Decision Trees, Random Forests, Boosting

ISLR Chapter 8

Yuan YAO

Hong Kong University of Science and Technology

Department of Mathematics

Fall 2021

Best Machine Learning Algorithms in history?

- **Boosting (ISLR chapter 8): CART, AdaBoost, Random Forests (Leo Breiman)**, etc.
- Support Vector Machines (ISLR chapter 9): or kernel methods (V. Vapnik), etc.
- Neural Networks: perceptrons (1950s), deep learning (2010s), CNN/RNN/LSTM (¡2000), ResNet/Transformers (¿2015), etc.

The basics of decision trees.

- Regression Trees
- Classification Trees

Bagging, random forests and boosting

- Bagging
- Random Forest
- Boosting

About this chapter

- Decisions trees: splitting each variable sequentially, creating rectangular regions.
- Making fitting/prediction locally at each hyper-rectangular region.
- It is intuitive and easy to implement, may have good interpretation.
- Generally of lower prediction accuracy (weak learners).
- "The Boosting problem" (Kearns & Valiant): *Can a set of weak learners create a single strong learner?*
  - Bagging, random forests and boosting ... make fitting/prediction based on a number (ensemble) of trees.
  - Bagging and Boosting are general methodologies, not just limited to trees.

About this chapter

- Decisions trees: splitting each variable sequentially, creating rectangular regions.
- Making fitting/prediction locally at each hyper-rectangular region.
- It is intuitive and easy to implement, may have good interpretation.
- Generally of lower prediction accuracy (weak learners).
- "The Boosting problem" (Kearns & Valiant): *Can a set of weak learners create a single strong learner?*
  - Bagging, random forests and boosting ... make fitting/prediction based on a number (ensemble) of trees.
  - Bagging and Boosting are general methodologies, not just limited to trees.

About this chapter

- Decisions trees: splitting each variable sequentially, creating rectangular regions.
- Making fitting/prediction locally at each hyper-rectangular region.
- It is intuitive and easy to implement, may have good interpretation.
- Generally of lower prediction accuracy (weak learners).
- "The Boosting problem" (Kearns & Valiant): *Can a set of weak learners create a single strong learner?*
  - Bagging, random forests and boosting ... make fitting/prediction based on a number (ensemble) of trees.
  - Bagging and Boosting are general methodologies, not just limited to trees.

Interpretability vs. Prediction

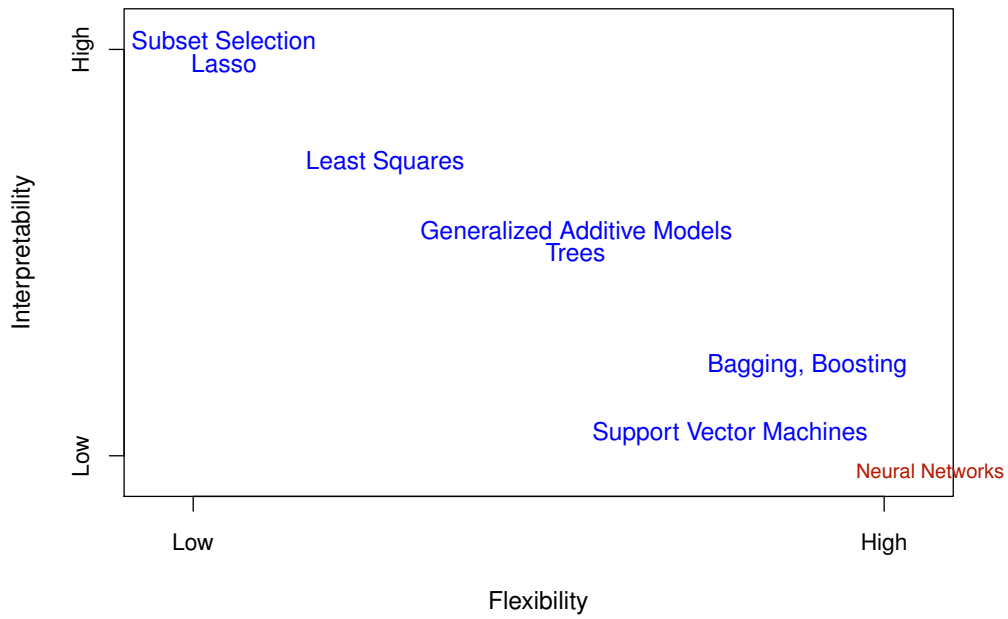


Figure: 2.7. As models become flexible, interpretability drops. **Occam Razor principle**: Everything has to be kept as simple as possible, but not simpler (Albert Einstein).

Outline

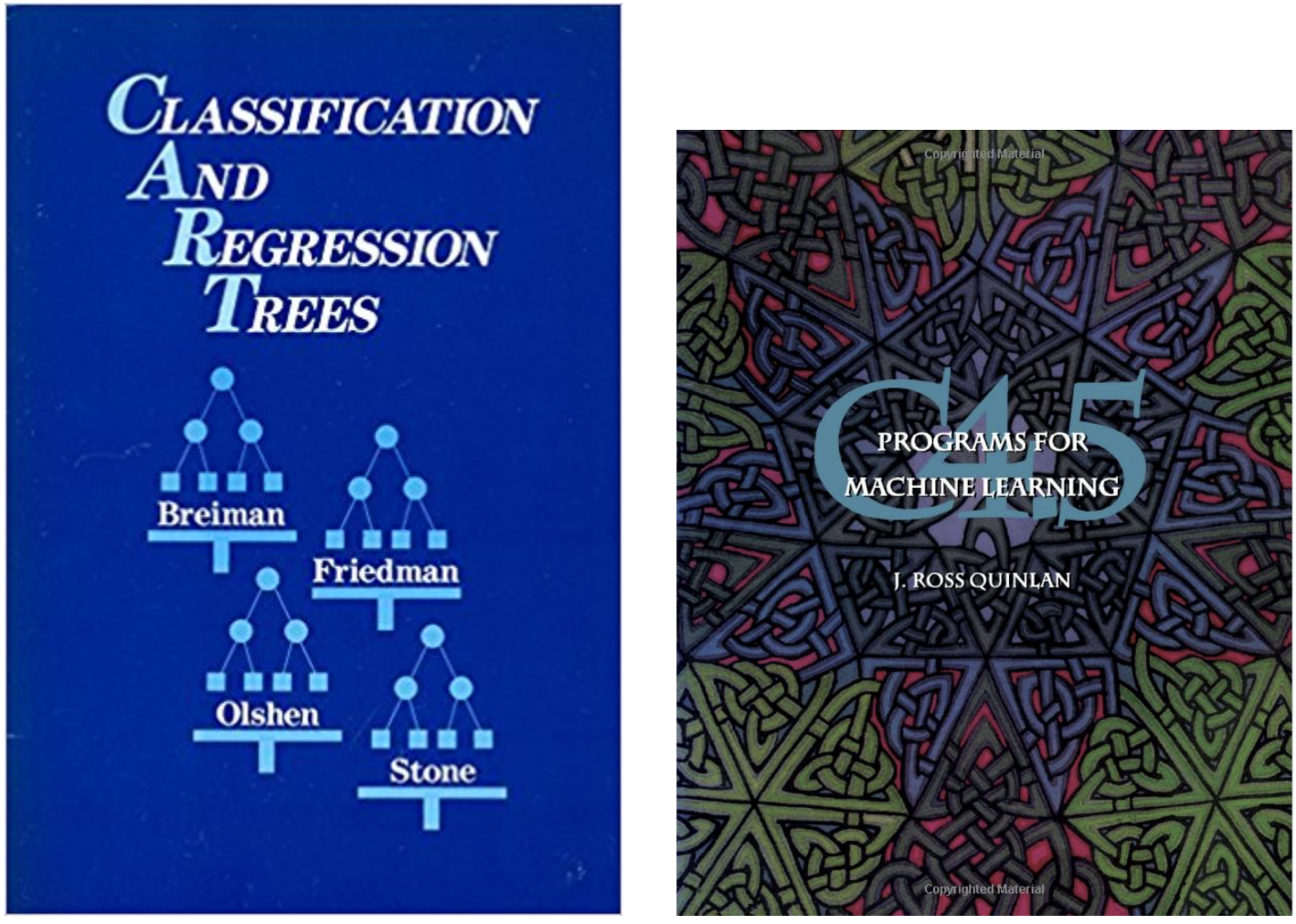


Figure: CART and C4.5

PREFACE

The tree methodology discussed in this book is a child of the computer age. Unlike many other statistical procedures which were moved from pencil and paper to calculators and then to computers, this use of trees was unthinkable before computers.

Binary trees give an interesting and often illuminating way of looking at data in classification or regression problems. They should not be used to the exclusion of other methods. We do not claim that they are always better. They do add a flexible nonparametric tool to the data analyst's arsenal.

Both practical and theoretical sides have been developed in our study of tree methods. The book reflects these two sides. The first eight chapters are largely expository and cover the use of trees as a data analysis method. These were written by Leo Breiman with the exception of Chapter 6 by Richard Olshen. Jerome Friedman developed the software and ran the examples.

Chapters 9 through 12 place trees in a more mathematical context and prove some of their fundamental properties. The first three of these chapters were written by Charles Stone and the last was jointly written by Stone and Olshen.

Trees, as well as many other powerful data analytic tools (factor analysis, nonmetric scaling, and so forth) were originated

by social scientists motivated by the need to cope with actual problems and data. Use of trees in regression dates back to the AID (Automatic Interaction Detection) program developed at the Institute for Social Research, University of Michigan, by Morgan and Sonquist in the early 1960s. The ancestor classification program is THAID, developed at the institute in the early 1970s by Morgan and Messenger. The research and developments described in this book are aimed at strengthening and extending these original methods.

Our work on trees began in 1973 when Breiman and Friedman, independently of each other, "reinvented the wheel" and began to use tree methods in classification. Later, they joined forces and were joined in turn by Stone, who contributed significantly to the methodological development. Olshen was an early user of tree methods in medical applications and contributed to their theoretical development.

Our blossoming fascination with trees and the number of ideas passing back and forth and being incorporated by Friedman into CART (Classification and Regression Trees) soon gave birth to the idea of a book on the subject. In 1980 conception occurred. While the pregnancy has been rather prolonged, we hope that the baby appears acceptably healthy to the members of our statistical community.

Figure: Historical story about CART: dates back to Automatic Interaction Detection ... in 1960s

Regression trees

- Trees can be applied to both regression and classification.
- CART refers to *classification and regression trees*.
- We first consider regression trees through an example of predicting Baseball players' salaries.

The Hitters data

- Response/Outputs: Salary.
- Predictors/Inputs/Covariates:
  Years (the number of years that he has played in the major leagues)
  Hits (the number of hits that he made in the previous year).
- preparing data: remove the observations with missing data and log-transformed the Salary (preventing heavy right-skewness)

Figure: 1. Next page

Figure 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.

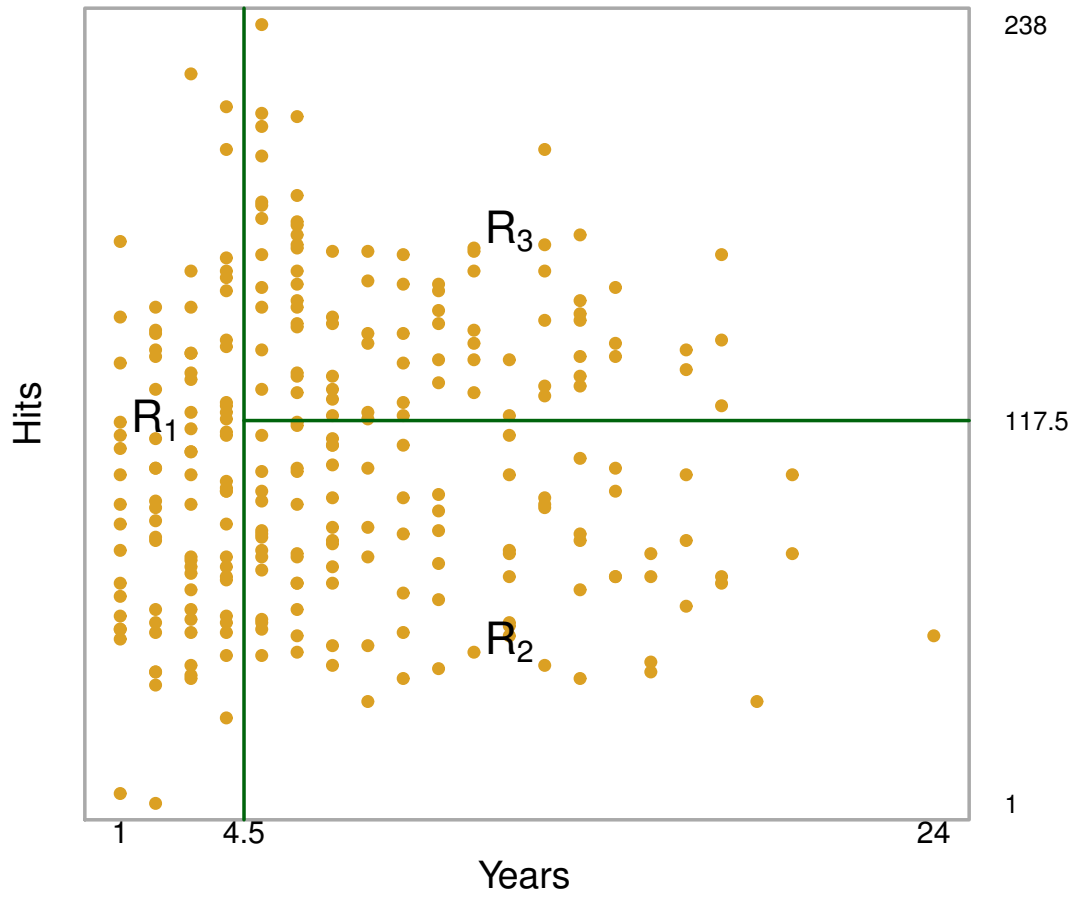


Figure: 2. The three-region partition for the Hitters data set from the regression tree illustrated in Figure 1.

Estimation/prediction

- On Regions R_1, R_2, R_3, the mean-log-salary is 5.107, and 6.74.
- Our prediction for any players in R_1, R_2 and R_3 are, respectively
  $1,000 \times e^{5.107} = \$165,174$, $1,000 \times e^{5.999} = \$402,834$, and
  $1,000 \times e^{6.740} = \$845,346$.

Estimation/prediction

- Trees involve a series of splittings of the data, each time by one variable.
- The series of actions taken place sequentially creates a tree-like results.
- As in Figure 1, the terminal nodes are the three indexed by the numbers, which represent the regions R_1, R_2 and R_3. These regions constitute the final partition of the data.
- Terminal nodes are also called leaves.
- Each *internal node* represents a splitting,
- In Figure 1, the two internal nodes are indexed by Years < 4.5 and Hits < 117.5.
- The lines connecting nodes are called branches.
- Trees are typically drawn upside down.

Two step towards prediction

- Run the splitting according to input values sequentially, and obtain final partition of the data in regions $R_1, ..., R_J$.
- For any new observation with covariates in region R_k, we predict its response by the average of the reponses of the data points in region R_k.

How to split

- Suppose we wish to partition a region R. In other words, we wish to separate the data in region R into two parts, day R_1 and R_2, according to one input values.
- What would be the optimal or efficient split in some sense?
- Only two parameters in the split:
  1. Choice of the input variable to split,
  2. the cutpoint of the split of that chose input.
- Imagine that this is the final split of R: R_1 and R_2 would be leaves.
  - And we would use the mean response of data in R_1 and R_2 to predict the response of any new/old observations.
  - We wish our choice of R_1 and R_2 would be optimal in the sense of achieving minimum prediction error on the training data in region R.

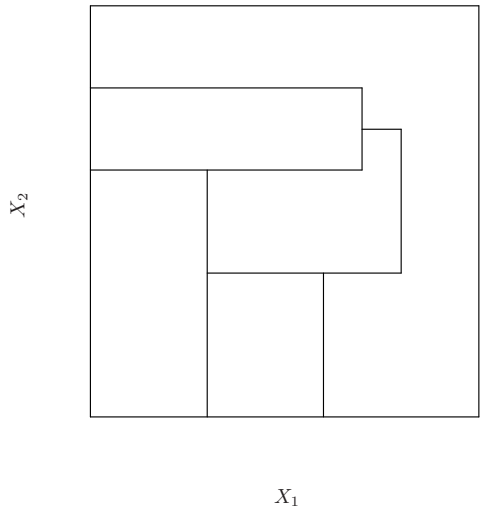
Recursive binary splitting

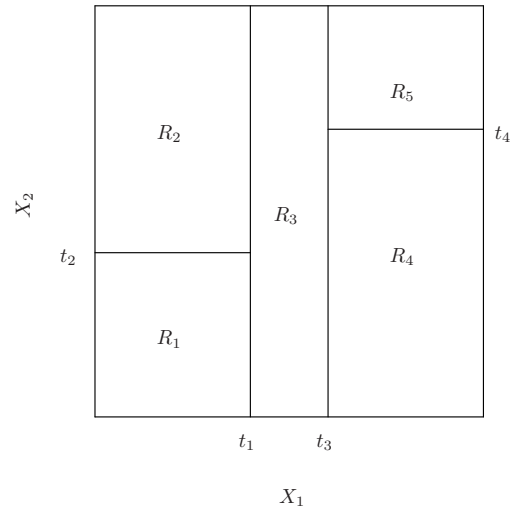
- A greedy algorithm (geedy means it is optimal at the current step):
  1. For $j = 1, ..., p$ and all real value s, let $R_1(j, s) = \{i \in R : X_j < s\}$ and $R_2(j, s) = \{i \in R : X_j \geq s\}$. And let $\hat{y}_1$ and $\hat{y}_2$ be the mean response of all observations in $R_1(j, s)$ and $R_2(j, s)$, respectively.
  2. Consider the following prediction error:

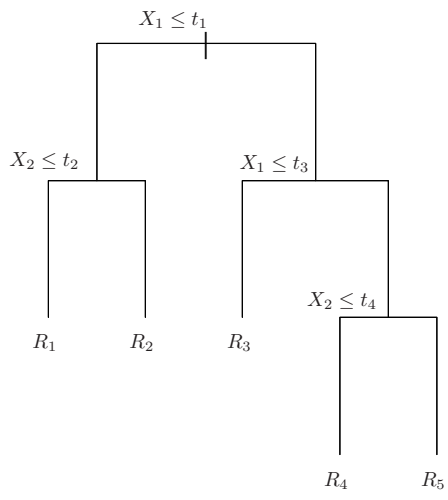
$$\text{RSS}_{new} = \sum_{i \in R_1(j,s)} (y_i - \hat{y}_1)^2 + \sum_{i \in R_2(j,s)} (y_i - \hat{y}_2)^2$$

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

- Continue the split till the final partition.


X_2
X_1


R_5
R_2
t_4
X_2
R_3
t_2
R_4
R_1
t_1
t_3
X_1


$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5

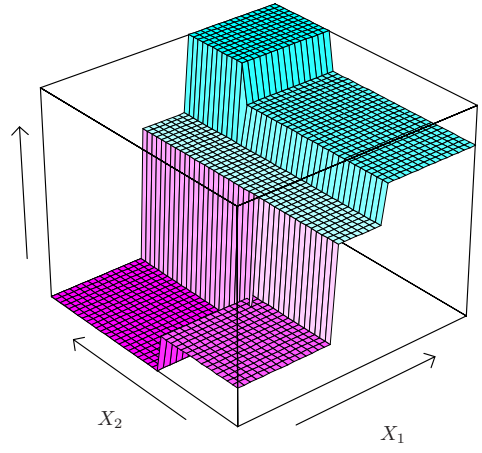

X_2
X_1

Figure 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.

When to stop split

- The problem of when to stop.
- If too many steps of splitting: many leaves, too complex model, small bias but large variance, may overfit.
- If too few steps of splitting: few leaves, too simple model, large bias but small variance, may underfit.

One natural idea

- When splitting R into R_1 and R_2, consider the RSS before the split

$$\text{RSS}_{old} = \sum_{i \in R} (y_i - \hat{y})^2$$

where $\hat{y}$ is the average of the response of data in R. With the optimal split, the reduction of RSS is

$$\text{RSS}_{old} - \text{RSS}_{new}$$

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

One natural idea

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

Tree pruning

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

Cost complexity pruning

- Let T_0 be the original (large) tree. Let T be any subtree. Use $|T_0|$ and $|T|$ to denote their numbers of teminal nodes, which represent complexity.
- Consider "Loss + Penalty":

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

  where R_m are the terminal nodes of the subtree T, and the mean response of R_m is $\hat{y}_m$; α is tuning parameter.
- Denote the minimized subtree as T_α.
  - If $\alpha = 0$, no penalty the optimal tree is the original T_0.
  - If $\alpha = \infty$, the tree has no split at all. The predictor is just $\bar{y}$.
  - The larger the α, the more penalty for model complexity.

Cost complexity pruning

- Just like Lasso, there exists efficient computation algorithm to compute the entire sequence of T_α for all α.
- Use cross-validation to find the best α to minimize the test error.

The 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 α.

The algorithm (continued)

- 3. Use K-fold cross-validation to determine best α. That is, divide the training observations into K folds. For each $k = 1, ..., K$
  - (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
  - (b) Evaluate the mean squared prediction error on the data in the left-out k-th fold, as a function of α.
  - (c) Average the results for each value of α, and pick α to minimize the average error.
- 4. Return the subtree from Step 2 that corresponds to the chosen value of α.

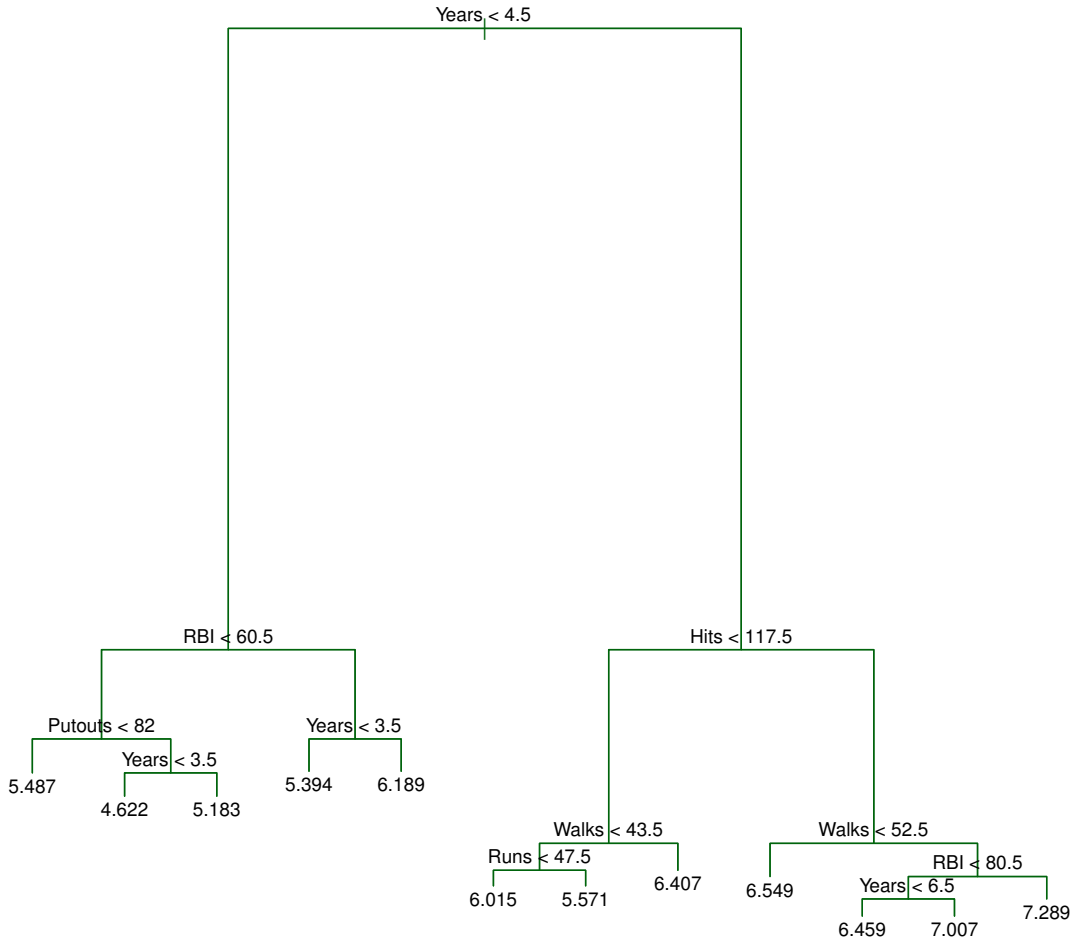


Figure: 4. Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown.

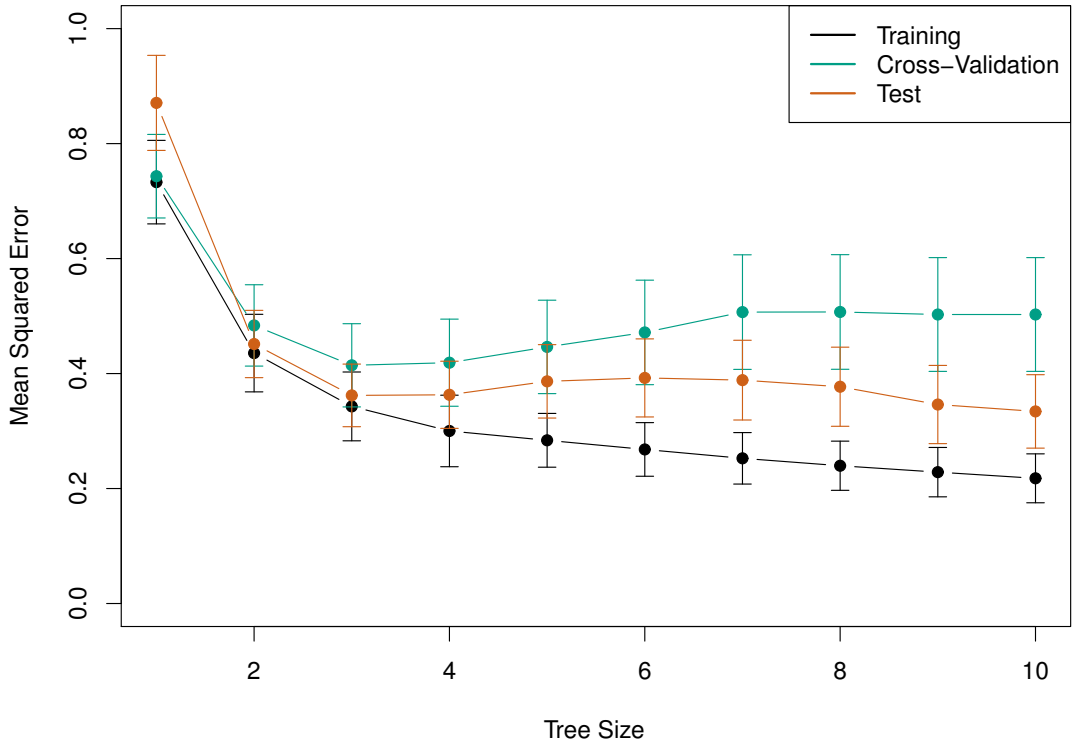


Figure: 5. Regression tree analysis for the Hitters data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.

Classification trees

- Regression has numerical responses; and classification has qualitative responses.
- Recall that for regression trees, we chose to obtain the greatest reduction of RSS.
  RSS is using sum of squares to measure the error.
- For classification trees, one can follow the same line of procedure as that of regression trees, but using error measurements that are more appropriate for classification.

Classification error rates

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

The impurity measure

- The classification error rate (for this region R) is

$$E = 1 - \max_k \hat{p}_k.$$

- The Gini index is

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

- The cross-entropy is

$$D = -\sum_{k=1}^{K} \hat{p}_k \log(\hat{p}_k)$$

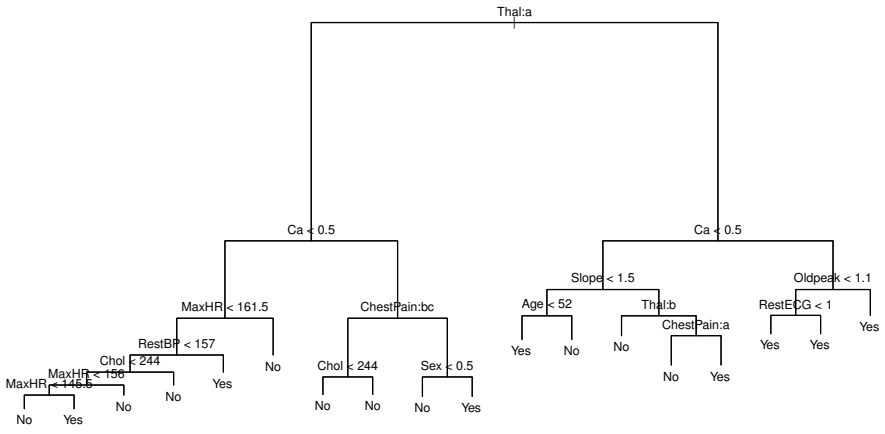
.

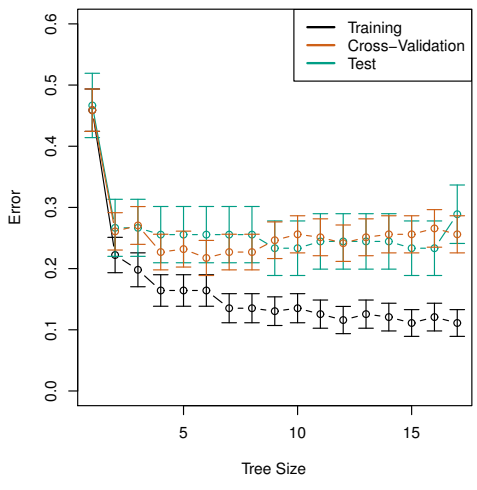
- Any of these three approaches might be used when pruning the tree.
- If R is nearly pure, most of the observations are from one class, then the Gini-index and cross-entropy would take smaller values than classification error rate.

$$\hat{p}_1 = [0.5, 0.25, 0.25] \Rightarrow E = 0.5, G = 0.625, D = 1.0397$$

$$\hat{p}_2 = [0.5, 0.4, 0.1] \Rightarrow E = 0.5, G = \mathbf{0.580}, D = \mathbf{0.9433}$$

- Gini-index and cross-entropy are more sensitive to node purity.
- To evaluate the quality of a particular split, the Gini-index and cross-entropy are more popularly used as error measurement criteria than classification error rate. (E can't distinguish $\hat{p}_1$ and $\hat{p}_2$ above, while G/D are more informative for $\hat{p}_2$)
- The classification error rate is preferable if only prediction accuracy of the final pruned tree is the goal.


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
Age < 52
Thal:b
RestECG < 1
ChestPain:a
RestBP < 157
Chol < 244
Chol < 244
Sex < 0.5
MaxHR < 156
MaxHR < 145.5
No
Yes


Training
Cross−Validation
Test
Error
Tree Size
0.0
0.1
0.2
0.3
0.4
0.5
0.6
5
10
15

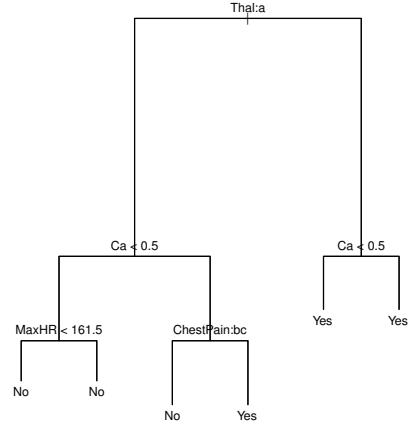

Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
No
Yes

Figure 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.

Trees vs. Linear models

- For regression model:

$$Y = f(X) + \epsilon$$

- Linear model assumes

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

- Regression trees assume

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

where $R_1, ..., R_M$ are rectagular partitions of the input space.

- If the underlying realation is close to linear, linear model is better. Otherwise, regression trees are generally better. (Useless comments)

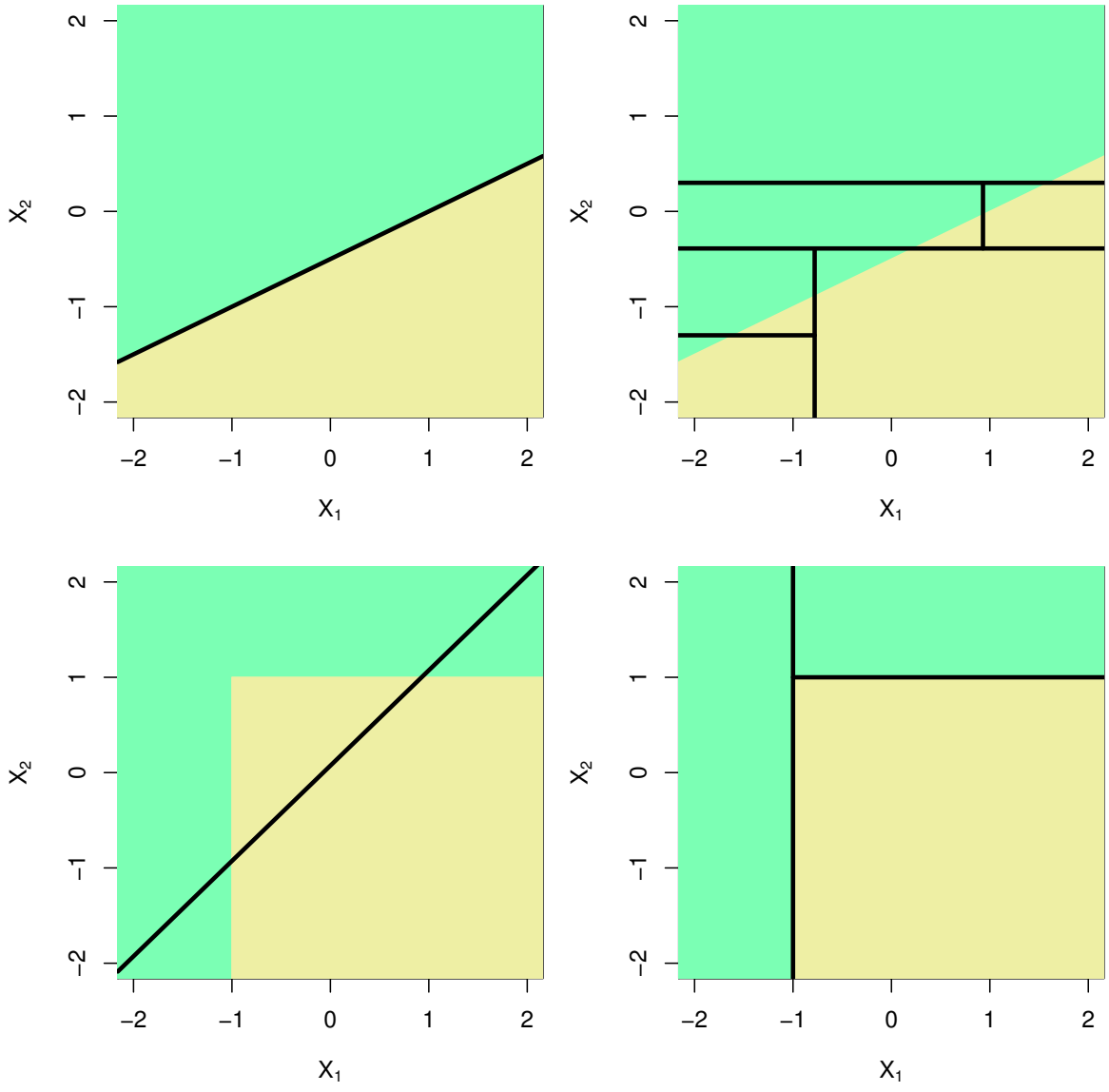

X2
X1
−2
−1
0
1
2

Figure 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 of Trees

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

Disadvantages of Trees

- Trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.
- Trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.
- However, by aggregating many decision trees, using methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved. We introduce these concepts in the next section.

Outline

Bagging (Boostrap Aggregating)

- A general purpose procedure to reduce variance of a learning method.
- A model averaging technique.
- Decision tree is generally a high variance method. (Apply the method based on different data based on same sampling scheme would lead to very different result.)
- Average of iid random variables would have a reduced variance σ^2/n

Statistical Estimates

- Model
$$y_i = f(x_i) + \epsilon_i, \quad i = 1, ..., n.$$
- Suppose a statistical learning method gives $\hat{f}(\cdot)$ based on the training data $(y_i, x_i), i =, 1..., n$.
- For example,
  1. Linear model: $\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}^T x_i$
  2. KNN: $\hat{f}(x) = \sum_{j=1}^{J} \bar{y}_{\tilde{R}_j}$ with least distance to K-cluster partition.
  3. Decision tree: $\hat{f}(x) = \sum_{j=1}^{J} \bar{y}_{R_j}$ with rectangular partition.
  4. ...

The procedure of Bagging

- Data $(y_i, x_i), i = 1, ..., n$; and a learning method $\hat{f}$
- Draw a bootstrap sample from the data, and compute a $\hat{f}_1^*$ based on this set of bootstrap sample.
- Draw another bootstrap sample from the data, and compute a $\hat{f}_2^*$ based on this set of bootstrap sample.
- ....
- Repeat B times, obtain $\hat{f}_1^*, ..., \hat{f}_B^*$.
- Produce the learning method with bagging as

$$\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b^*$$

Summary of the Bagging

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

Out-of-Bag (OOB) error estimation

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

Out-of-Bag (OOB) error estimation

- The OOB MSE is

$$\sum_{i=1}^{n}(y_i - \hat{f}^*_{-i}(x_i))^2$$

- The OOB classification error is

$$\sum_{i=1}^{n} I(y_i \notin \hat{f}^*_{-i}(x_i))$$

- The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation.
- It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.

Variable importance measures

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

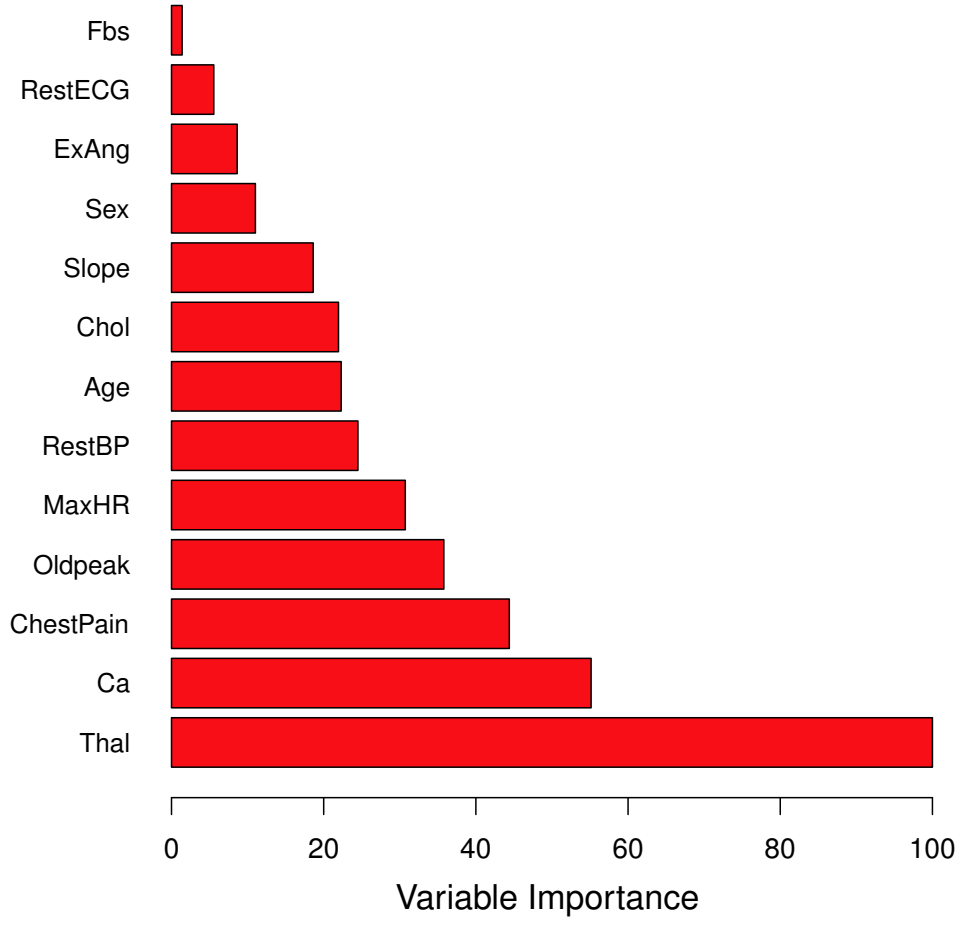


Figure: 9. A variable importance plot for the Heart data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.

Motivation of Random Forest

- An average of B i.i.d random variables, each with variance σ^2, has variance $\frac{1}{B}\sigma^2$.
- What if not independent but correlated?
- If the variables are simply i.d. (identically distributed but not necessarily independent) with positive pairwise correlation ρ, the variance of the average is

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

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

Random Forest

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

Random Forest

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

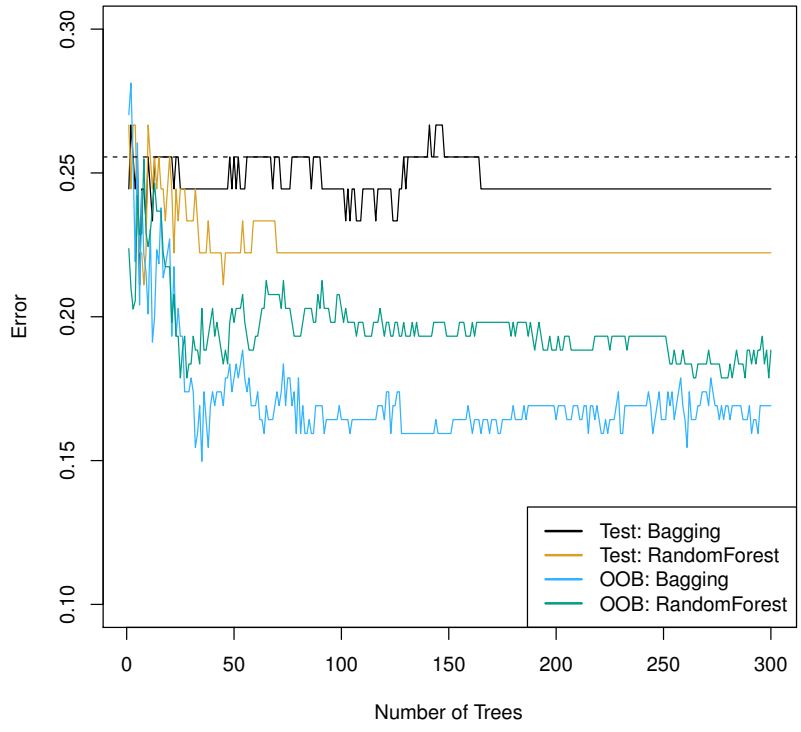


Figure: 8. Bagging and random forest results for the Heart data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower

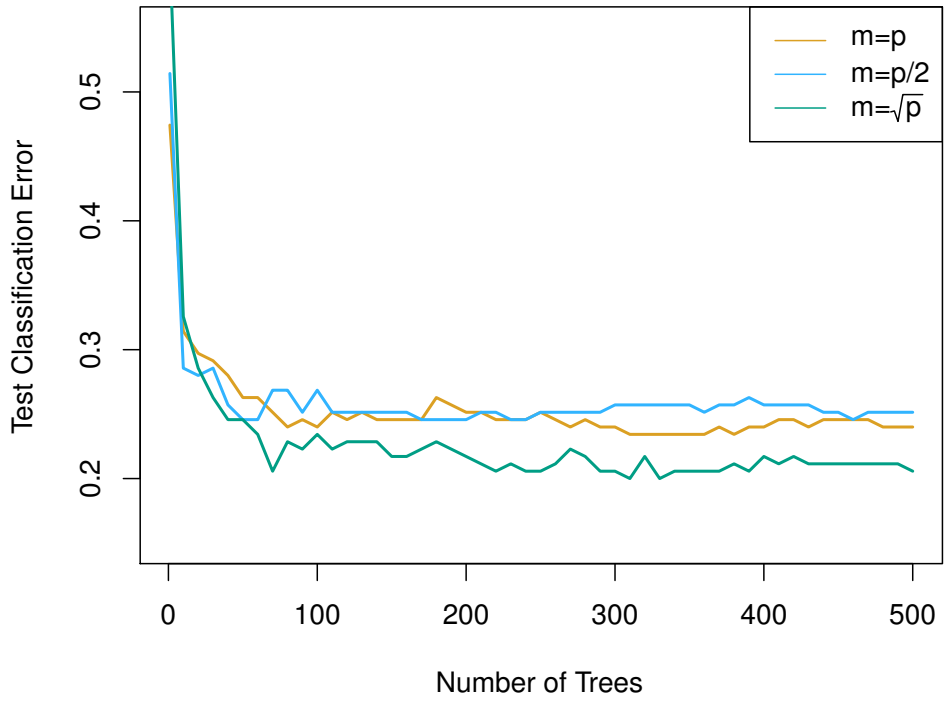


Figure: 10. Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node. Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Boosting

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

Adaboost (Yoav Freund, Robert E. Schapire (1996))

- 1. Initialize the data weights $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
- 2. For $m = 1, \ldots, M$:
  - (a) Fit a classifier $f_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} \mathbb{I}(f_m(\mathbf{x}_n) \neq y_n), \tag{1}$$

where $\mathbb{I}(f_m(\mathbf{x}_n) \neq y_n)$ is the indicator function and equals 1 when $f_m(\mathbf{x}_n) \neq y_n$ and 0 otherwise.

Adaboost

- 2. (Continued)
  - (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} \mathbb{I}(f_m(\mathbf{x}_n) \neq y_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{2}$$

  and then use these to evaluate

$$\alpha_m = \log \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\} \tag{3}$$

  - (c) Update the data weights

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m \mathbb{I}(f_m(\mathbf{x}_n) \neq y_n)\} \tag{4}$$

- 3. Make prediction by

$$F_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}) \right). \tag{5}$$

Illustration of Adaboost

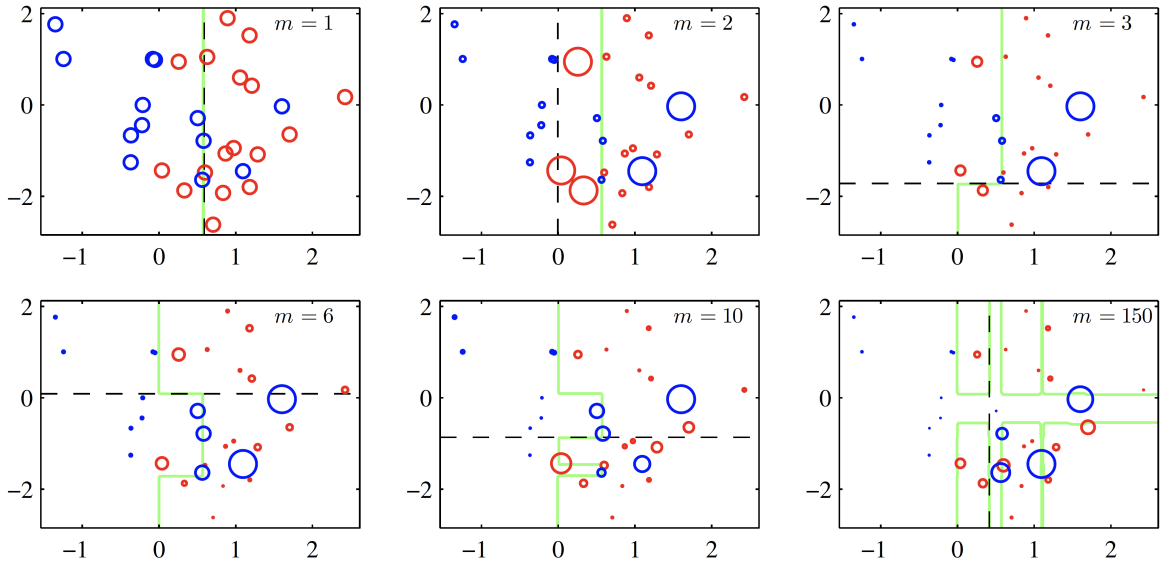


Figure: Each figure shows the number m of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the $m = 1$ base learner are given greater weight when training the $m = 2$ base learner. Bishop (2006).

Statistical view of Adaboost

- Consider the exponential error function

$$E = \sum_{n=1}^{N} \exp\{-y_n F_M(\mathbf{x}_n)\}, \tag{6}$$

  where $F_M(\mathbf{x}) = \frac{1}{2}\sum_{m=1}^{M} \alpha_m f_m(\mathbf{x})$ and $y_n \in \{-1, 1\}$.
- Our goal is to minimize E w.r.t. both α_m and $f_m(\mathbf{x})$.
- Instead of doing a global error function miminization, we shall suppose that the base classifiers $f_1(\mathbf{x}), \ldots, f_{m-1}(\mathbf{x})$ are fixed, as are their coefficients $\alpha_1, \ldots, \alpha_{m-1}$, and so we are minimizing only w.r.t. α_m and $f_m(\mathbf{x})$.

$$\begin{aligned} E &= \sum_{n=1}^{N} \exp\left\{-y_n F_{m-1}(\mathbf{x}_n) - \frac{1}{2} y_n \alpha_m f_m(\mathbf{x}_n)\right\} \\ &= \sum_{n=1}^{N} w_n^{(m)} \exp\{-\frac{1}{2} y_n \alpha_m f_m(\mathbf{x}_n)\} \end{aligned} \tag{7}$$

  where $w_n^{(m)} = \exp\{-y_n F_{m-1}(\mathbf{x}_n)\}$ can be viewed as constants.

- Denote $\mathcal{T}_m$ and $\mathcal{M}_m$ as the sets of correctly classified data points by $f_m(\mathbf{x})$ and the misclassified points, respectively.
- We have

$$E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{M}_m} w_n^{(m)}$$
$$= (e^{\alpha_m/2} - e^{-\alpha_m/2}) \sum_{n-1}^{N} w_n^{(m)} \mathcal{I}(f_m(\mathbf{x}_n) \neq y_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}. \tag{8}$$

  - When we minimize this w.r.t. $f_m(\mathbf{x})$, the second term is constant, and up to a multiplicative constant minimizing the first term is equivalent to minimizing (1).
  - Similarly, minimizing w.r.t. α_m, we obtain (3) in which ϵ_m is defined by (2).

- From (7), we see that, having found α_m and $f_m(\mathbf{x})$, the weights on the data points are updated using

$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{-\frac{1}{2} y_n \alpha_m f_m(\mathbf{x}_n)\right\}.$$

  - Making use of the fact that

$$y_n f_m(\mathbf{x}_n) = 1 - 2\mathbb{I}(f_m(\mathbf{x}_n) \neq y_n),$$

    we see that the weight update $w_n^{(m+1)}$ follows

$$w_n^{(m+1)} = w_n^m \exp(-\alpha_m/2) \exp\{\alpha_m \mathbb{I}(f_m(\mathbf{x}_n) \neq y_n)\}.$$

  - Because the term $\exp(-\alpha_m/2)$ is independent of n, we see that it weights all data points by the same factor and so can be discarded. Thus we obtain (4).
- Finally, once all the base classifiers are trained, new data points are classified by evaluating the **sign** of the combined function. Because the factor of $1/2$ does not affect the sign, this gives (5).

Why exponential loss function?

- We have shown that the Adaboost algorithm minimizes the exponential loss function (6). How to justify the minimization of the exponential loss function?
- Consider the population version of the exponential loss function,

$$\mathbb{E}_{\mathbf{x},y}[\exp\{-yF(\mathbf{x})\}] = \sum_y \int \exp\{-yF(\mathbf{x})p(y|\mathbf{x})p(\mathbf{x})\}d\mathbf{x}$$

- Taking the functional derivative w.r.t. $F(\mathbf{x})$, we get

$$\frac{\partial}{\partial F(\mathbf{x})}\mathbb{E}_{\mathbf{x},y}[\exp\{-yF(\mathbf{x})\}] = -\sum_y y\exp\{-yF(\mathbf{x})\}p(y|\mathbf{x})p(\mathbf{x})$$

$$= \{\exp\{F(\mathbf{x})\}p(y=-1|\mathbf{x}) - \exp\{-F(\mathbf{x})\}p(y=1|\mathbf{x})\}p(\mathbf{x}).$$

- Setting this equal to zero and rearranging, we obtain

$$F(\mathbf{x}) = \frac{1}{2}\log\left\{\frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})}\right\},$$

whose sign is the Bayes optimal classification.

Convex Losses

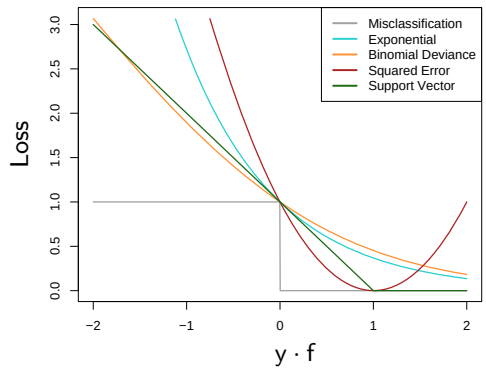


FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:* $(y - f)^2$*; and support vector:* $(1 - yf)_+$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$*.*

Friedman's Gradient Boost algorithm

- Initialize $\hat{F}_0(\mathbf{x})$ to be a constant, $\hat{F}_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^N L(y_i, \rho)$.
- For m in $1, \ldots, M$ do
  1. Compute the negative gradient as the working response

$$r_i = -\frac{\partial}{\partial F_{m-1}(\mathbf{x}_i)} L(y_i, F_{m-1}(\mathbf{x}_i)) \Big|_{F_{m-1}(\mathbf{x}_i)=\hat{F}_{m-1}(\mathbf{x}_i)} \tag{9}$$

  2. Fit a regression model, $\hat{g}(\mathbf{x})$, predicting r_i from the covariates $\mathbf{x}_i$.
  3. Choose a gradient descent step size as

$$\hat{\lambda} = \arg\min_{\lambda \in R} \sum_{i=1}^{N} L(y_i, \hat{F}_{m-1}(\mathbf{x}_i) + \lambda \hat{g}(\mathbf{x}_i)) \tag{10}$$

  4. Update the estimate of $F(\mathbf{x})$ as

$$\hat{F}_m(\mathbf{x}) = \hat{F}_{m-1}(\mathbf{x}) + \hat{\lambda}\hat{g}(\mathbf{x}) \tag{11}$$

Algorithm for tree boosting in regression

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

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_b(x)$$

  3. Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}_b(x_i) = y_i - \hat{f}(x_i).$$

- 3. Output the boosted model $\hat{f}$. In fact,

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

Tuning parameters for boosting trees

- In regression, square loss minimization over $\hat{f}_t$ is obtained by fitting the residue r_i
- There are three hyper-parameters:
  - The number of trees M. Large M leads to overfit. (not a tuning parameter for bagging)
  - The learning rate λ.
  - The number d in splits in each tree (the size of each tree). Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split

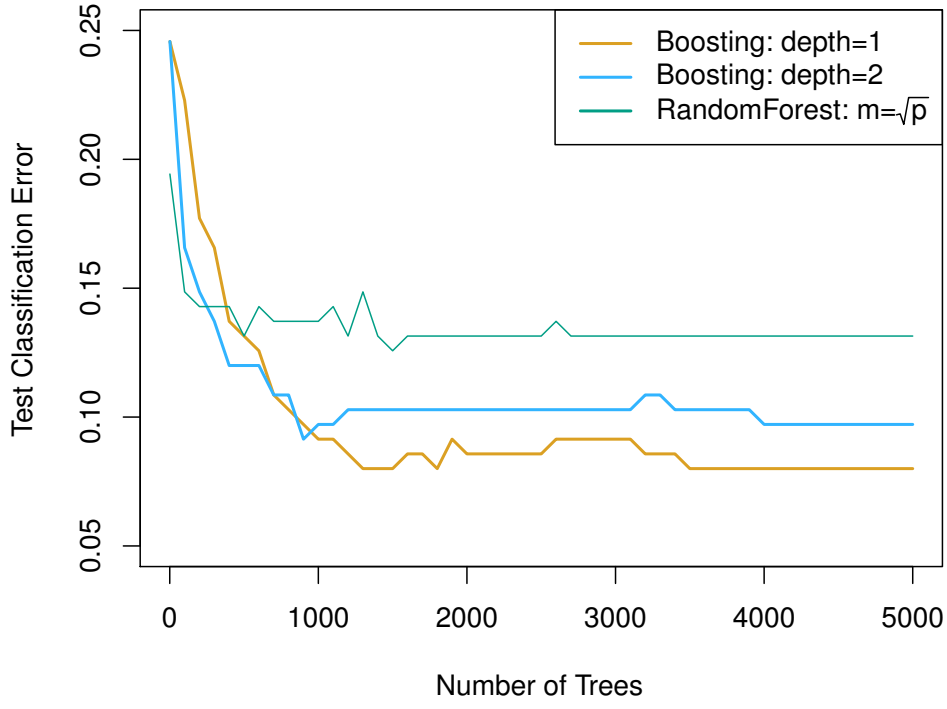


Figure: 8.11. Results from performing boosting and random forests on the 15-class gene expression data set in order to predict cancer versus normal. The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant. The test error rate for a single tree is 24%.

Forward Stagewise Regression as Coordinate Descent

Consider a set of fixed basis functions $\{T_k\}_{k=1,\ldots,K}$.

- Initialze $\hat{\alpha}_k = 0, k = 1, \ldots, K$. Set $\epsilon > 0$ to some small constant, and M large.
- For $m = 1$ to M:
  - $(\beta^*, k^*) = \arg\min_{\beta,k} \sum_{i=1}^{N} \ell\left(y_i - \sum_{l=1}^{K} \hat{\alpha}_l T_l(\mathbf{x}_i) - \beta T_k(\mathbf{x}_i)\right)$
  - $\hat{\alpha}_{k^*} \leftarrow \hat{\alpha}_{k^*} + \epsilon \cdot \mathrm{sign}(\beta^*)$.
- Output $F_M(\mathbf{x}) = \sum_{k=1}^{K} \hat{\alpha}_k T_k(\mathbf{x})$.

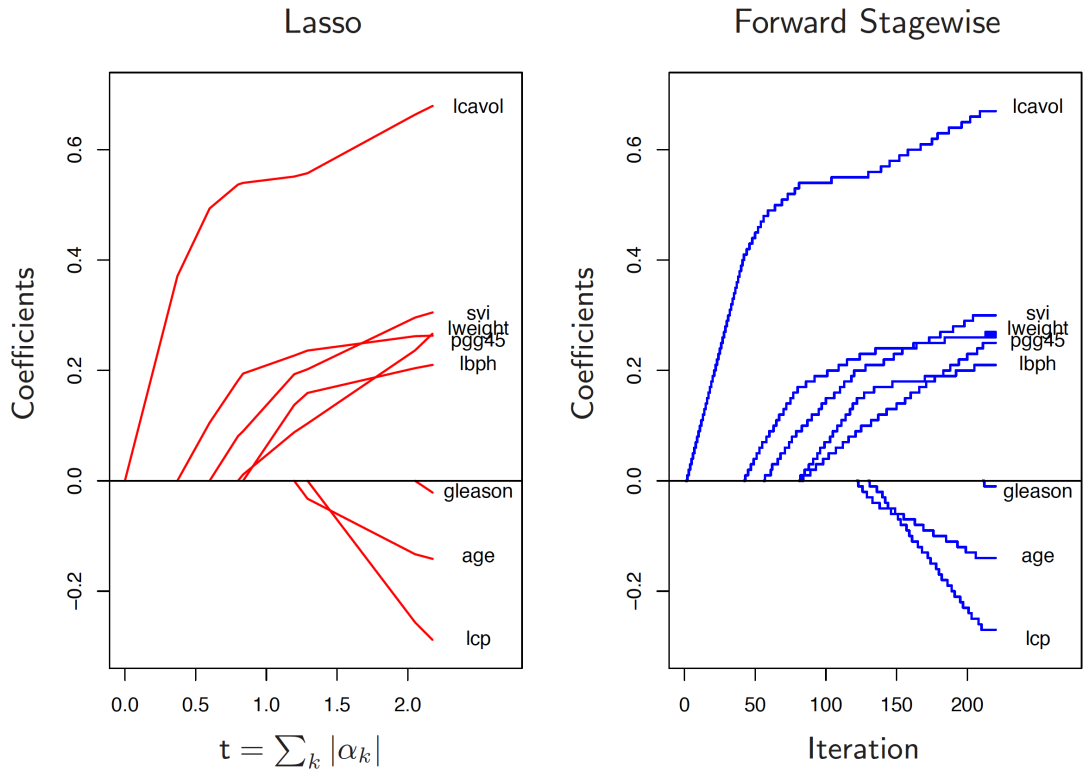


Figure: Profiles of estimated coefficients from linear regression, for the prostate data. The left panel shows the results from the Lasso, for different values of the bound parameter $t = \sum_k |\alpha_k|$. The right panel shows the results of the forward stagewise linear regression, using $M = 220$ consecutive steps of size $\epsilon = 0.01$.

Summary: Statistical view of boosting methods

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

- they fit by functional gradient descent or coordinate descent an additive model in a flexible set of basis functions.
- they use a suitable loss function for the fitting process.
- they regularize by forward stagewise fitting with shrinkage (small step-size in coordinate descent), this mimics an ℓ_1 (Lasso) penalty on the weights.