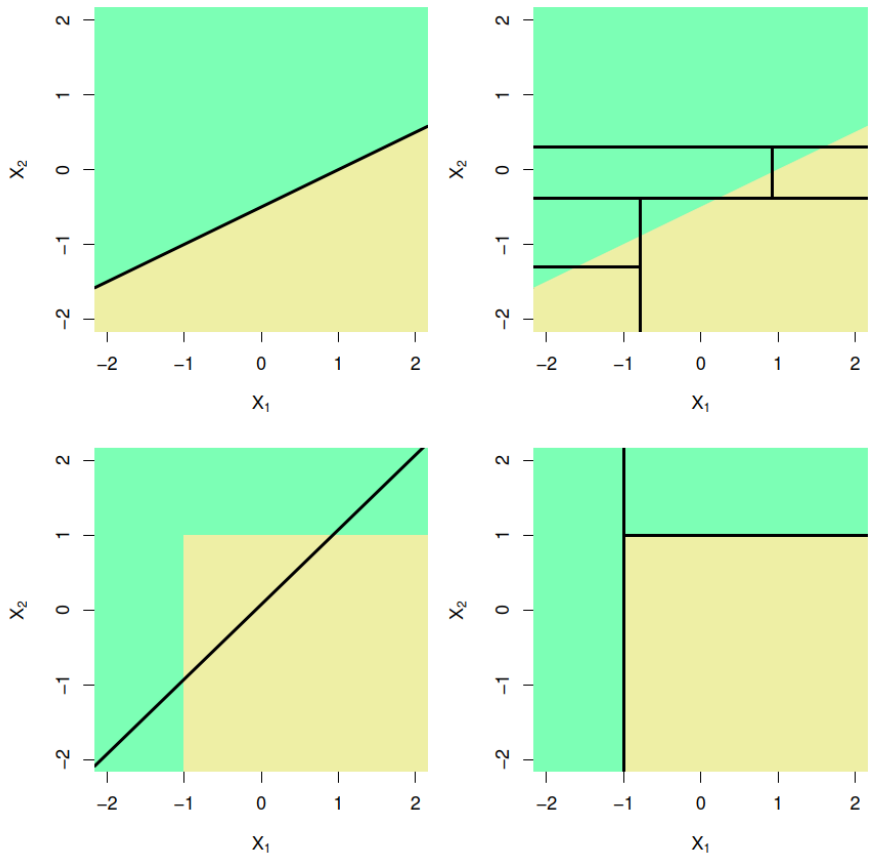


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: *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 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 lectures, e.g., "Middle-aged professionals who reside in Hong Kong have the highest churn rate" is a statement based on decision trees.
- 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 (i.e., have high variance) as some of the other regression and classification approaches seen in this course.
- 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.

- However, by aggregating many decision trees, the predictive performance of trees can be substantially improved.

Bagging, Random Forests, Boosting, and Bayesian Additive Regression Trees

(Section 8.2)

History of The Bootstrap

Bradley Efron (1938-, Stanford)[4]

- Efron, B., 1979, Bootstrap Methods: Another Look at the Jackknife, *Annals of Statistics*, 7, 1-26.

[4] He won almost all honors in Statistics, e.g., the International Prize in Statistics, one of the two highest honours in the field of Statistics (the other one is the COPPS Award, but Efron is too old to be eligible when the award started in 1981 since the candidates must be under age 41), and the National Medal of Science, the highest scientific honor by the United States.

The Bootstrap (Section 5.2)

- The bootstrap is a powerful tool to quantify the uncertainty of an estimator or a statistical learning method, especially when a measure of variability is difficult to obtain, e.g., it can provide an estimate of the standard error of a coefficient, or a confidence interval for that coefficient.
- Where does the name come from? The use of the term bootstrap derives from the phrase to pull oneself up by one's bootstraps, widely thought to be based on one story of the 18th century "*The Surprising Adventures of Baron Munchausen*" by Rudolph Erich Raspe:
  The Baron had fallen to the bottom of a deep lake. Just when it looked like all was lost, he thought to pick himself up by his own bootstraps. [figure here]
- In general, bootstrapping usually refers to a self-starting process that is supposed to continue or grow without external input.
- (**) In computer science, the term bootstrapping refers to language compilers that are able to be coded in the same language (e.g., a C compiler is now written in the C language). It also refers to the process of loading the basic software into the memory of a computer after power-on, the kernel will load the operating system which will then take care of loading other device drivers and software as needed.

A pair of boots with one bootstrap visible

Baron Munchausen pulls himself and his horse out of a swamp by his pigtail (not bootstraps)

A Simple Example

- Suppose we want to invest our money to two financial assets that yield returns X and Y, respectively. It can be shown that the fraction of our money in X, say α, that minimizes the total risk, $\text{Var}(\alpha X + (1-\alpha)Y)$ is

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}},$$

  but $\sigma_X^2 = \text{Var}(X)$, $\sigma_Y^2 = \text{Var}(Y)$, and $\sigma_{XY} = \text{Cov}(X, Y)$ are unknown.
- Given a sample of (X, Y), we can estimate α by

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}},$$

  but the uncertainty of $\hat{\alpha}$ is not easy (although not impossible) to quantify.

Continued

- If we could obtain N samples of (X, Y) and each sample contains n data points, (x_i, y_i), $i = 1, 2, \cdots, n$, then we can calculate $\hat{\alpha}$ N times to have $\hat{\alpha}_r$, $r = 1, 2, \cdots, N$. [see Figure 5.9 for $n = 100$, and the left panel of Figure 5.10 for the histogram of $\hat{\alpha}_r$ when $N = 1000$]
- Based on $\hat{\alpha}_r$, we have

$$\overline{\hat{\alpha}} = \frac{1}{N}\sum_{r=1}^{N}\hat{\alpha}_r \text{ and } \mathrm{SD}_N(\hat{\alpha}) = \sqrt{\frac{1}{N-1}\sum_{r=1}^{N}\left(\hat{\alpha}_r - \overline{\hat{\alpha}}\right)^2},$$

and we expect $\overline{\hat{\alpha}}$ is close to $E[\hat{\alpha}]$ (or even α) and $\mathrm{SD}_N(\hat{\alpha})$ is close to $\mathrm{sd}(\hat{\alpha})$ when N is large.
- This is indeed the case, e.g., $\overline{\hat{\alpha}} = 0.5996$ in the left panel of Figure 5.10, very close to the true $\alpha = 0.6$.

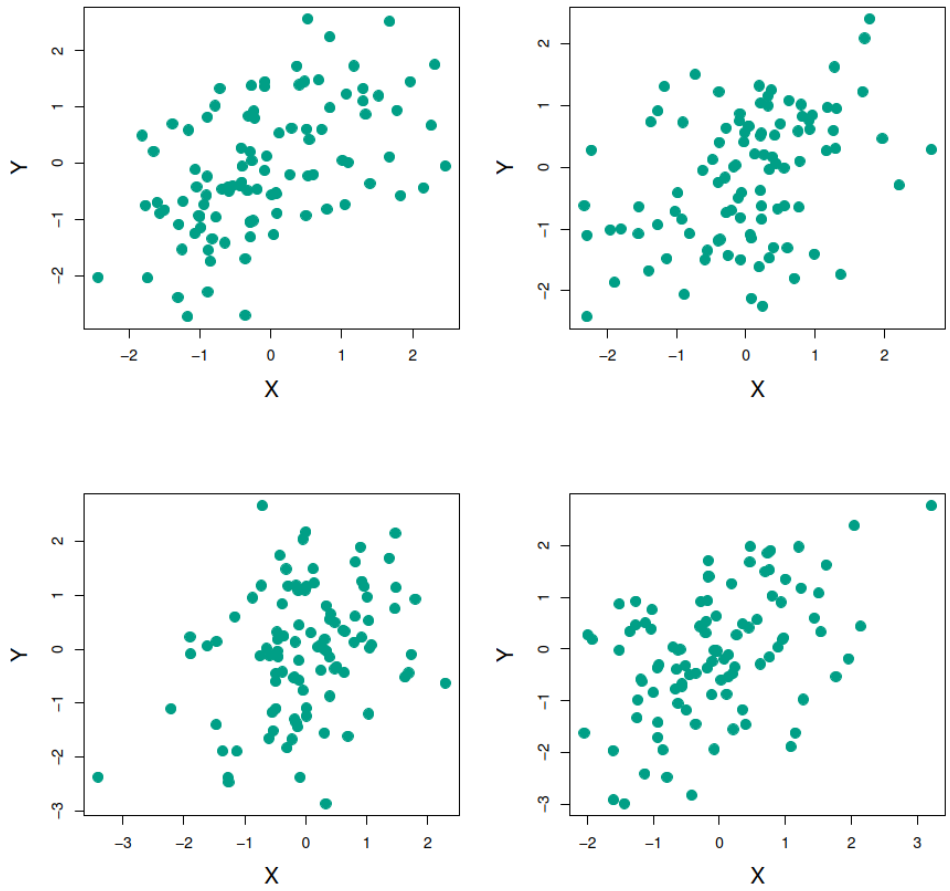


FIGURE 5.9. *Each panel displays* 100 *simulated returns for investments* X *and* Y. *From left to right and top to bottom, the resulting estimates for* α *are* 0.576, 0.532, 0.657, *and* 0.651.

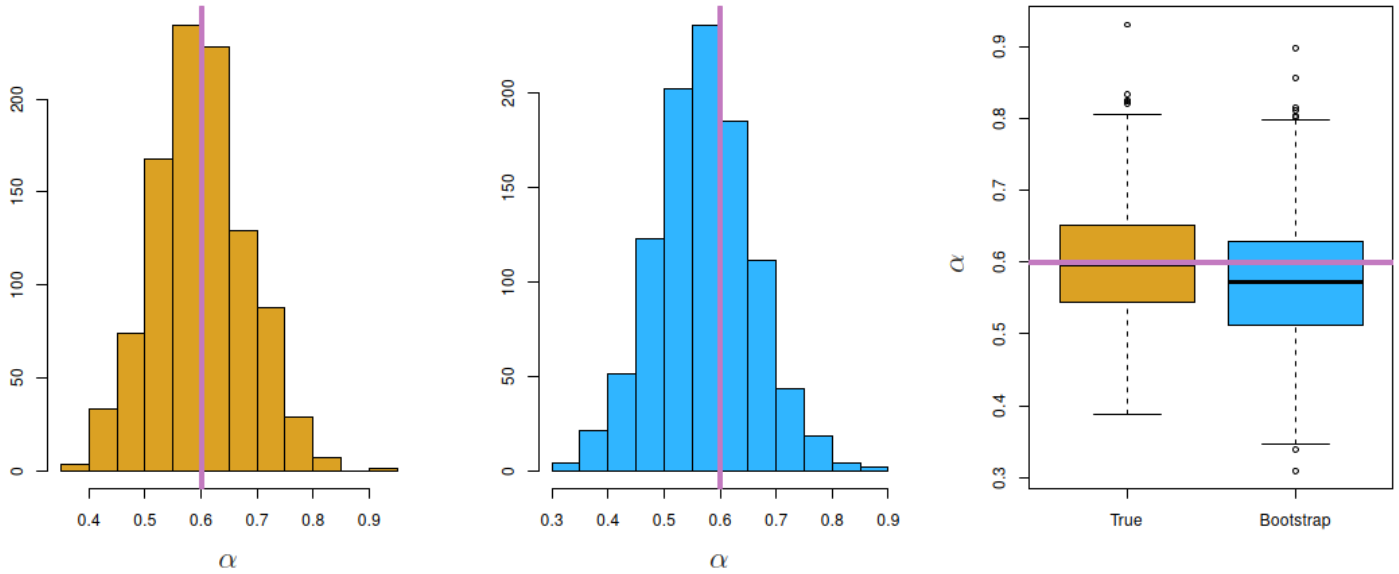


FIGURE 5.10. Left: *A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population.* Center: *A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set.* Right: *The estimates of α displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α.*

- $\sigma_X^2 = 1, \sigma_Y^2 = 1.25$, and $\sigma_{XY} = 0.5$, so the true $\alpha = 0.6$.

Now Back to the Real World

- But in practice, $N = 1$, i.e., we have only one sample.
- The bootstrap samples B distinct data sets (with sample size n) by repeatedly and randomly sampling observations from the original data set with replacement. [see Figure 5.11]
  - What happens if sampling without replacement?
  - Since with replacement, some observations may appear more than once in a given bootstrap data set and some not at all.
  - The pair (x_i, y_i) must be sampled as a whole with probability $\frac{1}{n}$.
  - This idea is to mimic sampling N independent samples from the population, but the population in the bootstrap world is the original data set.

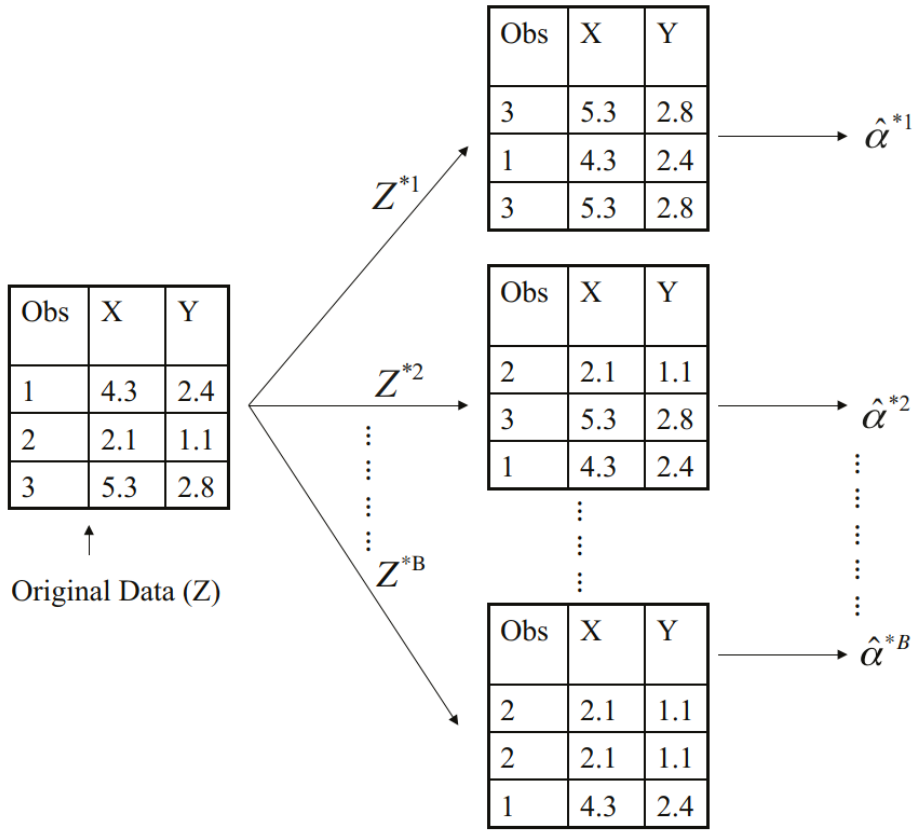


FIGURE 5.11. *A graphical illustration of the bootstrap approach on a small sample containing $n = 3$ observations. Each bootstrap data set contains n observations, sampled with replacement from the original data set. Each bootstrap data set is used to obtain an estimate of α.*

Continued

- The counterpart of $\text{SD}_N(\hat{\alpha})$ is

$$\text{SE}_B(\widehat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}\left(\hat{\alpha}^{*r} - \overline{\hat{\alpha}^*}\right)^2},$$

  where $\hat{\alpha}^{*r}$ is the α estimator based on the rth bootstrap data set, Z^{*r}, and

$$\overline{\hat{\alpha}^*} = \frac{1}{B}\sum_{r=1}^{B}\hat{\alpha}^{*r}.$$

- We expect that $\text{SE}_B(\widehat{\alpha})$ is close to $\text{sd}(\hat{\alpha})$ when B is large.
  - This is indeed the case, e.g., in the middle panel of Figure 5.10, $\text{SE}_B(\widehat{\alpha}) = 0.087 \approx 0.083 = \text{SD}_N(\hat{\alpha})$ with $B = N = 1000$.
  - (**) When B is large, $\overline{\hat{\alpha}^*}$ should be close to $\hat{\alpha}$ which is close to α when n is large.
  - Check also the right panel of Figure 5.10 for a comparison of the distributions of $\widehat{\alpha}$ and $\hat{\alpha}^*$.

(**) The Bootstrap in General

- In more complex data situations, figuring out the appropriate way to generate bootstrap samples can require some thought.
- For example, if the data is a time series, we can't simply sample the observations with replacement (why not?).
- We can instead create blocks of consecutive observations, and sample those with replacements. Then we paste together sampled blocks to obtain a bootstrap dataset.

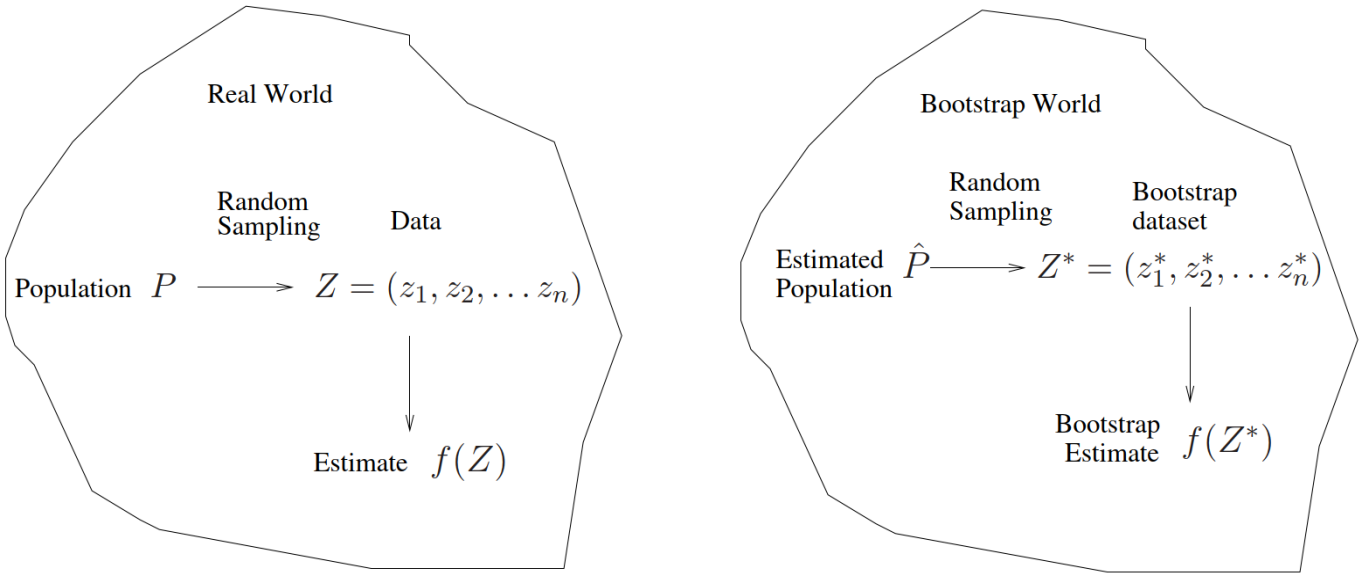


Figure: A general picture for the bootstrap

Bagging

- Recall that given a set of n independent observations $Z_1, \cdots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n. In other words, averaging a set of observations reduces variance.
- In the regression context, we could estimate $f(x)$ by

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

  based on B training sets, where $\hat{f}^b(x)$ is the regression tree estimate of $f(x)$ based on the bth training set.
- Of course, this is not practical because we generally do not have access to multiple training sets. 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 bth 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 Bootstrap aggregation, or bagging.

Bagging Classification Trees

- For each test observation, we record the class predicted by each of the B classification trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.
- In Figure 8.8, the bagging test error rate is slightly lower than that in a single tree for the Heart data.
  - Note that B is not a critical parameter; using a large B will not lead to overfitting.
  - In practice, use a B sufficiently large such that the error has settled down.

- We grow deep and unpruned trees in bagging such that individual trees have low bias but high variance.
- In other words, the main purpose of bagging is to reduce variance rather than reduce bias.

[Example] Heart Data

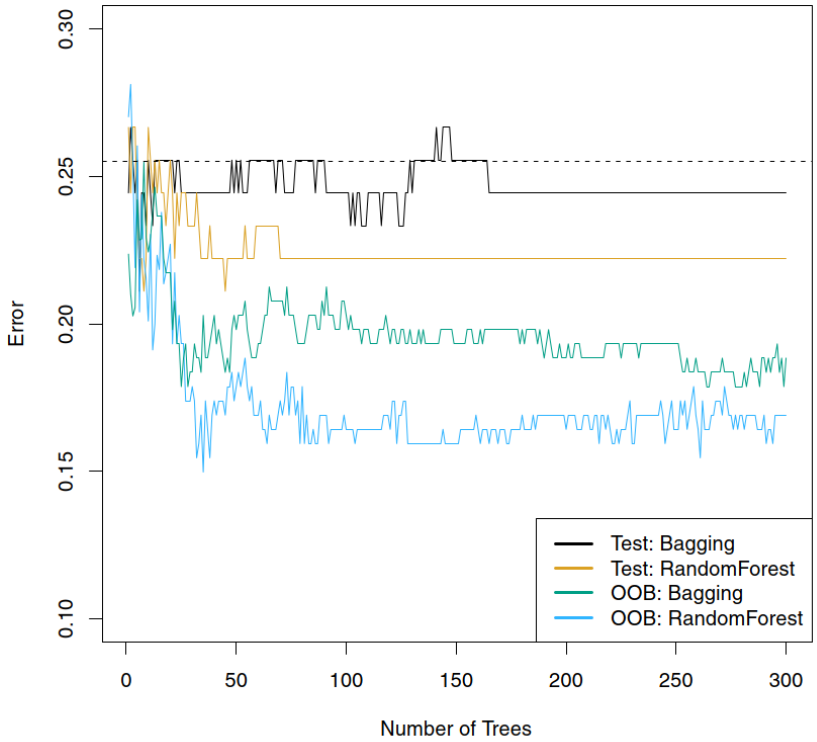


FIGURE 8.8. *Bagging and random forest results for the* `Heart` *data. The test error (black and orange) is shown as a function of* B*, the number of bootstrapped training sets used. Random forests were applied with* $m = \sqrt{p}$*. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is — by chance — 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, without the need to perform CV or the validation set approach.
- 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.
  - (*) $1-\left(1-\frac{1}{n}\right)^n \longrightarrow 1-e^{-1}=0.632$.
- 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. [figure here]
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the ith observation, which we average or take a majority vote to get a single OOB prediction $\hat{f}_{\text{OOB}}(x_i)$.

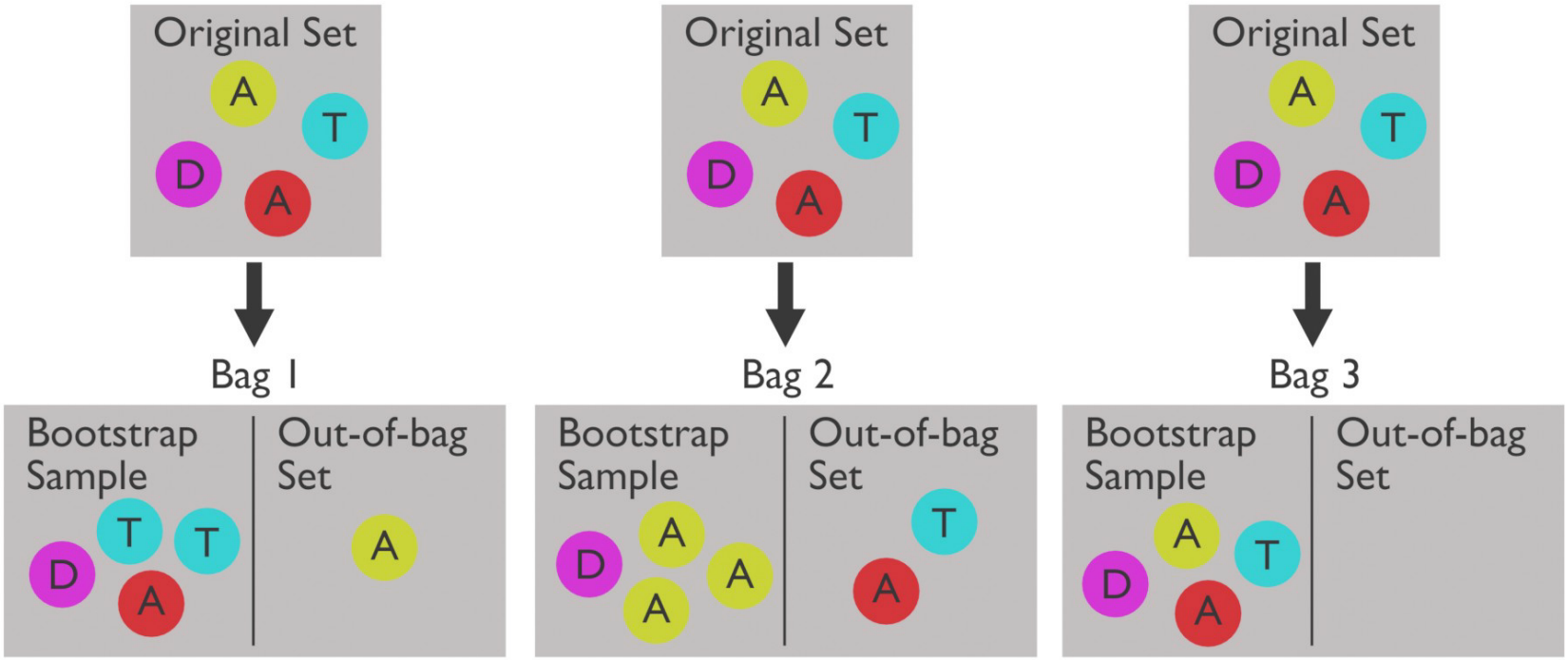


Figure: Out-of-Bag Samples

- One set, the bootstrap sample, is the data chosen to be "in-the-bag" by sampling with replacement. The out-of-bag set is all data not chosen in the sampling process.

Continued

- The OOB error estimate is

$$\text{err}_{\text{OOB}} = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{f}_{\text{OOB}}(x_i)\right),$$

where

$$L\left(y_i, \hat{f}_{\text{OOB}}(x_i)\right) = (y_i - \hat{f}_{\text{OOB}}(x_i))^2$$

in regression and

$$L\left(y_i, \hat{f}_{\text{OOB}}(x_i)\right) = I(y_i \neq \hat{f}_{\text{OOB}}(x_i))$$

in classification. [see Figure 8.8]
- When B is large enough, it is equivalent to LOOCV error; i.e., the number of trees can be chosen along the way without extra cost (for each B, we plant B trees rather than nB trees as in LOOCV).

Variable Importance Measures

- Bagging improves prediction accuracy at the expense of interpretability.
- But we can obtain an overall summary of the importance of each predictor using the RSS (for regression) or the Gini index (for classification).
  - For classification, deviance is also used, which is defined as -2 times the maximized log-likelihood,
$$-2\sum_m\sum_k n_{mk}\log\hat{p}_{mk}.$$
- Specifically, we record the total amount that the objective function is decreased due to splits over a given predictor (maybe multiple times, maybe none), averaged over all B trees, referred to as "relative importance".
- The variable importance plot for the Heart data is shown in Figure 8.9, where the most important variables include two qualitative variables Thal and ChestPain, and one quantitative variable Ca among the 13 predictors.
- (**) Another way to measure variable importance is referred to as "permutation importance" and is considerably slower than the relative importance: one at a time, each feature is shuffled and an OOB estimation of the prediction error is made on this 'shuffled' data set; intuitively, irrelevant features will not change the prediction error when altered in this way, opposite to the very relevant ones; the relative loss in performance between the 'original' and 'shuffled' data sets is therefore related to the relevance of the shuffled feature.

[Example] Heart Data

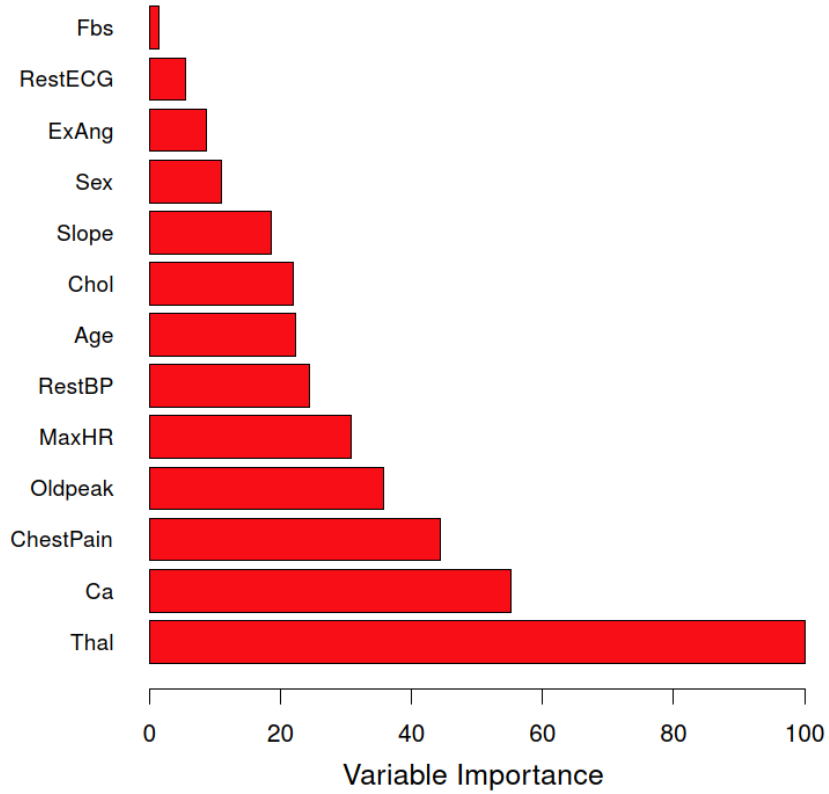


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

Random Forests

- Random forests (RF) 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.
  - If $Z_1 = \cdots = Z_n$, then $Var\left(\bar{Z}\right) = Var\left(Z_i\right)$!
- 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 sample 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 $m \approx \sqrt{p}$ (4 out of the 13 for the Heart data; see Figure 8.8 for the improvement over bagging where $m = p$).
- Why works? Think of the case with a strong predictor; give other predictors a chance (with $\frac{p-m}{p}$ probability the strong predictor not chosen, usually greater than $\frac{1}{2}$!).
  - This is typically helpful when we have a large number of correlated predictors.
- OOB error estimation and variable importance measures are similarly defined as in bagging; also, a large B will not overfit.

History of Boosting

Yoav Freund (1961-, UCSD)

Robert E. Schapire (1963-, Microsoft)[5]

- Freund, Y., and R.E. Schapire, 1997, A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, *Journal of Computer and System Sciences*, 55, 119-139.

[5]previoiusly at Princeton.

Boosting

- Different from bagging and RF, boosting repeatedly grow shallow trees, so its main purpose is bias reduction (or debiasing) rather than variance reduction.
- The trees in boosting are grown in an adaptive and sequential way to remove bias, and hence are not fit independently and not i.d. (identically distributed) as in bagging and RF.
  - Each tree is grown using information from previously grown trees – the residuals rather than y_i. [see Algorithm 8.2]
- Boosting has three tuning parameters, d, λ and B.
- d is typically small, e.g., $d = 1$ (a stump). In general, d is the interaction depth, e.g., $d = 2$, each tree involves at most two variables. Smaller trees aid in interpretability, e.g., $d = 1$ leads to an additive model. [Exeicise]
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ, typically, 0.01 or 0.001 (depending on the problem), slows the process down even further, allowing more and different shaped trees to attack the residuals.
  - In general, statistical learning approaches that learn slowly tend to perform well.
- Different from bagging and RF, boosting can overfit if B is too large.
  - B can be chosen by CV.
  - Trade-off between λ and B.

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

- Boosting is somewhat similar to PLS.

Boosting Classification Trees and Variable Importance

- The original boosting algorithm is AdaBoost (adaptive boosting) which is developed for the two-class classification problem and uses an exponential loss.
- Adaboost and Algorithm 8.2 are special cases of gradient boosting. [see Appendix B for more details on AdaBoost and Gradient Boosting]
- The **R** package gbm (gradient boosted models) handles a variety of regression and classification problems.

- **R** uses the "relative importance" to measure the variable importance in boosting: record how much a given metric (e.g. MSE) changes every time a given variable is used for splitting, get the average reductions across all base-learners for each variable, and then normalize the total importance as 100 (rather than the most important variable as 100 as in Figure 8.9).
  - "relative importance" is not "absolute importance", i.e., it is not the percentage of $Var(Y)$ explained.
  - (**) We can imagine the "permutation importance" can also be computed.
- **R** also reports partial dependence plots, which plot

$$\hat{f}(x_S) = \frac{1}{n}\sum_{i=1}^{n}\hat{f}(x_S, x_{ic})$$

against x_S, where $x = (x_S, x_C)$, x_S is the variable of interest, and x_C includes all other variables.

History of BART

Hugh A. Chipman (Acadia)

Edward I. George (UPenn)

Robert E. McCulloch (ASU)

- Chipman, H.A., E.I. George, and R. McCulloch, 2010, BART: Bayesian Additive Regression Trees, *Annals of Applied Statistics*, 4, 266-298.

Bayesian Additive Regression Trees

- We discuss BART only for regression (see Appendix D for the classification algorithm and more details on the regression algorithm).
- Bagging/RF: average of regression trees, each of which is built using a random sample of data and/or predictors.
- Boosting: weighted sum of trees, each of which is constructed by fitting a tree to the residual of the current fit.
- BART is related to both approaches: each tree is constructed in a random manner as in the former, and each tree tries to capture signal not yet accounted for by the current model as in the latter.
- The main novelty in BART is the way in which new trees are generated.
- Notations: K is the number of trees in each iteration, B is the number of iterations, and $\hat{f}_k^b(x)$ is the prediction at x for the kth tree used in the bth iteration.
- At the end of each iteration, the K trees from that iteration will be summed, i.e.,

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x).$$

Algorithm 8.3 *Bayesian Additive Regression Trees*

1. Let $\hat{f}_1^1(x) = \hat{f}_2^1(x) = \cdots = \hat{f}_K^1(x) = \frac{1}{nK}\sum_{i=1}^n y_i$.
2. Compute $\hat{f}^1(x) = \sum_{k=1}^K \hat{f}_k^1(x) = \frac{1}{n}\sum_{i=1}^n y_i$.
3. For $b = 2, \ldots, B$:
   (a) For $k = 1, 2, \ldots, K$:
      i. For $i = 1, \ldots, n$, compute the current partial residual
      $$r_i = y_i - \sum_{k'<k} \hat{f}_{k'}^b(x_i) - \sum_{k'>k} \hat{f}_{k'}^{b-1}(x_i).$$
      ii. Fit a new tree, $\hat{f}_k^b(x)$, to r_i, by randomly perturbing the kth tree from the previous iteration, $\hat{f}_k^{b-1}(x)$. Perturbations that improve the fit are favored.
   (b) Compute $\hat{f}^b(x) = \sum_{k=1}^K \hat{f}_k^b(x)$.
4. Compute the mean after L burn-in samples,
$$\hat{f}(x) = \frac{1}{B-L}\sum_{b=L+1}^{B} \hat{f}^b(x).$$

Discussions on Algorithm 8.3

- Step 3(a).i: Note that for fitting the partial residual in the kth tree of the bth iteration, only $\hat{f}_{k'}^{b}$, $k' < k$, and $\hat{f}_{k'}^{b-1}$, $k' > k$, are involved. [hand-drawn figure here]
  - (**) This is similar to the Gibbs sampler, a Markov chain Monte Carlo (MCMC) algorithm; this is also the qualifier "Bayesian" in BART from.
  - (**) This is easier to understand through the backfitting algorithm in GAM of Lecture 7.
- Step 3(a).ii: Rather than fitting a fresh tree to this partial residual as in boosting, BART randomly chooses a perturbation to the tree from the previous iteration ($\hat{f}_k^{b-1}$) from a set of possible perturbations, favoring ones that improve the fit to the partial residual.
  - Fitting a perturbation rather than a fresh tree guards against overfitting since it limits how "hard" we fit the data.
  - Also, the individual trees are typically quite small ((**) implied by the prior on the tree structure), so avoid overfitting.
  - (**) More formally, BART avoids overfitting by fitting weak learners through prior distribution specification and posterior sampling and averaging rather than loss minimization as in boosting; in other words, BART is model-based rather than algorithm-based.

Perturbations

- There are two components to the perturbation in Step 3(a).ii:
  1. We may change the structure of the tree by adding or pruning branches.[6] [see Figure 8.12, (d)&(c)]
  2. We may change the prediction in each terminal node of the tree. [see Figure 8.12, (b)]
- (**) The first component is from the proposal distribution for the tree structure in a blocked Metropolis-Hastings algorithm (which is another popular MCMC algorithm), and the second component is from the conditional distribution of the leaf parameters given the tree structure (with the prior implying "shrunk toward zero").
- (**) In summary, Step 3(a) is a Metropolis-within-Gibbs sampler by assuming

$$y_i = f(x_i) + \varepsilon_i, \varepsilon_i \overset{\text{iid}}{\sim} N\left(0, \sigma^2\right),$$

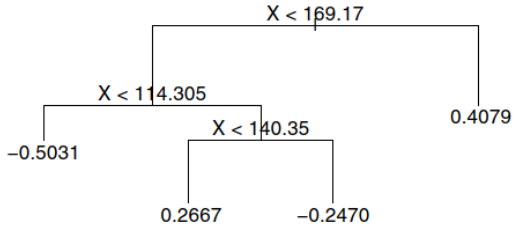
where

$$f(x) = E[Y|X = x] = \sum_{k=1}^{K} g(x, T_k, M_k)$$

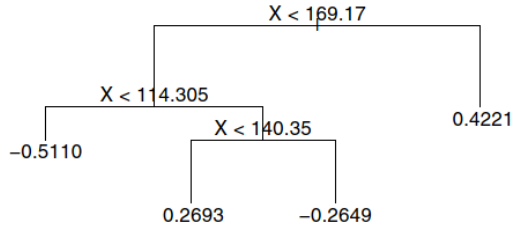
with K being fixed, T_k being the tree structure and M_k being the leaf parameters in the kth tree.

[6] (**) There are two other operations, change an interior node and swap two interior (parent-child) nodes, but these two operations are particularly computationally efficient.

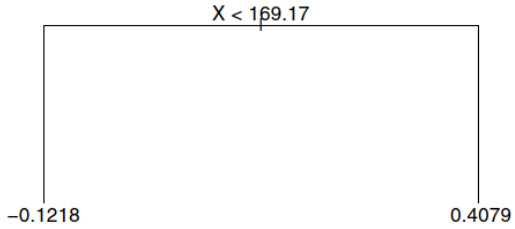
(a): $\hat{f}_k^{b-1}(X)$



(b): Possibility #1 for $\hat{f}_k^b(X)$



(c): Possibility #2 for $\hat{f}_k^b(X)$



(d): Possibility #3 for $\hat{f}_k^b(X)$

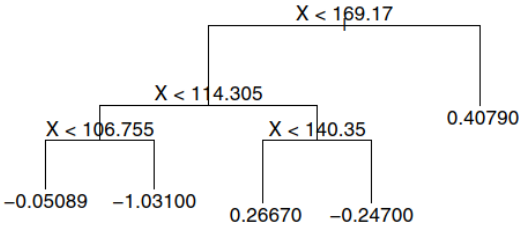


FIGURE 8.12. *A schematic of perturbed trees from the BART algorithm.* (a): *The kth tree at the* $(b-1)$*st iteration,* $\hat{f}_k^{b-1}(X)$*, is displayed. Panels (b)–(d) display three of many possibilities for* $\hat{f}_k^b(X)$*, given the form of* $\hat{f}_k^{b-1}(X)$*.* (b): *One possibility is that* $\hat{f}_k^b(X)$ *has the same structure as* $\hat{f}_k^{b-1}(X)$*, but with different predictions at the terminal nodes.* (c): *Another possibility is that* $\hat{f}_k^b(X)$ *results from pruning* $\hat{f}_k^{b-1}(X)$*.* (d): *Alternatively,* $\hat{f}_k^b(X)$ *may have more terminal nodes than* $\hat{f}_k^{b-1}(X)$*.*

[Example] Heart Data ($K = 200$)

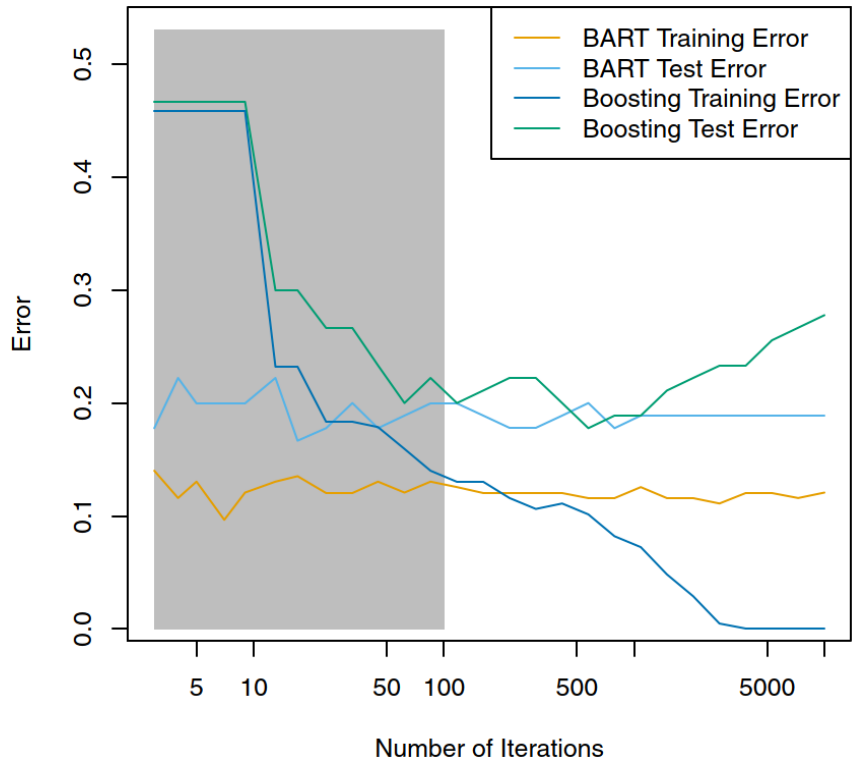


FIGURE 8.13. *BART and boosting results for the* `Heart` *data. Both training and test errors are displayed. After a burn-in period of* 100 *iterations (shown in gray), the error rates for BART settle down. Boosting begins to overfit after a few hundred iterations.*

- See also the discussions in the next slide.

More Discussions on BART

- Compared with boosting, BART tends not to overfit. In Figure 8.13,
  - BART: only a small difference between the training error and the test error, indicating no overfitting.
  - Boosting: the training error decreases as B increases and the test error increases when $B > 500$, indicating overfitting.
- BART has three tuning parameters: K, B and L. We typically choose large values for B and K, and a moderate value for L: e.g., the default $K = 200, B = 1000$, and $L = 100$.
  - BART has been shown to have impressive out-of-box performance – i.e., it performs well with minimal tuning.
- To measure the importance of variables, we can count the average frequency of each variable in all splittings:

$$v_j = \frac{1}{B-L} \sum_{b=L+1}^{B} z_{jb},$$

where z_{jb} is the total number of times that x_j is used in a tree decision rule (over all K trees) of the bth iteration.

Summary of Tree Ensemble Methods

- Trees are an attractive choice of weak learner for an ensemble method because of their flexibility and ability to handle predictors of mixed types.
- Bagging: the trees tend to be quite similar to each other such that bagging can get caught in local optima and can fail to thoroughly explore the model space.
- RF: decorrelate the trees by randomly sampling the features in each splitting, so the model space is more thoroughly explored than bagging.
- Boosting: use only the original data, and no random sampling is involved; use "slow" learning approach: each new tree is fit to the signal left over from the earlier trees, and shrunken down before it is used.
- BART: like boosting, use only the original data, and grow the trees successively, but each tree is perturbed to avoid local minima and achieve a more thorough exploration of the model space.

Lab: Decision Trees (Section 8.3)

- Fitting Classification Trees
- Fitting Regression Trees

- The Bootstrap (Section 5.3.4)
- Bagging
- Random Forests
- Boosting
- BART

Appendix A: More on Entropy

Axioms for Entropy

- If we denote the entropy of a discrete r.v. X as $H(X)$, then

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log p(x_i) =: -\sum_{i=1}^{n} p_i \log p_i =: H_n(p_1,\cdots,p_n).$$

- If the base of the logarithm is b, we denote the entropy as $H_b(X)$.
  - Because $\log_b p = \log_b a \log_a p$, we have $H_b(X) = (\log_b a) H_a(X)$.
- The choice of logarithm in the definition of entropy seems arbitrary; it is actually a must if we assume $H(p_1,\cdots,p_n)$ satisfies the following three natural conditions:
  - (i) Continuity: $H_n(p_1,\cdots,p_n)$ is continuous, i.e., changing the values of the probabilities by a very small amount should only change the entropy by a small amount.
  - (ii) Positive Monotonicity: If $p_i = \frac{1}{n}$, then $H_n\left(\frac{1}{n},\cdots,\frac{1}{n}\right)$ is increasing in n.
  - (iii) Additivity: if an experiment can be decomposed into a few successive experiments, then the entropy of the original experiment is the weighted sum of entropies of these successive experiments.

    - For example, $H_n\left(\frac{1}{n},\cdots,\frac{1}{n}\right) = H_k\left(\frac{b_1}{n},\cdots,\frac{b_k}{n}\right) + \sum_{i=1}^{k} \frac{b_k}{n} H_{b_i}\left(\frac{1}{b_i},\cdots,\frac{1}{b_i}\right)$.
    Choosing $b_i = 1$, we have $H_1(1) = 0$.

Theorem

The H function satisfying the conditions (i)-(iii) must take the form

$$H_n(p_1,\cdots,p_n) = -K\sum_{i=1}^{n} p_i \log p_i,$$

where K plays the role of $\log_b a$ *in base changing.*

Proof.

We prove this theorem in three steps.

Step 1: denote $A(n) = H_n\left(\frac{1}{n},\cdots,\frac{1}{n}\right)$; then $A(s^m) = mA(s)$, where $s,m \in \mathbb{N}$.

By (iii),

$$\begin{aligned} & H_{s^m}\left(\frac{1}{s^m},\cdots,\frac{1}{s^m}\right) \\ = \; & H_s\left(\frac{s^{m-1}}{s^m},\cdots,\frac{s^{m-1}}{s^m}\right) + s\left[\frac{s^{m-1}}{s^m}H_s\left(\frac{s^{m-2}}{s^{m-1}},\cdots,\frac{s^{m-2}}{s^{m-1}}\right)\right] + \cdots + s^{m-1}\left[\frac{s}{s^m}H_s\left(\frac{1}{s},\cdots,\frac{1}{s}\right)\right] \\ = \; & mH_s\left(\frac{1}{s},\cdots,\frac{1}{s}\right). \end{aligned}$$

Proof.

If $s^m \le t^n < s^{m+1}$, where $s,m,t,n \in \mathbb{N}$, then taking logarithm on both sides, we have

$$m\log s \le n\log t < (m+1)\log s,$$

i.e.,

$$\frac{m}{n} \le \frac{\log t}{\log s} < \frac{m}{n} + \frac{1}{n},$$

so

$$\left|\frac{m}{n} - \frac{\log t}{\log s}\right| < \frac{1}{n}. \tag{1}$$

By (ii), $A(n)$ is increasing in n, and from Step 1, we have

$$mA(s) \le nA(t) < (m+1)A(s),$$

i.e.,

$$\frac{m}{n} \le \frac{A(t)}{A(s)} < \frac{m}{n} + \frac{1}{n},$$

so

$$\left|\frac{m}{n} - \frac{A(t)}{A(s)}\right| < \frac{1}{n}. \tag{2}$$

□

Proof.

By (1) and (2), we have

$$\left|\frac{A(t)}{A(s)}-\frac{\log t}{\log s}\right|<\frac{2}{n}.$$

Because the left hand side does not depend on n, n can be arbitrary, which implies

$$\frac{A(t)}{A(s)}=\frac{\log t}{\log s} \text{ or } \frac{A(t)}{\log t}=\frac{A(s)}{\log s}=K>0;$$

in other words,

$$A(t)=K\log t.$$

Hence,

$$H_n\left(\frac{1}{n},\cdots,\frac{1}{n}\right)=K\log n=-K\sum\frac{1}{n}\log\frac{1}{n},$$

i.e.,, the theorem holds for $p_i=\frac{1}{n}$, $i=1,\cdots,n$. □

Proof.

Step 2: If p_i's are nonnegative rational numbers, we can show the theorem holds. First of all, p_i can be written as

$$p_i = \frac{n_i}{\sum_{i=1}^r n_i}, i = 1, \cdots, r.$$

By (iii), we have

$$H_r(p_1, \cdots, p_r) + \sum_{i=1}^r p_i A(n_i) = A\left(\sum_{i=1}^r n_i\right),$$

so from Step 1,

$$\begin{aligned} H_r(p_1, \cdots, p_r) &= A\left(\sum_{i=1}^r n_i\right) - \sum_{i=1}^r p_i A(n_i) = K \log\left(\sum_{i=1}^r n_i\right) - K \sum_{i=1}^r p_i \log(n_i) \\ &= K \sum_{i=1}^r p_i \left[\log\left(\sum_{i=1}^r n_i\right) - \log(n_i)\right] = -K \sum_{i=1}^r p_i \left[\log\left(\frac{n_i}{\sum_{i=1}^r n_i}\right)\right] \\ &= -K \sum_{i=1}^r p_i \log p_i. \end{aligned}$$

Step 3: From Step 2, the general result holds by (i). □

Further Properties of Entropy

- Symmetry: H is unchanged if the outcomes x_i are re-ordered. That is,

$$H_n(p_1, p_2, \cdots, p_n) = H_n(p_{i_1}, \cdots, p_{i_n})$$

for any permutation $\{i_1, \cdots, i_n\}$ of $\{1, \cdots, n\}$.

- Maximum: H is maximized if all outcomes are equally likely, i.e.,

$$H_n(p_1, \cdots, p_n) = H_n\left(\frac{1}{n}, \cdots, \frac{1}{n}\right).$$

- Because $\log \mu$ is a strictly concave function, for any $p_1, \cdots, p_n > 0$ and $\sum_{i=1}^n p_i = 1$,

$$\begin{aligned} H_n(p_1, \cdots, p_n) &= \sum_{i=1}^n p_i \log \frac{1}{p_i} \le \log\left(\sum_{i=1}^n p_i \frac{1}{p_i}\right) \\ &= \log n = H_n\left(\frac{1}{n}, \cdots, \frac{1}{n}\right). \end{aligned}$$

When some p_i, say p_1, equals zero, then

$$H_n(0, p_2, \cdots, p_n) = H_{n-1}(p_2, \cdots, p_n) \le H_{n-1}\left(\frac{1}{n-1}, \cdots, \frac{1}{n-1}\right) < H_n\left(\frac{1}{n}, \cdots, \frac{1}{n}\right),$$

where $0 \log \infty = 0$ by convention, and the last equality is from (ii).

Cross-Entropy

- The cross-entropy of the distribution q relative to the distribution p is defined as $H(p,q) = -\sum_{m=1}^{M} p_m \log(q_m)$, where $(q_1, \cdots, q_M)$ are the predicted probabilities while $(p_1, \cdots, p_M)$ are the true probabilities.
- For example, in the negative multinomial log-likelihood,

$$-\sum_{i=1}^{n}\sum_{m=1}^{M} y_{im} \log\left(f_m(x_i)\right),$$

  $q_{im} = f_m(x_i)$, and $p_{im} = y_{im}$ which takes 1 for some m and 0 for all others since we know the true class to which the ith observation belongs.
- In information theory, the cross-entropy is the average message length to code data generated from p under q (in bits when the base for log is 2); if the predicted probabilities match the true probabilities, the cross-entropy is equal to entropy, and the length is shortest.

Appendix B: AdaBoost and Gradient Boosting (Chapter 10 of ESL)

AdaBoost

- When $Y \in \{-1, 1\}$, we mimic the idea in boosting for regression trees.
- First, we use the exponential loss $L(z) = e^{-z}$ to substitute the squared-error loss, and define some weak learners, e.g., $h_j(x) = \text{sign}(x_j - t_j)$ with $t_j \in \mathbb{R}$, to mimic the stumps in regression trees.
- The principle of AdaBoost is to perform a greedy minimization of

$$\hat{f} = \arg \min_{f \in \text{span}\{h_1, \cdots, h_p\}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \exp(-y_i f(x_i)) \right\}.$$

- More precisely, it computes a sequence of functions $\hat{f}_m$ for $m = 0, \cdots, M$ by starting from $\hat{f}_0 = 0$ and then solving for $m = 1, \cdots, M$

$$\hat{f}_m = \hat{f}_{m-1} + \beta_m h_{j_m},$$

where

$$(\beta_m, j_m) = \arg \min_{\substack{j=1,\cdots,p \\ \beta \in \mathbb{R}}} \frac{1}{n} \sum_{i=1}^{n} \exp\left(-y_i \left(\hat{f}_{m-1}(x_i) + \beta h_j(x_i)\right)\right).$$

- The final classification is performed according to $\hat{h}_M(x) = \text{sign}\left(\hat{f}_M(x)\right)$.

Continued

- The exponential loss allows us to compute (β_m, j_m) very efficiently.
- Actually, setting $w_i^{(m)} = n^{-1} \exp\left(-y_i \hat{f}_{m-1}(x_i)\right)$, we have

$$\begin{aligned} & \frac{1}{n} \sum_{i=1}^{n} \exp\left(-y_i \left(\hat{f}_{m-1}(x_i) + \beta h_j(x_i)\right)\right) \\ = & \sum_{i=1}^{n} w_i^{(m)} \exp\left(-\beta y_i h_j(x_i)\right) \\ = & e^{-\beta} \sum_{y_i = h_j(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq h_j(x_i)} w_i^{(m)} \\ = & \left(e^{\beta} - e^{-\beta}\right) \sum_{i=1}^{N} w_i^{(m)} I\left(y_i \neq h_j(x_i)\right) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)}. \end{aligned}$$

- $w_i^{(m)} > 1/n$ if x_i is misclassified in stage $m-1$ and $w_i^{(m)} < 1/n$ otherwise; $\sum_{i=1}^{n} w_i^{(m)}$ need not equal 1 when $m \geq 1$.

Continued

- When no initial classifier h_j perfectly classifies the data $(x_i, y_i)_{i=1,\cdots,n}$, so that

$$\text{err}_m(j) = \frac{\sum_{i=1}^n w_i^{(m)} I\left(y_i \neq h_j(x_i)\right)}{\sum_{i=1}^n w_i^{(m)}} < 1 \text{ for all } j = 1, \cdots, p.$$

  - $\text{err}_m(j)$ is a weighted classification error rate with the weights determined from the previous stage.
- As a result, the minimizers (β_m, j_m) are given by

$$j_m = \arg\min_{j=1,\cdots,p} \text{err}_m(j) \text{ and } \beta_m = \frac{1}{2}\log\left(\frac{1-\text{err}_m(j_m)}{\text{err}_m(j_m)}\right).$$

  - $\beta_m > 0$ because h_{j_m} is better than random guessing whose error rate is 50%.
  - When h_j depends on unknown parameters such as t_j in $\text{sign}(x_j - t_j)$, we can estimate them together with j_m.
- So fitting the remaining misclassification in the original formulation of AdaBoost turns out to be minimizing a weighted error rate with the weights determined by the remaining misclassification from the last iteration.
- For squared-error loss, $L\left(y_i, \hat{f}_{m-1}(x_i) + \beta h_j(x_i)\right) = \left(y_i - \hat{f}_{m-1}(x_i) - \beta h_j(x_i)\right)^2 = \left(r_{im} - \beta h_j(x_i)\right)^2$, where r_{im} is the residual of the current model on the ith observation, and $\beta h_j(x_i)$ is a shallow tree; but we do not shrink $\beta h_j(x_i)$ in AdaBoost.

AdaBoost in Summary

- Init: $w_i^{(m)} = 1/n$ for $i = 1, \cdots, n$.
- Iterate: For $m = 1, \cdots M$ do

$$\begin{aligned} j_m &= \arg \min_{j=1,\cdots,p} \text{err}_m(j) \text{ with } \text{err}_m(j) = \frac{\sum_{i=1}^n w_i^{(m)} I\left(y_i \neq h_j(x_i)\right)}{\sum_{i=1}^n w_i^{(m)}}, \\ 2\beta_m &= \log\left(\frac{1-\text{err}_m(j_m)}{\text{err}_m(j_m)}\right), \\ w_i^{(m+1)} &= w_i^{(m)} \cdot e^{-\beta_m y_i h_{j_m}(x_i)} = w_i^{(m)} \cdot e^{2\beta_m I(y_i \neq h_{j_m}(x_i)) - \beta_m}, \end{aligned}$$

  where the second equality in $w_i^{(m+1)}$ is due to $-y_i h_{j_m}(x_i) = 2 \cdot I\left(y_i \neq h_{j_m}(x_i)\right) - 1$.
- Output: $\hat{f}_M = \sum_{m=1}^M \beta_m h_{j_m}(x)$, and $\hat{h}_M(x) = \text{sign}\left(\hat{f}_M(x)\right)$.
- From the expression of $w_i^{(m+1)}$, AdaBoost gives more and more weight in $\text{err}_m(j)$ to the data points X_i, which are wrongly classified at the stage m [$2\beta_m I\left(y_i \neq h_{j_m}(x_i)\right) - \beta_m > 0$ if $y_i \neq h_{j_m}(x_i)$], whereas gives less and less weights to those that were classified correctly.

Why Exponential Loss?

- AdaBoost was originally motivated differently.
- The main advantage of exponential loss is computational.
- What does it estimate? It is easy to show that

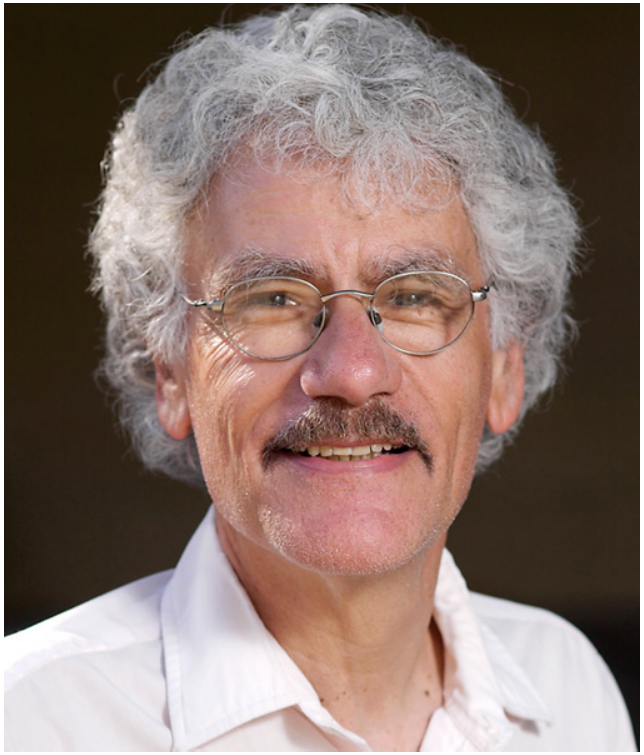
$$\begin{aligned} f^*(x) &= \arg\min_{f(x)} E_{Y|x}\left(e^{-Yf(x)}\right) = \arg\min_{f(x)} e^{-f(x)}\Pr(Y=1|x) + e^{f(x)}\Pr(Y=-1|x) \\ &= \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}, \end{aligned}$$

or equivalently,

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

- So f_M in AdaBoost is estimating one-half the log-odds of $\Pr(Y=1|x)$.
  - $f^*(x) > 0 \Longleftrightarrow \Pr(Y=1|x) > 1/2$, so $h_M(x) = \text{sign}(f_M(x))$.
- The exponential loss imposes exponential penalties on large negative margins $yf(x)$, so is not robust; the deviance loss imposes linear penalties for large negative margins, so is more robust.

History of Gradient Boosting



Jerome H. Friedman (1939-, Stanford)[7]

- Friedman, J.H., 2001, Greedy Function Approximation: A Gradient Boosting Machine, *Annals of Statistics*, 29, 1189-1232.
- Friedman, J.H., 1991, Multivariate Adaptive Regression Splines, *Annals of Statistics*, 19, 1-141.

[7] He is also famous for MARS in Appendix C and projection pursuit.

Gradient Boosting

- For a general loss function, we conduct the gradient boosting:

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Comments on Algorithm 10.3

- $f_0(x)$ is obtained from a single terminal node tree.
- r_{im}'s are referred to as generalized or pseudo residuals.
- Algorithm 10.3 was called MART for "multiple additive regression tress".
- To appreciate the definition of r_{im}, note that for squared-error loss,

$$r_{im} = -\left[\frac{\partial (y_i - f(x_i))^2}{\partial f(x_i)}\right]_{f=f_{m-1}} = y_i - f_{m-1}(x_i);$$

and for exponential loss,

$$r_{im} = -\left[\frac{\partial \exp(-y_i f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}} = \exp(-y_i f_{m-1}(x_i)) y_i = n w_i^{(m)} y_i$$

is a re-weighted y_i.

- In two-class classification, the deviance loss is

$$L(y, f(x)) = -y \log f(x) - (1-y) \log(1 - f(x)),$$

so

$$r_{im} = \frac{y_i}{f_{m-1}(x_i)} - \frac{1-y_i}{1-f_{m-1}(x_i)} = \frac{y_i - f_{m-1}(x_i)}{f_{m-1}(x_i)(1-f_{m-1}(x_i))}.$$

Appendix C: Multivariate Adaptive Regression Splines (MARS)

(Section 9.4 of ESL)

MARS

- MARS uses expansions in piecewise linear basis functions of the form $(x-t)_+$ and $(x-t)_-$. [see Figure 9.9]
- Specifically, the collection of basis functions is

$$\mathscr{C}=\left\{(X_j-t)_+,(t-X_j)_+\right\}_{\substack{t\in\{x_{1j},\cdots,x_{nj}\}\\ j=1,\cdots,p}},^{8}$$

  i.e., totally $2np$ basis functions formed for each input X_j with knots at each observed value x_{ij}, where each basis fucntion is considered as function in $\mathbb{R}^p$ although it depends only on X_j.
- MARS can be viewed as a generalization of stepwise linear regression. Instead of using the original input X_j, it uses functions from $\mathscr{C}$ and their products:

$$f(X)=\beta_0+\sum_{m=1}^{M}\beta_m h_m(X),$$

  where each $h_m(X)$ is a function in $\mathscr{C}$, or a product of two or more such functions.
- Given $\{h_m(X)\}_{m=1}^M$, $\{\beta_m\}_{m=1}^M$ are estimated by least squares.

[8] Does not require feature standardization on X_j, but missing values must be pre-processed. For quantiative predictors, consider all possible binary partitions of the categories; each partition generates a pair of piecewise constant basis functions.

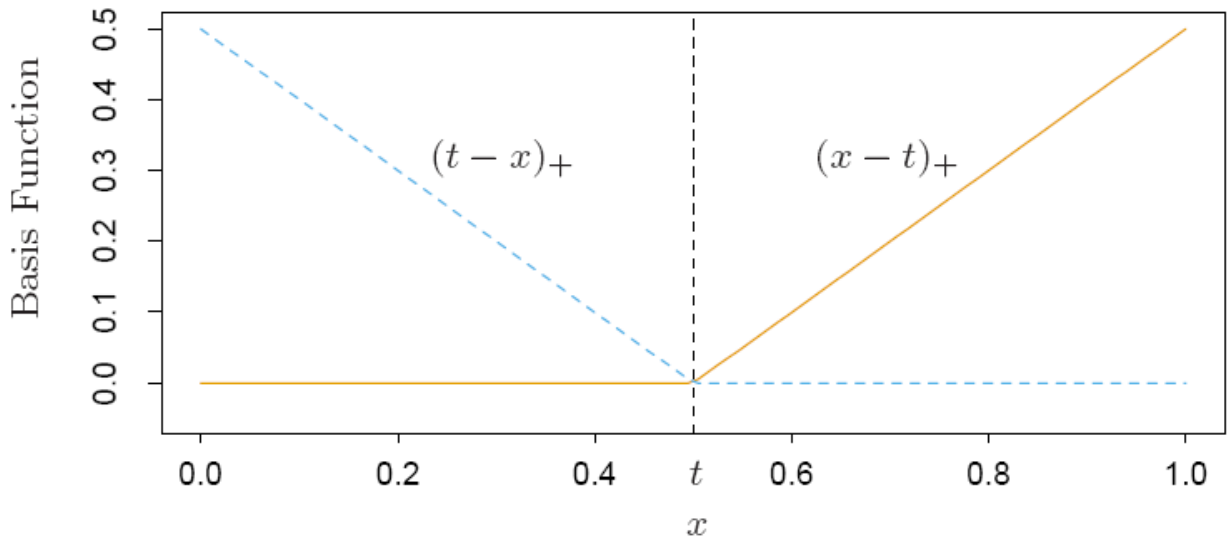


FIGURE 9.9. *The basis functions* $(x-t)_+$ *(solid orange) and* $(t-x)_+$ *(broken blue) used by MARS.*

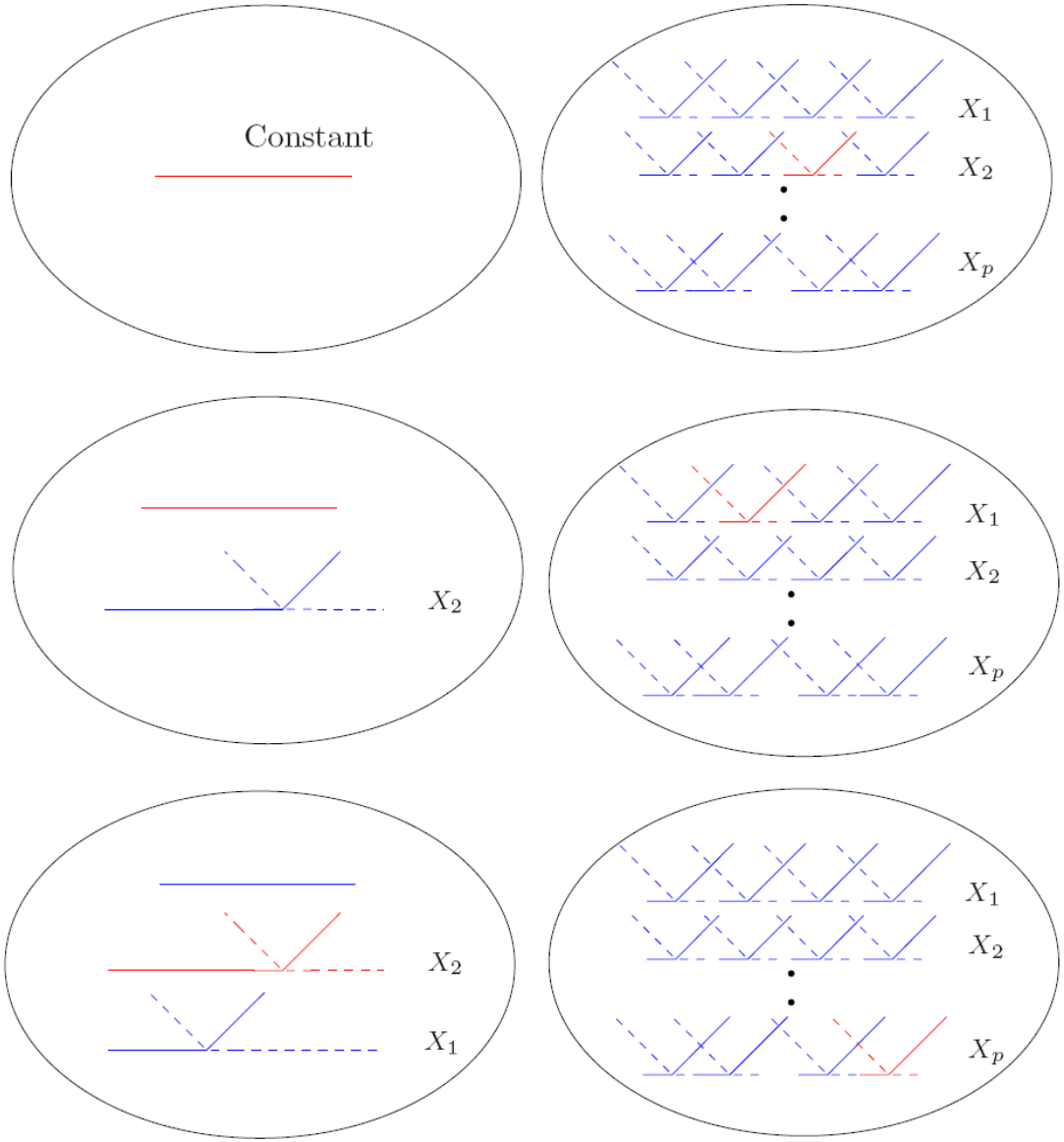
- The basis functions are linear splines with a knot at t.
- $(x-t)_+$ and $(t-x)_+$ are called a reflected pair.

The Forward Pass

- The real art of MARS is in the construction of the functions $h_m(x)$.
- MARS is a greedy forward algorithm for including only those tensor products that are deemed necessary by least squares.
- Start from the constant function $h_0(X) = 1$, and all functions in $\mathscr{C}$ are candidate functions. [figure here]
- Suppose the current model is $\mathscr{M}$ (see the left panel of the figure below). We search for a new basis function pair of this form

$$h_\ell(X) \cdot (X_j - t)_+ \text{ and } h_\ell(X) \cdot (t - X_j)_+$$

  that produces the largest decrease in residual sum-of-squares, where $h_\ell \in \mathscr{M}$, $j = 1, \cdots, p$, and $t \in \{x_{ij}\}$.
  - i.e., we choose one pair from $|\mathscr{M}| \cdot np$ candidate pairs.
- Continue this process until the model set $\mathscr{M}$ contains some preset maximum number of terms.
  - A useful option is to set an upper limit on the order of interaction κ; e.g., if $\kappa = 2$, we search over only the pairs among $|\mathscr{M}| \cdot np$ with $\kappa \leq 2$, which can aid interpretation of the final model, and if $\kappa = 1$, we have an additive model.
- In summary, MARS fits a low-order interaction model adaptively and is suitable to large p problem.


Constant
X_1
X_2
X_p
X_2
X_1
X_2
X_p
X_2
X_1
X_1
X_2
X_p

An Illustrative Example

- At the first stage, we add a pair of the form $(X_j - t)_+$ and $(t - X_j)_+$, $t \in \{x_{ij}\}$, since multiplication by the constant function just produces the function itself.
- Suppose the best choice is $(X_2 - x_{72})_+$ and $(x_{72} - X_2)_+$.
- Now, $\mathscr{M} = \{1, (X_2 - x_{72})_+, (x_{72} - X_2)_+\} =: \{h_0(X), h_1(X), h_2(X)\}$.
- At the second stage, we search for a pair of product in the form

$$h_m(X) \cdot (X_j - t)_+ \text{ and } h_m(X) \cdot (t - X_j)_+, m = 0, 1, 2,$$

  where $j \neq 2$ when $m = 1, 2$, i.e., each input can appear at most once in a product to prevent higher-order powers of an input.
  - This restriction avoids too sharply increase (or decrease) of higher-order powers near the boundaries of the feature space; rather, such powers can be approximated in a more stable way with piecewise linar functions (similar to natural cubic splines).
- The third choice produces functions such as $(X_1 - x_{51})_+ \cdot (x_{72} - X_2)_+$. [see Figure 9.11]

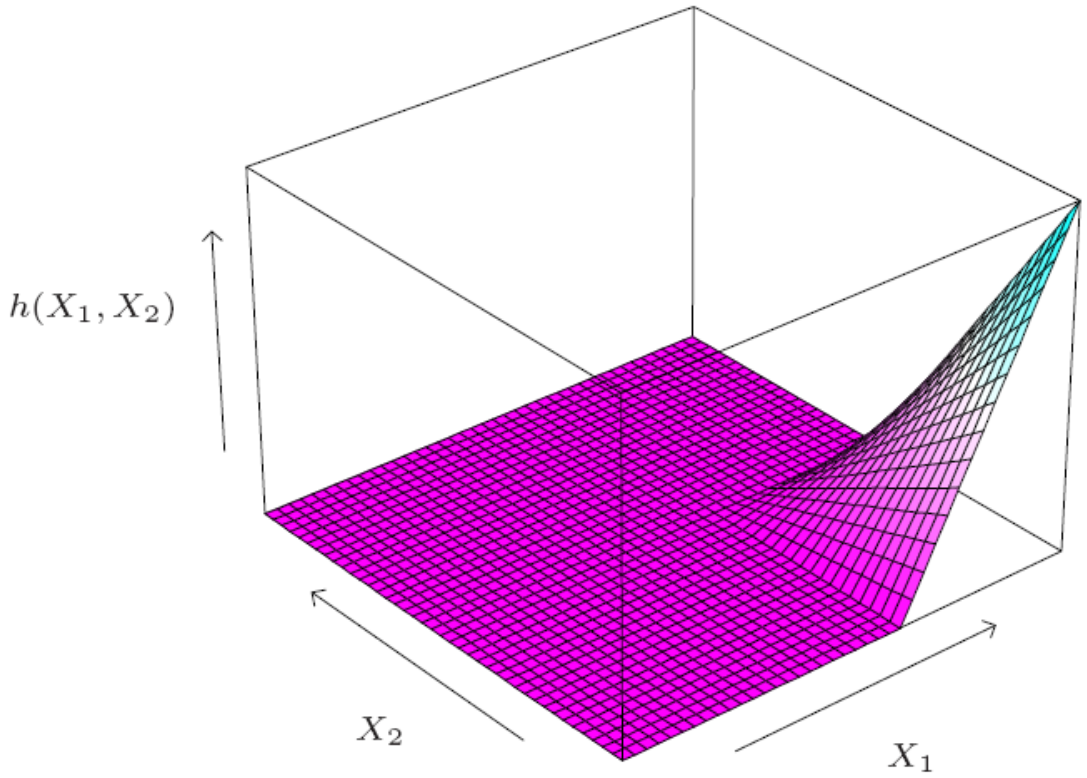


FIGURE 9.11. *The function* $h(X_1, X_2) = (X_1 - x_{51})_+ \cdot (x_{72} - X_2)_+$, *resulting from multiplication of two piecewise linear MARS basis functions.*

The Backward Pass

- The final model in the above procedure is large and typically overfits the data, so a backward deletion procedure is applied.
- At each stage, delete the term (not pair) whose removal causes the smallest increase in residual squared error, producing an estimated best model $\hat{f}_\lambda$ of each size λ, where λ is the number of terms.
- Then choose λ by CV, or for computational savings by GCV, i.e., minimizing

$$\text{GCV}(\lambda) = \frac{\sum_{i=1}^{n}\left(y_i - \hat{f}_\lambda(x_i)\right)^2}{\left(1 - M(\lambda)/n\right)^2},$$

  where $M(\lambda)$ is the effective number of parameters in the model, which accounts both for the number of terms in the models, plus the number of parameters used in selecting the optimal positions of the knots.
- Some mathematical and simulation results suggest that one should pay a price of three parameters for selecting a knot in a piecewise linear regression.
- In other words, $M(\lambda) = \lambda + cK$, where K is number of knots (selected in the forward process) in the model $\hat{f}_\lambda$, and $c = 3$.
  - If the interaction order $\kappa = 1$, set $c = 2$.
  - c is a smoothing parameter; larger c will lead to fewer knots and thereby smoother function estimates.

Advantages of Piecewise Linear Basis Functions

- First, they can operate locally. They are zero over part of their range and when multiplied together, the result is nonzero only over the small part of the feature space where all component functions are nonzero.
  - The regression surface is built up parsimoniously, and the parameters are "spent" only where they are needed, which is important when p is large.
- Second, the structure of $\mathscr{C}$ makes the computation much quickly than the brute-force fit of tons of least squares.

- The forward modeling strategy in MARS is hierarchical, e.g., a four-way product can only be added to the model if one of its three-way components is already in the model.
  - The philosophy here is that a higher-order interaction will likely only exist if some of its lower-oder "footprints" exist as well.
  - This is also the reason why the model can be built up parsimoniously.

- Disadvantages: not accurate if the local linear relationships are incorrect (see Section 9.4.2 of ESL); typically not as accurate as more advanced nonlinear algorithms (random forests, gradient boosting).

[Example] Spam Data

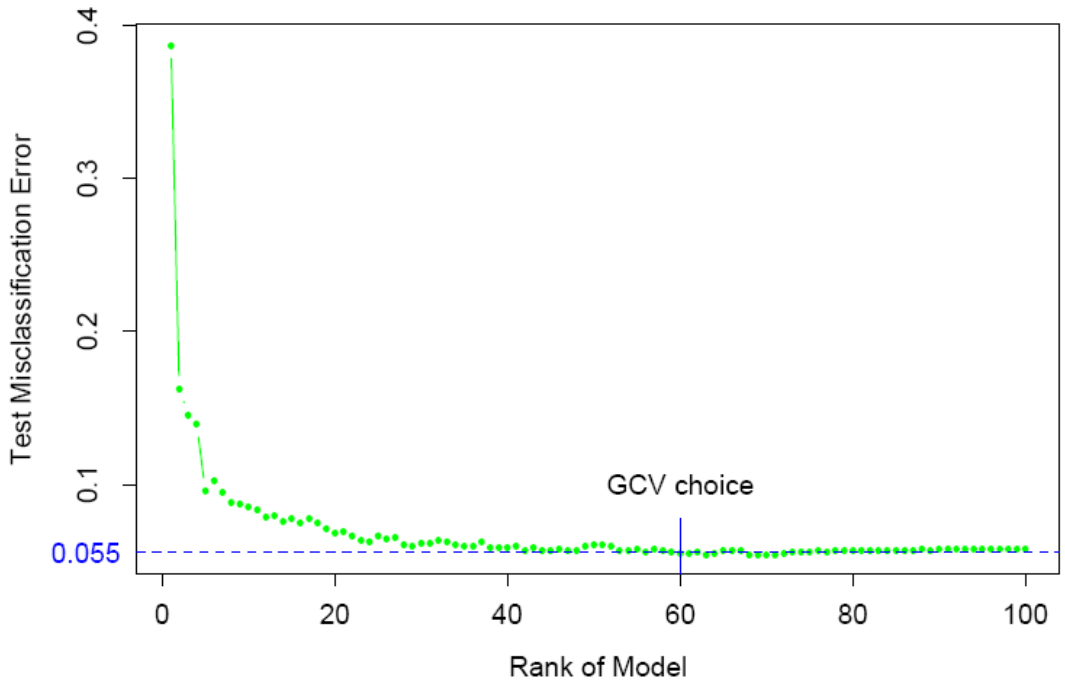


FIGURE 9.12. *Spam data: test error misclassification rate for the MARS procedure, as a function of the rank (number of independent basis functions) in the model.*

- Apply MARS with $\kappa = 2$ although Y is binary; GCV chooses $\lambda = 60$, which is roughly the samllest model giving optimal performace.
- The leading interactions involve $(\texttt{ch\$}, \texttt{remove})$, $(\texttt{ch\$}, \texttt{free})$ and $(\texttt{hp}, \texttt{CAPTOT})$, but these interactions give no improvement in performance over GAM (which involves also cubic terms of a single X_j).

MARS for Classification

- When $K = 2$, code Y as $0/1$, and perform MARS as in the above example.
- When $K > 2$, code Y as a K-dimensional $0/1$ vector, and perform a multi-response MARS regression.
  - Use common set of basis functions for all K response variables.
  - Classification is made to the class with the largest predicted response value.
  - There are potential masking problems; a generally superior approach is the "optimal scoring" method discussed in Section 12.5.1 of ESL.
- PolyMARS: use the multinomial logit, but at each stage use a quadratic approximation to the log-likelihood to search for the next basis-function pair; once found, the enlarged model is fit by maximum likelihood, and the process is repeated.

Relationship of MARS to CART

- MARS can also be viewed as an improvement of CART in the regression setting.
- MARS is the same as CART if we make the following two changes:

1. Replace the piecewise linear basis functions by step functions $I(x-t>0)$ and $I(x-t\leq 0)$.
   - Multiplying a step function by a pair of reflected step functions is equivalent to splitting a node at the step.
2. When a model term in $\mathscr{M}$ is involved in a multiplication by a candidate term, it gets replaced by the interaction, and hence is not available for further interactions.
   - It implies that a node may not be split more than once.
   - It also implies that CART cannot model additive structures.

Appendix D: More on BART

The Model

- Suppose

$$y = f(x) + \varepsilon := \sum_{j=1}^{m} g(x; T_j, M_j) + \sigma\varepsilon, \tag{3}$$

  where $\varepsilon \sim N(0,1)$, T_j summarizes the structure (i.e., topology and splitting rules) of the jth binary regression tree, and M_j summarizes the terminal node parameters.
- Although a single tree is enough to approximate $f(x)$, BART borrows an idea from boosting and uses the sum of many small trees to approximate $f(x)$.
- Like other ensemble methods, a sum-of-trees model tends to be more robust (to a small change in data) and less variable (in prediction) than a single-tree model.
- Here, we follow the notations of BART in the original paper, which is a little different from those in the main text.

Prior Specification

- The unknown parameters include m, $\left\{T_j, M_j\right\}_{j=1}^m$, and σ.
- In practice, we usually fix m at a large (default) integer, say 200, rather than treat it as random.
- For other parameters, the BART priors impose independence and symmetry.
- Specifically, we assume

$$p\left(\left(T_1, M_1\right), \cdots, \left(T_m, M_m\right), \sigma\right) = \prod_{j=1}^m p\left(T_j, M_j\right) p\left(\sigma\right) = \left[\prod_{j=1}^m p\left(M_j | T_j\right) p\left(T_j\right)\right] p\left(\sigma\right)$$

and

$$p\left(M_j | T_j\right) = \prod_{t=1}^{\kappa_j} p\left(\mu_{tj} | T_j\right),$$

where κ_j is the number of terminal nodes of the jth tree, and μ_{tj} is the mean assigned to the tth terminal node of the jth tree.

- Furthermore, we assume $p\left(\mu_{tj} | T_j\right)$ and $p\left(T_j\right)$ are the same for each t and j, thus we need only specify three priors, $p\left(\mu_{tj} | T_j\right)$, $p\left(T_j\right)$ and $p\left(\sigma\right)$.

$p\left(T_j\right)$

- Given the data, there are only finite many possible trees, but BART does not impose prior on each of these trees because there are too many; rather, it describes how to grow a tree which implies a prior on each of the possible trees.
- (i): The probability of splitting a node (i.e., the node is not terminal) at depth d $(=0,1,2,\cdots)$ is

$$\alpha\left(1+d\right)^{-\beta},\ \alpha\in\left(0,1\right),\beta\in\left[0,\infty\right).$$

  - A larger α and smaller β imply a larger tree; the default $\left(\alpha,\beta\right)$ is $\left(0.95,2\right)$.
  - Because the probability is decreasing in d, this prior implies "bushy" trees, i.e., trees with terminal nodes having similar depths.
- (ii): In case of splitting an interior node, each variable has the same probability to be chosen as the splitting variable.
- (iii): Given a splitting variable, there are finite discrete possible splitting values; each possible splitting rule has the same probability to be employed.
  - (ii) and (iii) indicate that BART uses the uniform prior on the splitting variables and rules.

$p\left(\mu_{tj}|T_j\right)$

- BART uses conjugate normal priors on μ_{tj}, $N\left(\mu_\mu,\sigma_\mu^2\right)$, where "conjugate" means that the prior takes the same functional form as the likelihood function.
- To specify the hyperparameters μ_μ and σ_μ^2, BART assumes the resulting prior on $f(x)$ can cover the data with high probability.
- Since μ_{tj}'s are i.i.d., the induced prior on $f(x)$ is $N\left(m\mu_\mu,m\sigma_\mu^2\right)$. So we choose μ_μ and σ_μ^2 such that

$$m\mu_\mu-k\sqrt{m}\sigma_\mu=y_{\min}\text{ and }m\mu_\mu+k\sqrt{m}\sigma_\mu=y_{\max}$$

for some k, where $y_{\min}$ and $y_{\max}$ are the minimum and maximum of y_i, respectively.
- The default $k=2$ implies 95% of the prior probability of $f(x)$ falls in $(y_{\min},y_{\max})$.
- If y_i is shifted and rescaled such that $y_{\min}=-0.5$ and $y_{\max}=0.5$, then $\mu_\mu=0$ and $\sigma_\mu=\frac{0.5}{k\sqrt{m}}$, which implies shrinkage toward the center of y_i, more so for large m and/or k.

$p(\sigma)$

- BART also uses a conjugate prior on σ, $\sigma^2 \sim \text{IG}\left(\frac{\nu}{2}, \frac{\nu\lambda}{2}\right)$, the inverse Gamma distribution, which implies $\sigma^2 \sim \nu\lambda/\chi^2_\nu$.
- We still use data to guide the specification of the hyperparameters ν and λ.
- For a rough over-estimate of σ, $\widehat{\sigma}$, e.g., the standard error of the linear regression, we pick ν and λ such that

$$P(\sigma < \widehat{\sigma}) = q$$

for some q.
- The default (ν, q) is $(3, 0.90)$.

- BART seems robust to small changes in the specification of $(\alpha, \beta, m, k, \nu, q)$.
- We can use CV to choose (m, k, ν, q) for improvement in prediction, but it is more time-consuming.
- Compared with boosting where m is chosen by CV to avoid overfitting, m in BART is usually fixed, and a large m only slightly degrades its performance.
- Like neural networks in Lecture 10, BART is overparametrized when m is large; although the "fit" can be shifted among the trees when m is large, the overall fitting is stable to m.

Posterior Simulation

- At a general level, BART is a Gibbs sampler.
- We start the algorithm by setting

$$g^0(x;T_j,M_j)=\frac{1}{m}\bar{y}, \text{ and } f^0(x)=\bar{y},$$

  i.e., trees with a single node at depth $d=0$.
- We then successively draw $(T_1,M_1),\cdots,(T_m,M_m)$ and σ conditional on the data and the other parameters:

$$(T_j,M_j)\,|T_{(j)},M_{(j)},\sigma,\mathscr{D},$$
$$\sigma|T_1,\cdots,T_m,M_1,\cdots,M_m,\mathscr{D},$$

  where $\mathscr{D}$ is the observed data, $T_{(j)}$ is the set of all trees except T_j, and $M_{(j)}$ is similarly defined.
- For $(T_j,M_j)\,|T_{(j)},M_{(j)},\sigma,\mathscr{D}$, BART takes a blocked Metropolis-Hastings (MH) approach, i.e., first simulates the marginal posterior distribution of T_j, and then simulates M_j conditional on T_j.
- For $\sigma|T_1,\cdots,T_m,M_1,\cdots,M_m,\mathscr{D}$, it can be shown that

$$p(\sigma|T_1,\cdots,T_m,M_1,\cdots,M_m,\mathscr{D})=\text{IG}(\frac{\nu+n}{2},\frac{\nu\lambda+n\overline{\varepsilon^2}}{2}),$$

  where $\overline{\varepsilon^2}=\frac{1}{n}\sum_{i=1}^n\varepsilon_i^2$ with $\varepsilon_i=y_i-\sum_{j=1}^m g(x_i;T_j,M_j)$.

Simulate $\left(T_j, M_j\right) | T_{(j)}, M_{(j)}, \sigma, \mathscr{D}$

- It can be shown that $p\left(T_j, M_j | T_{(j)}, M_{(j)}, \sigma, \mathscr{D}\right)$ depends on $\left(T_{(j)}, M_{(j)}, \mathscr{D}\right)$ only through

$$R_{ij} := y_i - \sum_{k \neq j} g\left(x_i; T_k, M_k\right),$$

  which follows $N\left(g\left(x_i; T_j, M_j\right), \sigma^2\right)$, so we can treat R_j as the response and adopt the single-tree MCMC algorithm.
- The integrated likelihood of T_j is

$$L\left(T_j | T_{(j)}, M_{(j)}, \sigma, \mathscr{D}\right) = \int \prod_{i=1}^{n} p\left(R_{ij} | M_j, T_j, T_{(j)}, M_{(j)}, \sigma\right) p\left(M_j | T_j, \sigma\right) dM_j,$$

  which has a closed-form expression given the conjugate prior on μ_{tj}.
- We then generate a candidate tree $T_j^* \sim q(T_j^* | T_j)$, a proposal distribution, and accept this new T_j^* with probability

$$\min\left\{\frac{q\left(T_j | T_j^*\right)}{q\left(T_j^* | T_j\right)} \frac{L\left(T_j^* | T_{(j)}, M_{(j)}, \sigma, \mathscr{D}\right) p\left(T_j^*\right)}{L\left(T_j | T_{(j)}, M_{(j)}, \sigma, \mathscr{D}\right) p\left(T_j\right)}, 1\right\}.$$

- Given the updated T_j, we sample

$$M_j \sim p(M_j | T_j, T_{(j)}, M_{(j)}, \sigma, \mathscr{D}).$$

The Proposal Distribution

- The candidate trees take four possible forms of perturbations on T_j in the previous round:
- (i) grow: randomly pick a terminal node, and split it into two new ones based on the prior;
- (ii) prune: randomly pick a parent of two terminal nodes and collapse the nodes below it;
- (iii) change: randomly pick an internal node, and reassign a splitting rule based on the prior;
- (iv) swap: randomly pick a parent-child pair that are both internal nodes and swap their splitting rules.
  - The default probabilities of these four perturbations are $(0.25, 0.25, 0.4, 0.1)$.
  - Among the four operations, the birth/death pair (i)/(ii) are particularly computationally efficient.
  - All these perturbations are reversible, so $q(T_j|T_j^*)$ and $q(T_j^*|T_j)$ are both meaningful.
- One advantage of this MH algorithm is that by using the prior as the splitting rule in the grow (prune) step, there is substantial cancelation between $p(T_j^*)$ ($p(T_j)$) and $q(T_j^*|T_j)$ ($q(T_j|T_j^*)$), and in the change and swap steps, $\frac{q(T_j|T_j^*)}{q(T_j^*|T_j)} = 1$ so calculation of the q values is avoided.

Classification

- If $Y \in \{0,1\}$, assume
$$p(x) = \Pr(Y = 1|X = x) = \Phi(G(x)),$$
where $\Phi(\cdot)$ is the cdf of $N(0,1)$, and $G(x) = \sum_{j=1}^{m} g(x; T_j, M_j)$.
- Each classification probability $p(x)$ is obtained as a function of $G(x)$, our sum of regression trees, which contrasts with the often used aggregate classifier approaches which use a majority or an average vote based on an ensemble of classification trees.
- The prior specification is similar as in the regression case, but now $\sigma = 1$ is fixed, $(y_{\min}, y_{\max}) = (\Phi^{-1}(-3), \Phi^{-1}(3))$, and $\mu_{tj} \sim N(0, \frac{3}{k\sqrt{m}})$.
- To re-use the algorithm for regression, introduce latent variables $Z_1, \cdots, Z_n \overset{\text{i.i.d.}}{\sim} N(G(x), 1)$ such that $Y_i = 1$ if $Z_i > 0$ and $Y_i = 0$ if $Z_i \leq 0$.
- Note that $Z_i|(Y_i = 1) \sim \max\{N(G(x), 1), 0\}$ and $Z_i|(Y_i = 1) \sim \min\{N(G(x), 1), 0\}$.
- The Gibbs sampler iterations here entail n successive draws of $Z_i|y_i$, $i = 1, \cdots, n$, followed by m successive draws of $(T_j, M_j)|T_{(j)}, M_{(j)}, z_1, \cdots, z_n$, $j = 1, \cdots, m$.
- The induced sequence of sum-of-trees functions
$$p^*(\cdot) = \Phi(\sum_{j=1}^{m} g(\cdot; T_j^*, M_j^*))$$
for the sequence of draws $(T_1^*, M_1^*), \cdots, (T_m^*, M_m^*)$, is thus converging to the posterior distribution on the "true" $p(\cdot)$.