Machine Learning for OR & FE

Classification and Regression Trees with Bagging, Random Forests & Boosting

Martin Haugh

Department of Industrial Engineering and Operations Research
Columbia University
Email: `martin.b.haugh@gmail.com`

Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani (JWHT). Additional References: Chapter 8 of JWHT

Outline

Regression Trees

Regression problem: Learn the mapping $\rho(\mathbf{x}) : \mathcal{X} \subset \mathbb{R}^m \mapsto \mathbb{R}$.

Can formulate problem as one of learning a **partition** $\mathcal{X} = \cup_{j=1}^{J} \mathcal{X}_j$, i.e. $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$ for $i \neq j$, with

$$\rho(\mathbf{x}) = \sum_{j=1}^{J} \rho_j 1_{\{\mathbf{x} \in \mathcal{X}_j\}}.$$

and each $\rho_j \in \mathbb{R}$.

But searching over all possible partitions of $\mathcal{X}$ is intractable.

- So we impose some **structure** and restrict $\mathcal{X}_k$'s to be axis-aligned rectangles.

e.g. Use regression tree to predict baseball players' log salaries.

So a player with more than 4.5 years experience in major leagues and more than 117 hits in previous season has an estimated salary of

$$1,000 \times e^{6.74} \approx \$845,000$$

e.g. Estimating Baseball Players' Salaries

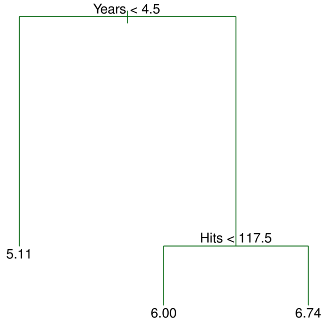


Figure 8.1 from ISLR: For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$)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

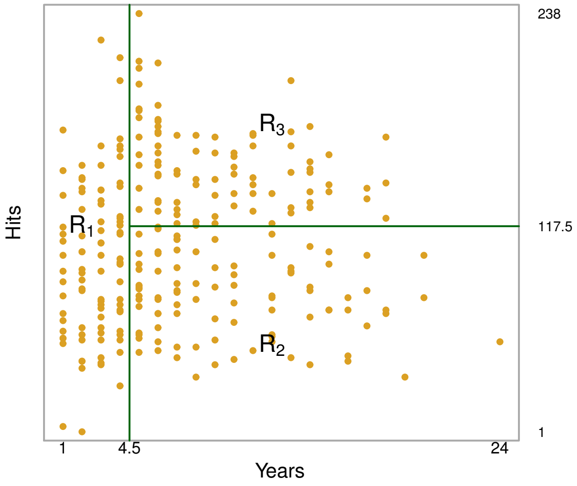


Figure 8.2 from ISLR: The three-region partition for the Hitters data set from the regression tree illustrated in Figure 8.1.

Regression Trees

But how to find a good partition of axis-aligned rectangles? This is hard!

So we use a **top-down greedy** approach known as recursive binary splitting.

But first need a **loss function**. For regression typically use RSS so goal is to find rectangles $R_1, \ldots, R_J$ to minimize

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

where $\hat{y}_{R_j}$ = mean response for training observations in the j^{th} rectangle.

Recursive binary splitting begins by selecting predictor x_j and cutpoint s to minimize

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

where

$$R_1(j,s) := \{\mathbf{x} \,:\, x_j < s\} \quad \text{and} \quad R_2(j,s) := \{\mathbf{x} \,:\, x_j \geq s\}$$

Regression Trees

Can find (j, s) quickly when number of predictors p not too large.

Next step is to choose a predictor and cutpoint to split one of the rectangles from the previous step, i.e. R_1 or R_2, again with the goal of minimizing the RSS.

- have 3 rectangles at this point.

Continue in this manner until some stopping criterion is reached.

Note that we have J rectangles after J steps of the algorithm.

Once tree has been constructed can predict response of a new test observation $\mathbf{x}_{new}$ by:

- determining rectangle that contains $\mathbf{x}_{new}$
- and using mean response of training observations in that rectangle.

Next slide displays a tree with 5 rectangles.

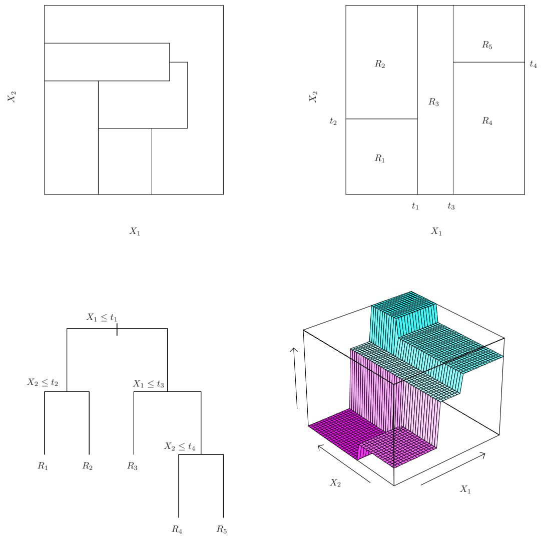


Figure 8.3 from ISLR: Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting. Top Right: The output of recursive binary splitting on a two-dimensional example. Bottom Left: A tree corresponding to the partition in the top right panel. Bottom Right: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

Pruning a Regression Tree

A common strategy is to:

1. Grow a large tree, T_0, until a stopping criterion is satisfied
2. Then prune the tree using regularization
    - done to control over-fitting.

Prune using cost complexity pruning – also known as weakest link pruning:

- Consider a sequence of subtrees indexed by a tuning parameter $\alpha \geq 0$.
- Each value of α corresponds to a subtree $T \subseteq T_0$ that minimizes

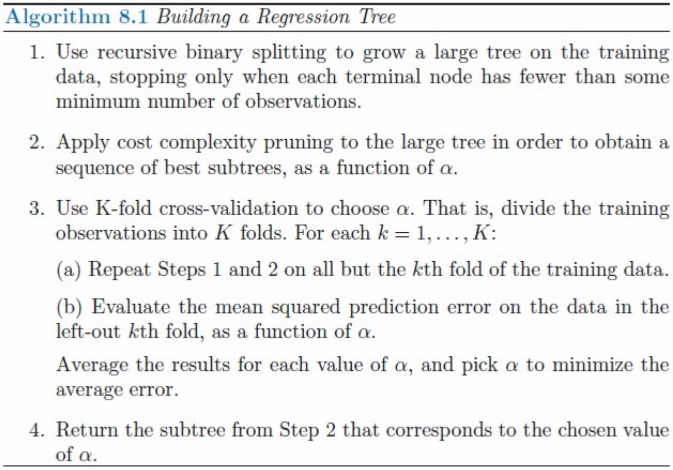
$$\sum_{m=1}^{|T|} \sum_{\mathbf{x}_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \tag{1}$$

where $|T| = \#$ of **terminal** nodes in T and R_m is the rectangle corresponding to m^{th} terminal node.

Fact: As we increase α from 0, branches get pruned from T_0 in a **nested** fashion

- so easy to get sequence of subtrees beginning with T_0 and ending with a single node tree with a single rectangle $= \mathcal{X}$.

Complete Algorithm for Training a Regression 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.

Algorithm 8.1 from ISLR

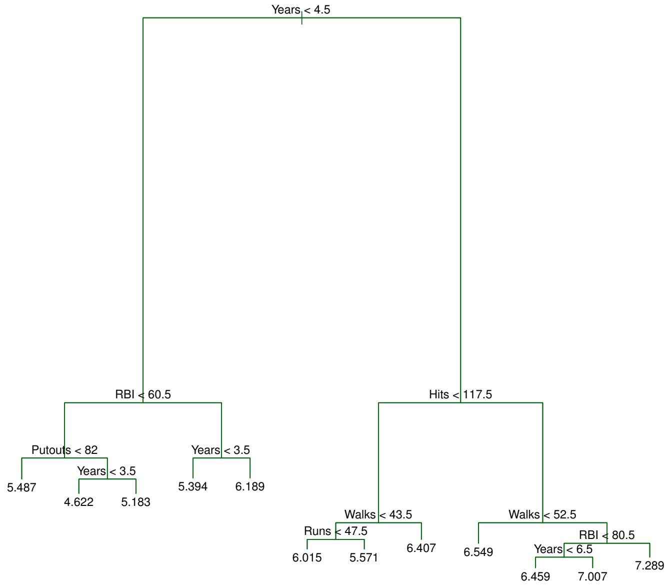


Figure 8.4 from ISLR: Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown.

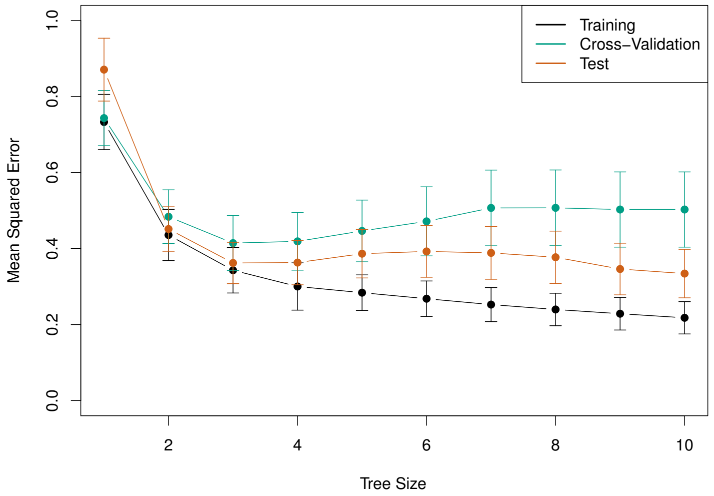


Figure 8.5 from ISLR: Regression tree analysis for the Hitters data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three (which is displayed in Figure 8.1).

Classification Trees

A classification tree is very similar to a regression tree except instead of predicting the mean response in a given rectangle we predict the most commonly occurring class in that region.

But also interested (why?) in the **class proportions**

$$p_{jk} = \frac{1}{|R_j|} \sum_{i:\mathbf{x}_i \in R_j} 1_{\{y_i = k\}}$$

among the training points in each rectangle.

Several possibilities for defining "error", $e(R_j)$, at a (terminal) "node" R_j of tree:

- Classification error: $E(R_j) := 1 - \max_k \hat{p}_{jk}$
- Cross-entropy : $D(R_j) := -\sum_{k=1}^{K} \hat{p}_{jk} \log(\hat{p}_{jk})$
- Gini index : $G(R_j) := 1 - \sum_{k=1}^{K} \hat{p}_{jk}^2$

Seems natural to use classification error instead of RSS when implementing recursive binary splitting for a classification tree

- but not a good idea since classification error not sufficiently sensitive.

Classification Trees

Cross-entropy and Gini index are both measures of **node purity** since both will have values near 0 if the $\hat{p}_{jk}$'s are all near 0 or 1.

Therefore typically use cross-entropy or Gini index to evaluate quality of a particular split with

$$e(T) := \sum_j |R_j| \, D(R_j)$$

or

$$e(T) := \sum_j |R_j| \, G(R_j)$$

replacing RSS that is used for regression trees.

When pruning the tree could use any of the 3 measures but common to use classification error, $E(R_j)$, if ultimate goal is prediction accuracy.

Question: Why do some splits yield terminal nodes with the **same** predicted value in Figure 8.6 (on next slide)?

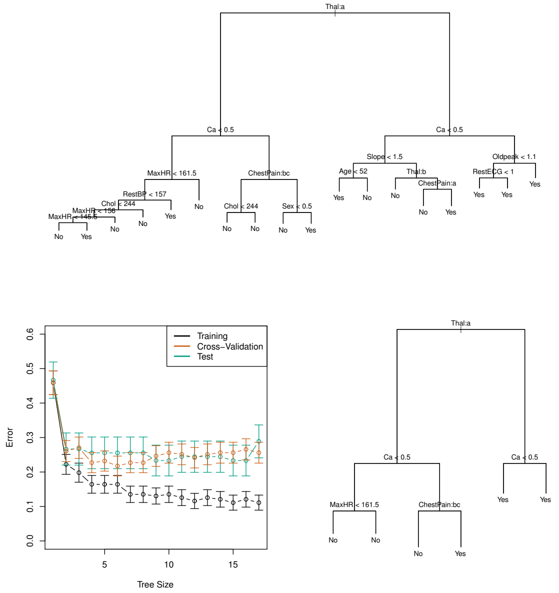


Figure 8.6 from ISLR: Heart data. Top: The unpruned tree. Bottom Left: Cross -validation error, training, and test error, for different sizes of the pruned tree. Bottom Right: The pruned tree corresponding to the minimal cross-validation error.

Regression Trees vs Linear Regression

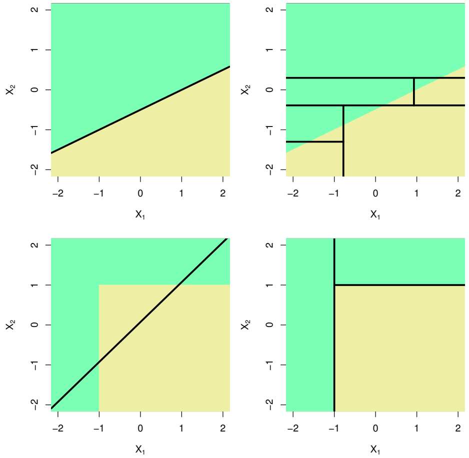


Figure 8.7 from ISLR: Top Row: A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right). 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).

Pros & Cons of Decision Trees

Pros:

- Interpretable and easy to explain
- Robust to outliers, scaling, and irrelevant variables
- Can easily handle categorical variables (without needing dummy variables) and missing data
- Only one tuning parameter

Cons:

- The true map, $\rho^{\text{true}}(\mathbf{x})$, may not be piecewise constant on axis-aligned rectangles
- High variance! Fitted models depends a lot on data split into training and test sets. Greedy algorithm makes the issue worse!
- Often not competitive with other regression and classification approaches

How does one get around the variance?

Improving Decision Trees

Decision trees often combined with other methods to produce superior classifiers including:

- Building and aggregating results of multiple trees using Bootstrap Aggregation i.e., bagging
- Building random forests
- Boosting with tree stumps.

Bagging

Bootstrap **Ag**gregation (Bagging)

- Generate bootstrap samples of the data: $\mathcal{D}_1, \ldots, \mathcal{D}_B$
- Train classification or regression trees $\rho_1, \ldots, \rho_B$
  - trees are grown deep and are not pruned.
- Classify using majority rule or (for regression trees) predict using average of the B predictions!

Can estimate test error of bagged model using out-of-bag (OOB) observations:

- For each of the $i = 1, \ldots, n$ of observations, predict response of i^{th} observation using only those trees for which the i^{th} data-point was OOB.
- This yields $\approx B/3$ (why?) predictions for i^{th} observation.
- Average these predictions for regression or use majority voting of predictions for classification and compare with true response to obtain error on i^{th} observation.
- Average across all n observations to obtain valid estimate of test error.

Improves performance ... but not by much. What could have gone wrong?

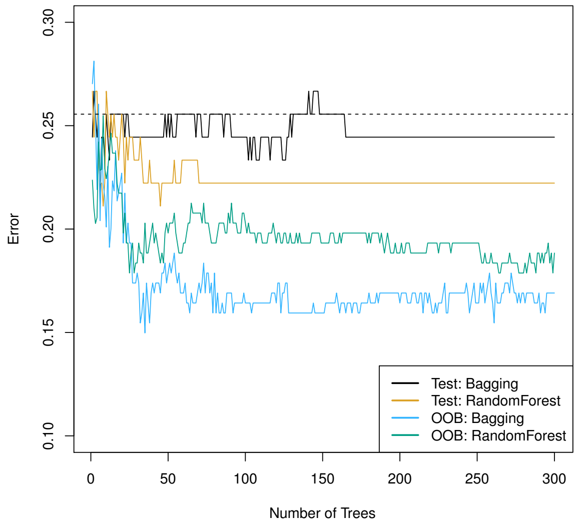


Figure 8.8 from ISLR: Bagging and random forest results for the Heart data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.

Variable Importance Measures

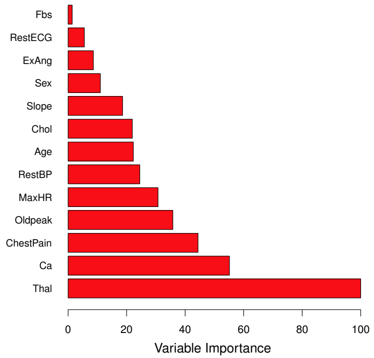


Figure 8.9 from ISLR: A variable importance plot for the Heart data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.

Bagged models clearly hard to interpret but can still obtain measure of overall importance of each predictor:

- For regression, compute total amount that RSS is decreased due to splits on each predictor, averaged over the B trees.
- For classification trees, use decrease in Gini index.

Random Forests

Random forests improve bagged trees by **de-correlating** the trees.

Random forests are constructed according to:

- Generate bootstrap samples of the data: $\mathcal{D}_1, \ldots, \mathcal{D}_B$.
- Build a tree on each bootstrapped data-set but:
  - At each node only a random sample of $m \leq p$ predictors are candidates for splitting
  - A fresh random sample is taken at each split
  - Typically take $m \approx \sqrt{p}$ for classification and $m \approx p/3$ for regression.
- Classify using majority rule or take average for regression!

Question: What does the extra randomness do?

Note bagging is a special case of random forests corresponding to taking $m = p$.

Random forests are very popular and can often lead to dramatic improvement over trees and bagging

- see improvement on gene expression example on next slide!

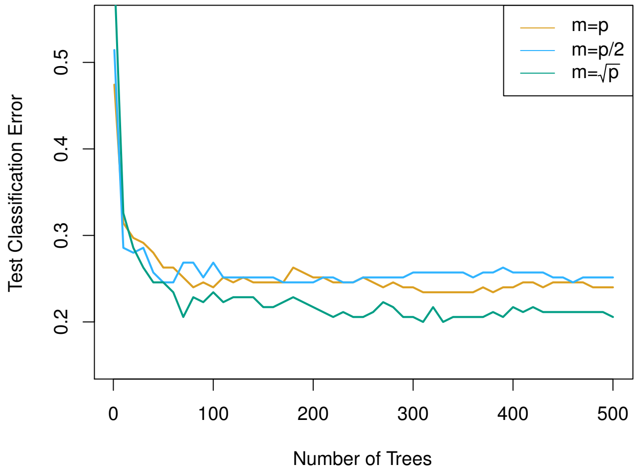


Figure 8.10 from ISLR: Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior 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%.

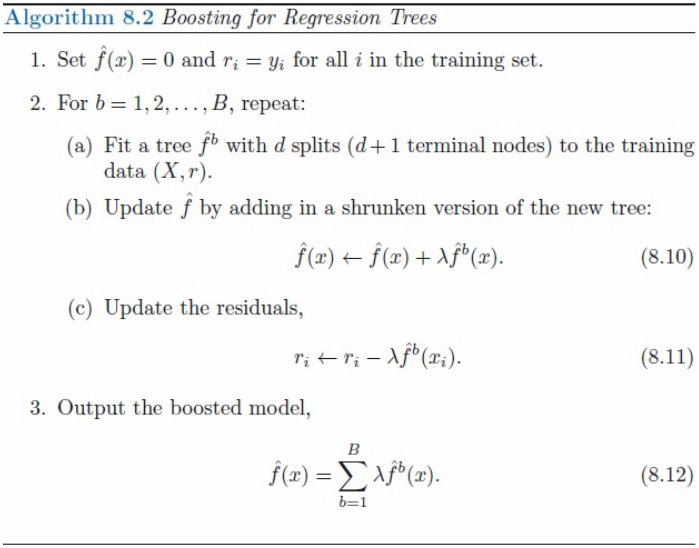
Boosting

Boosting a general approach for improving performance of weak classifiers / regression models.

Boosting decision trees also involved building many decision trees but now each tree is grown **sequentially** based on performance of previous tree.

Boosting does **not** involve bootstrap sampling.

Algorithm for Boosting Regression Trees



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:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \qquad (8.10)$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \qquad (8.11)$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \qquad (8.12)$$

Algorithm 8.2 from ISLR

Algorithm for Boosting Regression Trees

Note construction of each tree depends on the trees that have already been grown.

Boosting has 3 tuning parameters:

1. $B = \#$ of trees. Use cross-validation to select.
2. The shrinkage parameter, λ, controls the learning rate.
    - Typical values are $\lambda = 0.01$ or 0.001 with optimal value depending on problem.
    - Very small λ often results in much larger value of optimal B.
3. $d = \#$ of splits / terminal nodes.
    - $d = 1$ often works well in which case the trees are called stumps.

Boosting can often perform very well – see example using gene expression data to predict cancer on next slide.

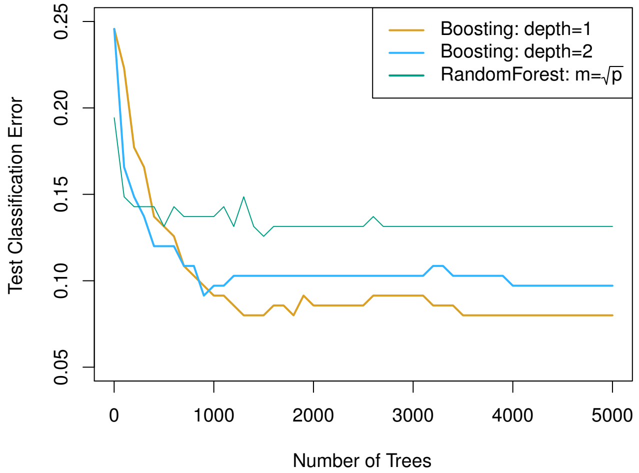


Figure 8.11 from ISLR: Results from performing boosting and random forests on the 15-class gene expression data set in order to predict cancer versus normal. The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform 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%.

Algorithm for Classification Boosting

Initialize the probability $w_i^{(1)}$ of each data point $(y_i, \mathbf{x}_i) = \frac{1}{N}$ and repeat the following for $t = 1, \ldots, T$:

- Fit a classifier $\rho^{(t)}$ using probability weights $w^{(t)}$
- Compute error e_t and update α_t:

$$\begin{aligned} e_t &= \sum_{i=1}^{N} w_i^{(t)} 1_{\{y_i \neq \rho^{(t)}(\mathbf{x}_i)\}} \\ \alpha_t &= \log\left(\frac{1-e_t}{e_t}\right) + \log(K-1) \end{aligned}$$

- Define new probability

$$w_i^{(t+1)} \propto w_i^{(t)} e^{\alpha_t 1_{\{y_i \neq \rho^{(t)}(\mathbf{x}_i)\}}}$$

i.e. **increase weight** on examples where **classifier makes an error**.

Final classifier:

$$\rho(\mathbf{x}) = \underset{1 \leq k \leq K}{\operatorname{argmax}} \left\{ \sum_{t=1}^{T} \alpha_t 1_{\{\rho_t(\mathbf{x}) = k\}} \right\}$$