Tree-Based Methods

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

Tree-based Methods

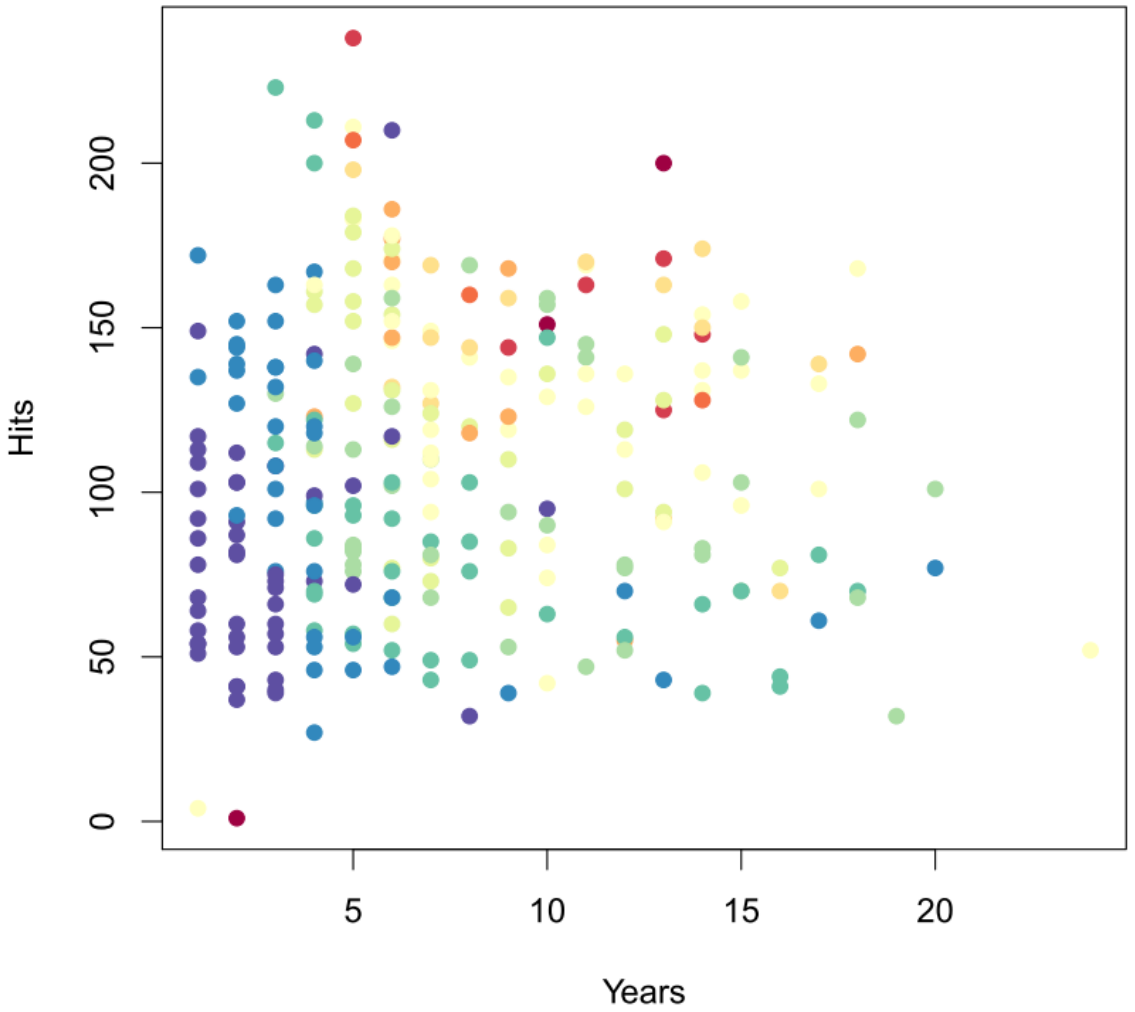
- Here we describe tree-based methods for regression and classification
  - These involve firstly stratifying or segmenting the predictor space into a number of simple regions
  - And use the mean or the mode response value for the training observations in the region to which it belongs for inference
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods

Pros and Cons

- Tree-based methods are simple and useful for interpretation
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy
- Hence we also discuss bagging, random forests, boosting and Bayesian additive regression trees. These methods grow multiple trees which are then combined to yield a single consensus prediction
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation

Baseball salary data: how would you stratify it?

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



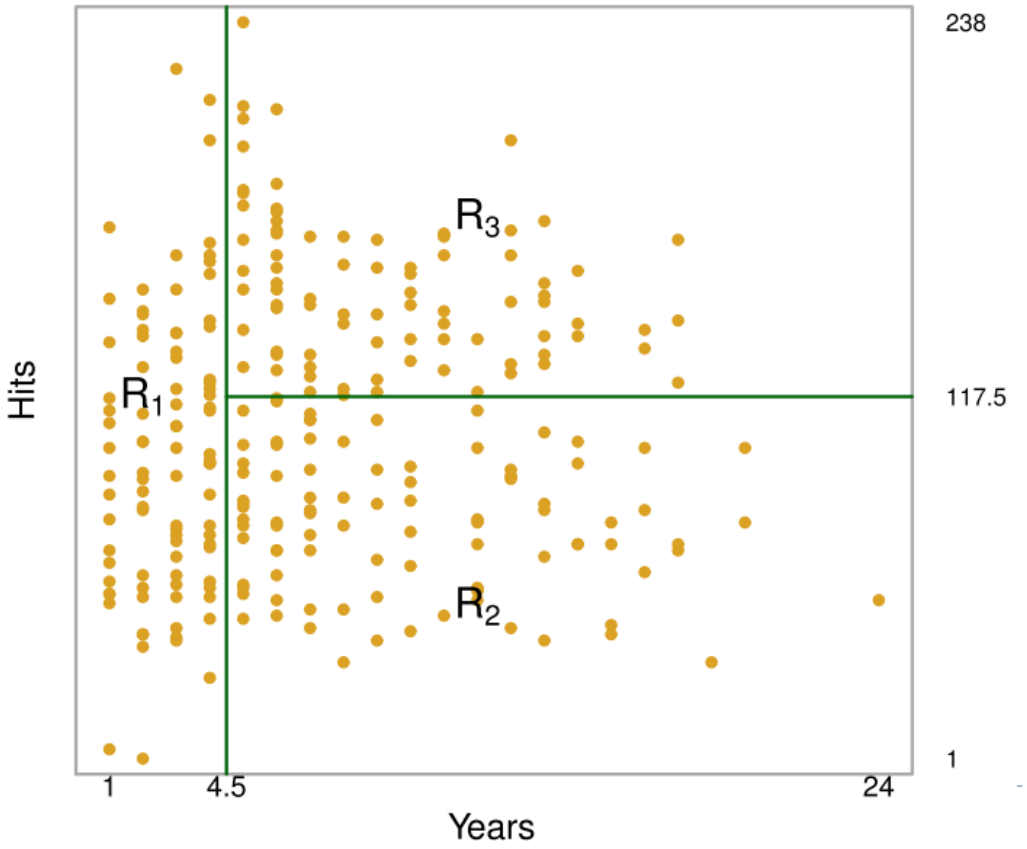
Decision tree for these data

Details of previous figure

- For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to Years< 4.5, and the right-hand branch corresponds to Years ≥ 4.5
- The tree has two internal nodes and three *terminal nodes*, or *leaves*. The number in each leaf is the *mean* of the response for the observations that fall there

Results

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



Terminology for Trees

- In keeping with the tree analogy, the regions R_1, R_2, and R_3 are known as terminal nodes
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree
- The points along the tree where the predictor space is split are referred to as internal nodes
- In the hitters tree, the two internal nodes are indicated by the text $Years < 4.5$ and $Hits < 117.5$

Interpretation of Results

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

Details of the tree-building process

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$— into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$
2. For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j

More details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model
- The goal is to find boxes $R_1, R_2, \ldots, R_J$ that minimize the RSS, given by

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

$\hat{y}_{R_j}$ 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

Details— Continued

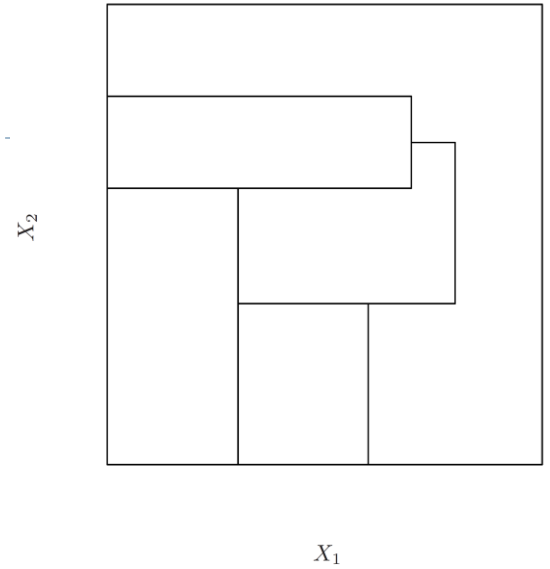
- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS (choosing j and s to minimize)

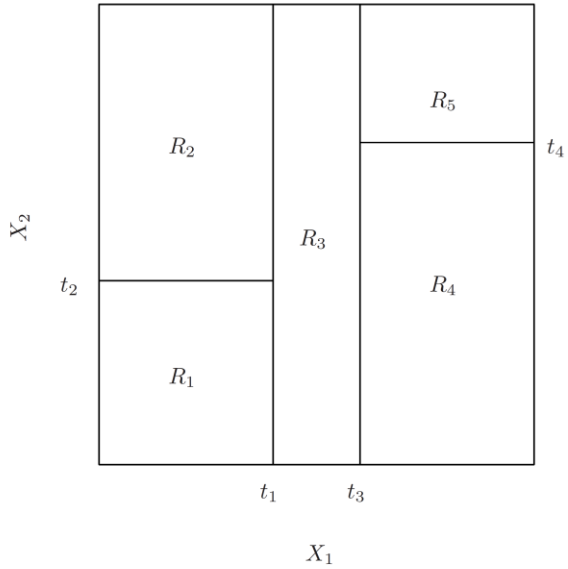
$$\sum_{i:x_i \epsilon R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \epsilon R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

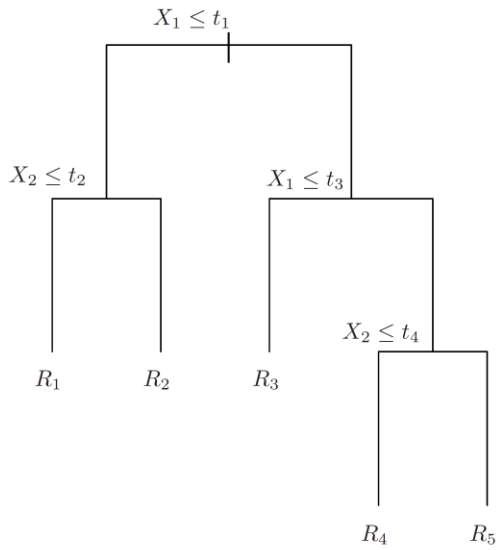
- Next, we looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions
  - Instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions
  - Again, 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

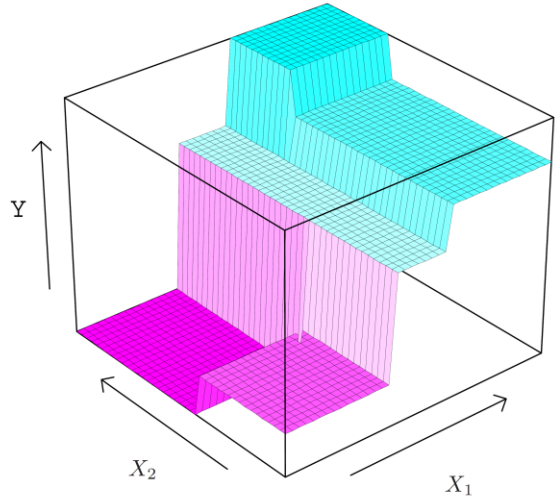
Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs
- A five-region example of this approach is shown









Details of previous figure

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

Regulization

- 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
- A simple way to limit a tree's size is to directly regulate its depth, the size of its terminal nodes (training observation belongs to them), or both

Pruning a tree

- A smaller tree with fewer splits (that is, fewer regions $R_1, R_2, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias
  - One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold
  - This strategy will result in smaller trees, but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on

Pruning a tree— continued

- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree
- Cost complexity pruning — also known as weakest link pruning — is used to do this
- We consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

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

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
- Note that similar formulation was used in order to control the complexity of a linear model when we discuss lasso
- It turns out that as we increase α from zero in (8.4), branches get pruned from the tree in a nested and predictable fashion!
  - We select an optimal value $\hat{\alpha}$ using cross-validation
  - We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$

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.

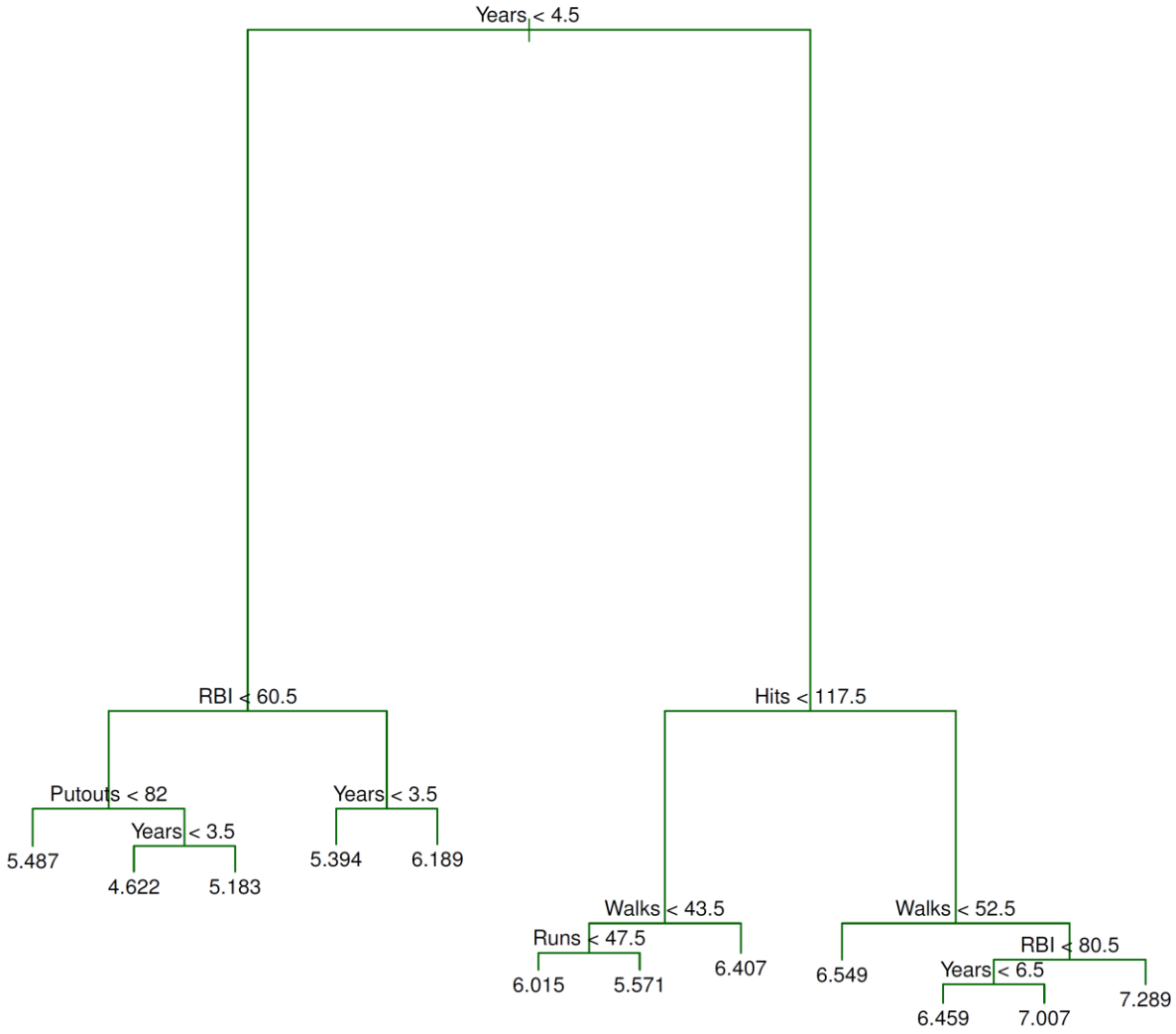
   Average the results for each value of α, and pick α to minimize the average error.

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

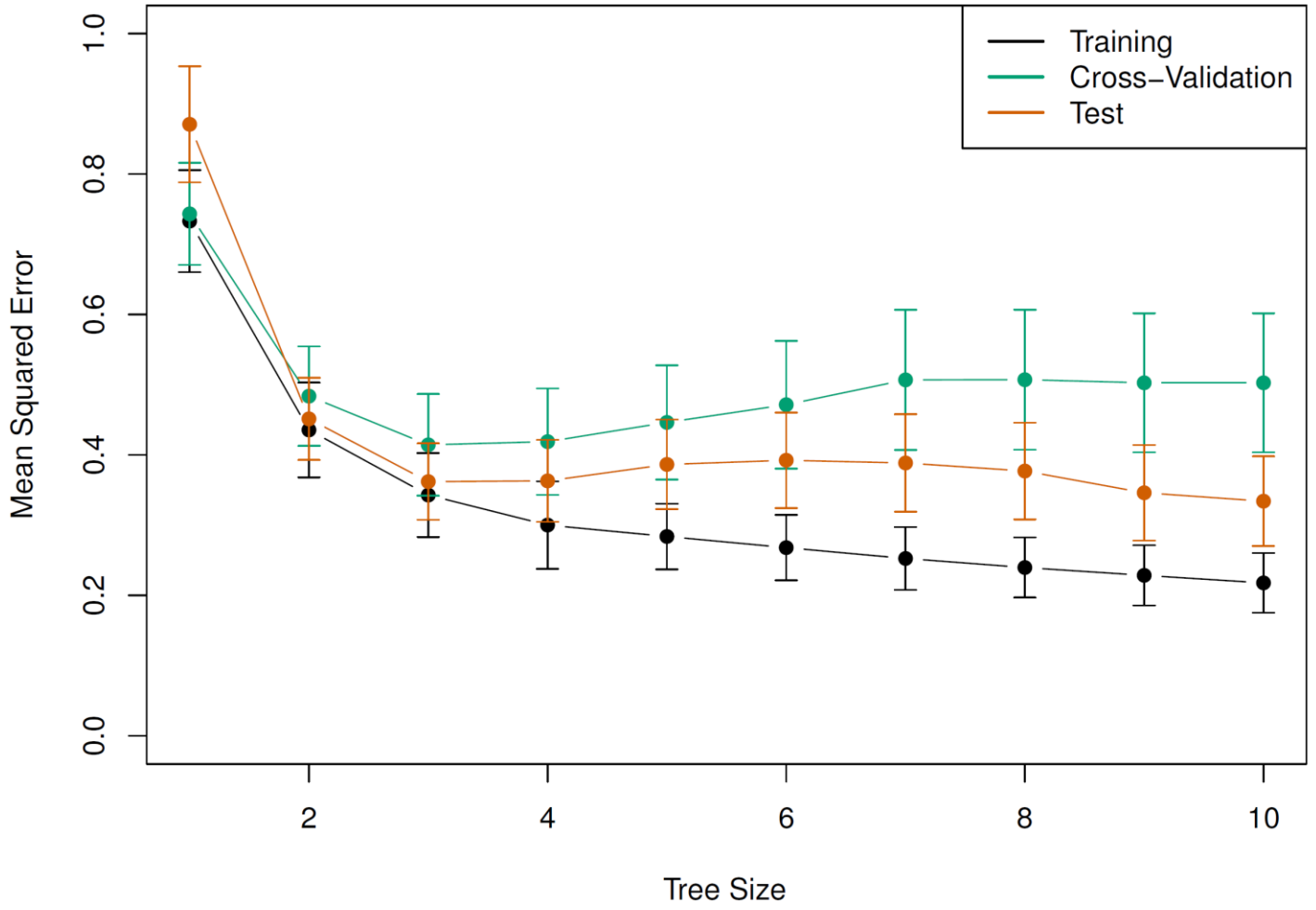
Baseball example continued

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

Baseball example continued



Baseball example continued



Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs
- 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

Details of classification trees

- A natural alternative to RSS is the classification error rate. In our case, this is simply the fraction of the training observations in that region that do not belong to the most common class

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

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class

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

Gini index and Deviance

- The Gini index is defined by

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

  - A measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one
  - For this reason the Gini index is referred to as a measure of node purity — a small value indicates that a node contains predominantly observations from a single class

- An alternative to the Gini index is cross-entropy, given by

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

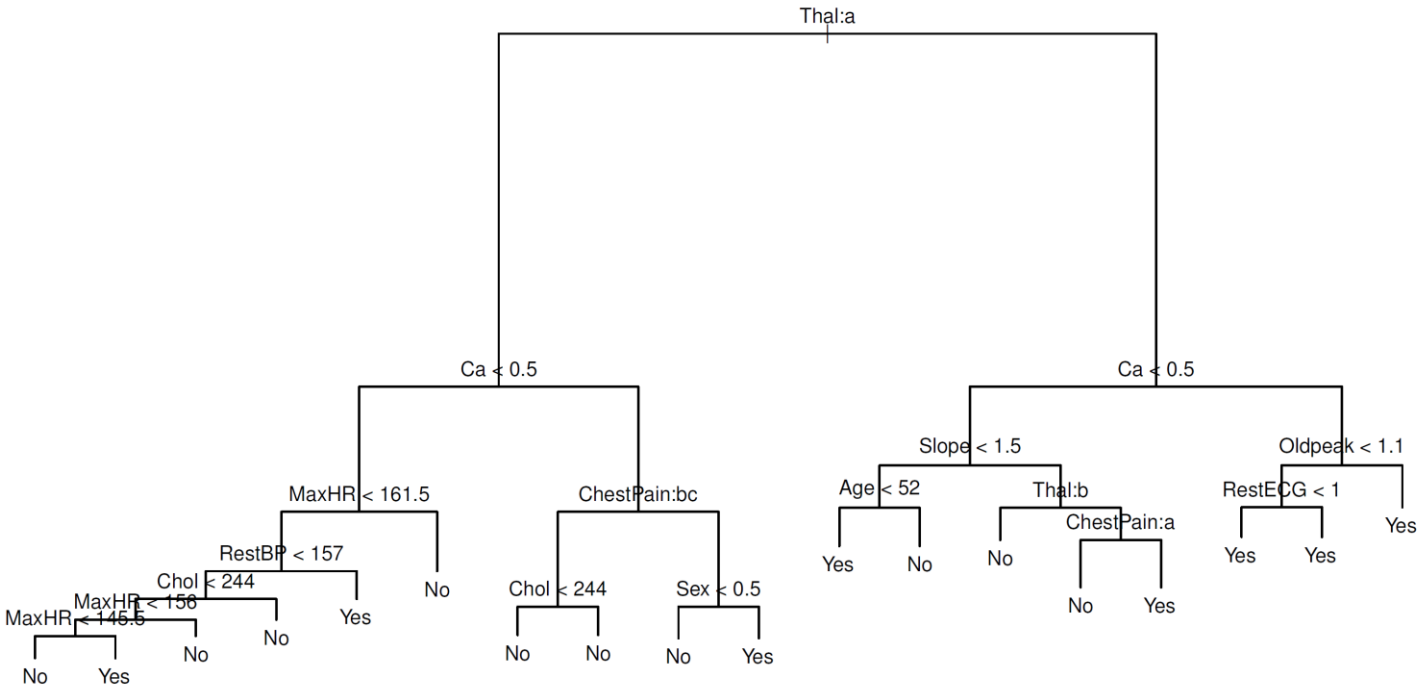
- It turns out that the Gini index and the cross-entropy are very similar numerically (differentiable)

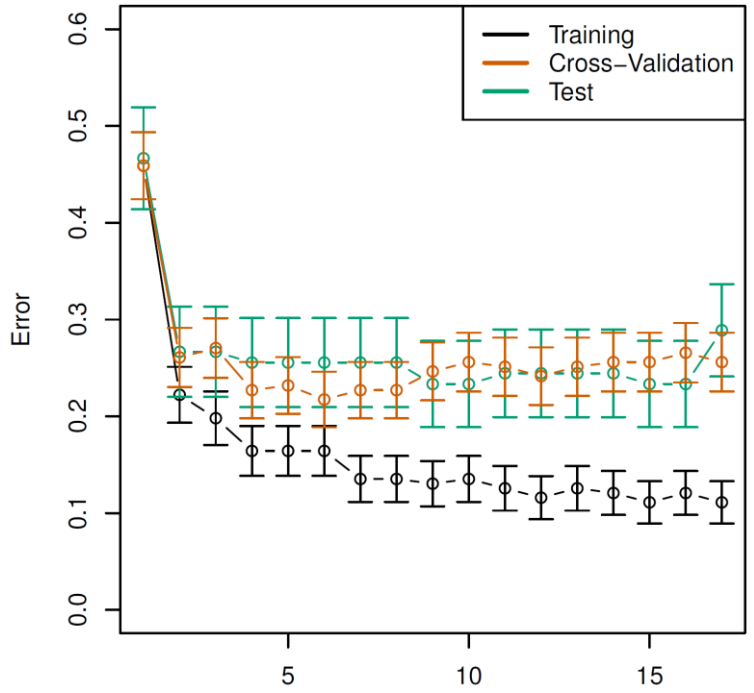
Example: heart data

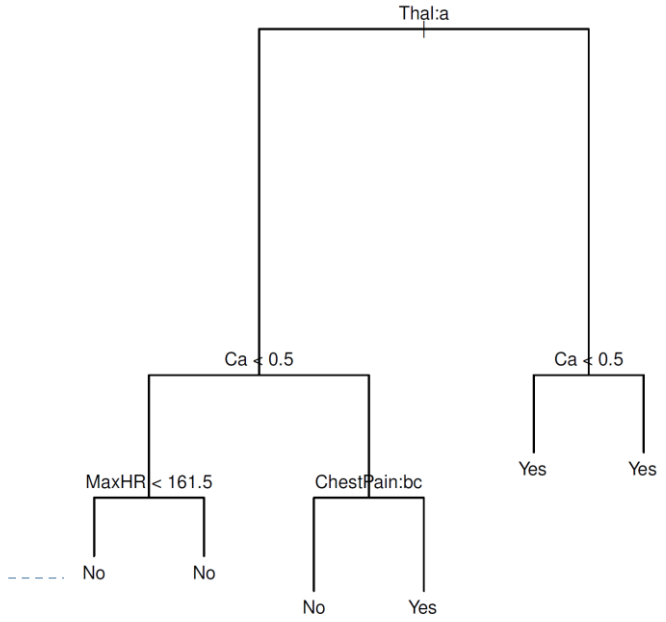
- These data contain a binary outcome HD for 303 patients who presented with chest pain
- An outcome value of Yes indicates the presence of heart disease based on an angiographic test, while No means no heart disease
- There are 13 predictors including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements
- Cross-validation yields a tree with six terminal nodes. See next figure

- There are some qualitative predictors
- Some of the splits yield two terminal nodes that have the same predicted value
  - Though the split $RestECG < 1$ does not reduce the classification error, it improves the Gini index and the entropy, which are more sensitive to node purity

(weighted by the observation in each subtree)





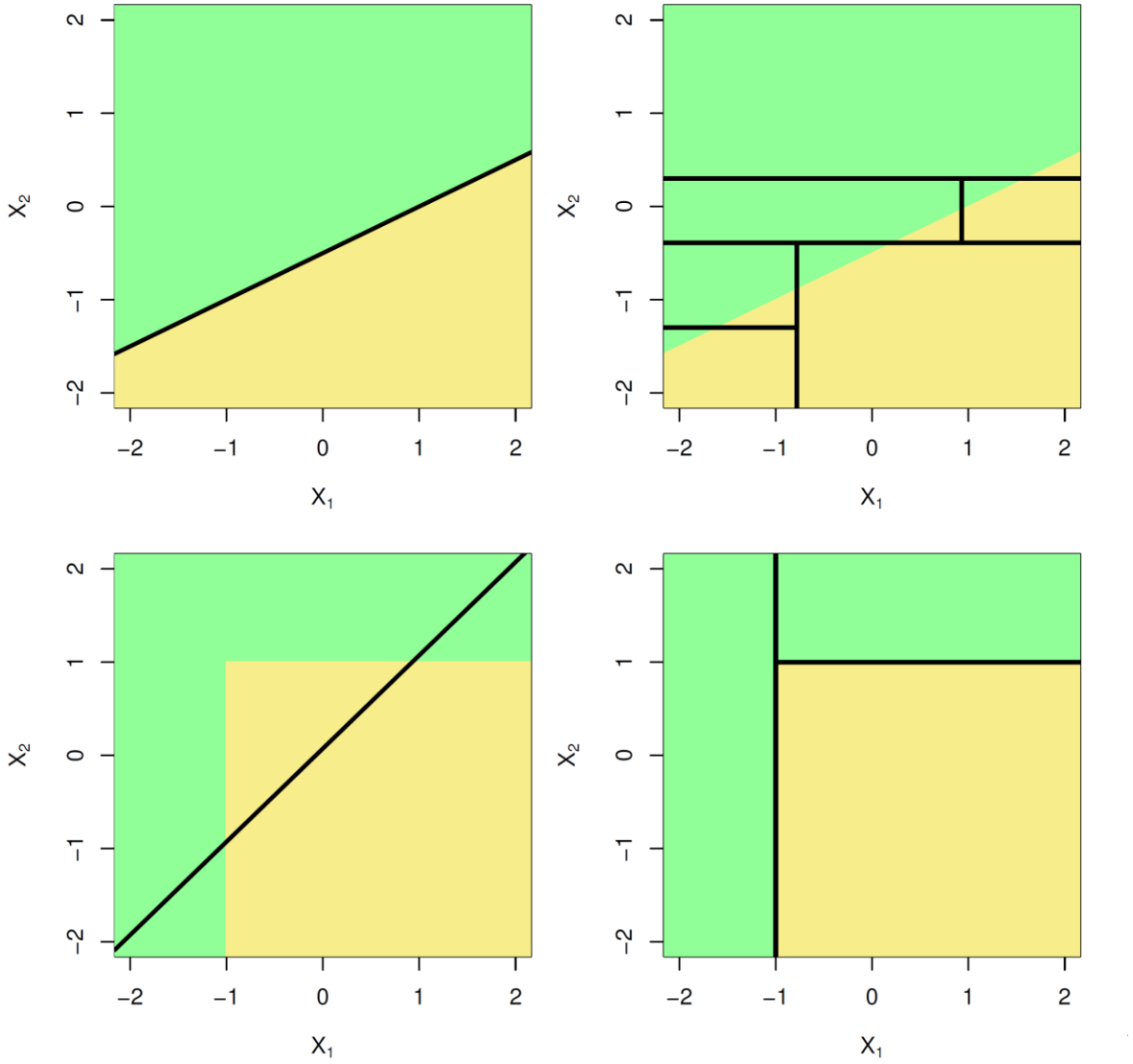


Trees Versus Linear Models

- Regression tree assume a model of a form

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

- Top Row: True linear boundary; Bottom row: true non-linear boundary
- Left column: linear model; Right column: tree-based model



Advantages and Disadvantages of Trees

- Pros
  - 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
- Cons
  - Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book
- However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next

Ensemble method - Bagging

- An ensemble method is an approach that combines many simple "building block" models in order to obtain a single and potentially very powerful model. These simple building block models are sometimes known as weak learners, since they may lead to mediocre predictions on their own
- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees
- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, 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. Of course, this is not practical because we generally do not have access to multiple training sets

Bagging— continued

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set
- In this approach we generate B different bootstrapped training data sets. We then train our method on the 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

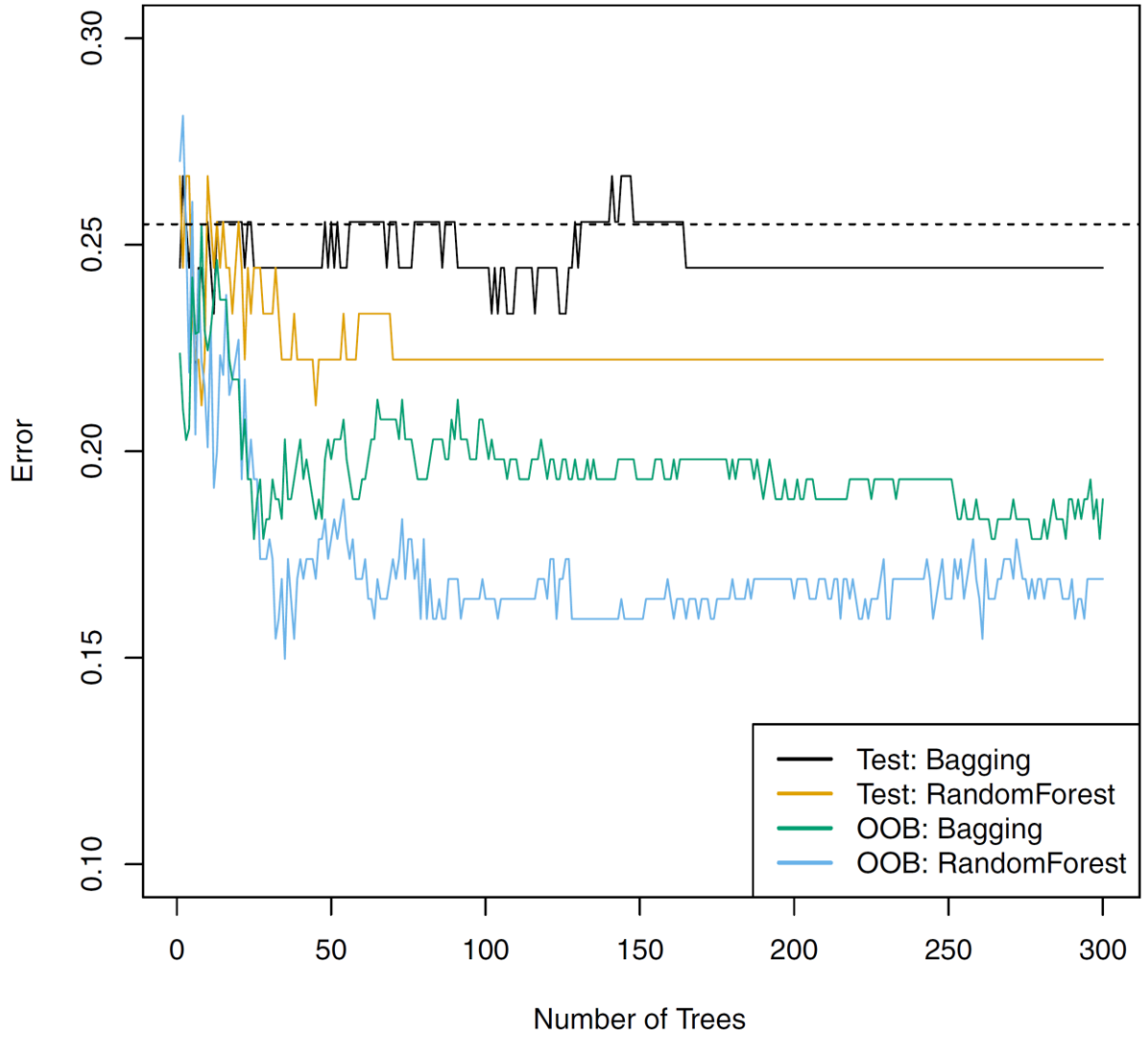
$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

- This is called bagging

Bagging classification trees

- The above prescription applied to regression trees
  - These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions

Bagging the heart data



Details of previous figure

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

Out-of-Bag Error Estimation

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

Ensemble method - Random Forests

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

Ensemble method - Random Forests

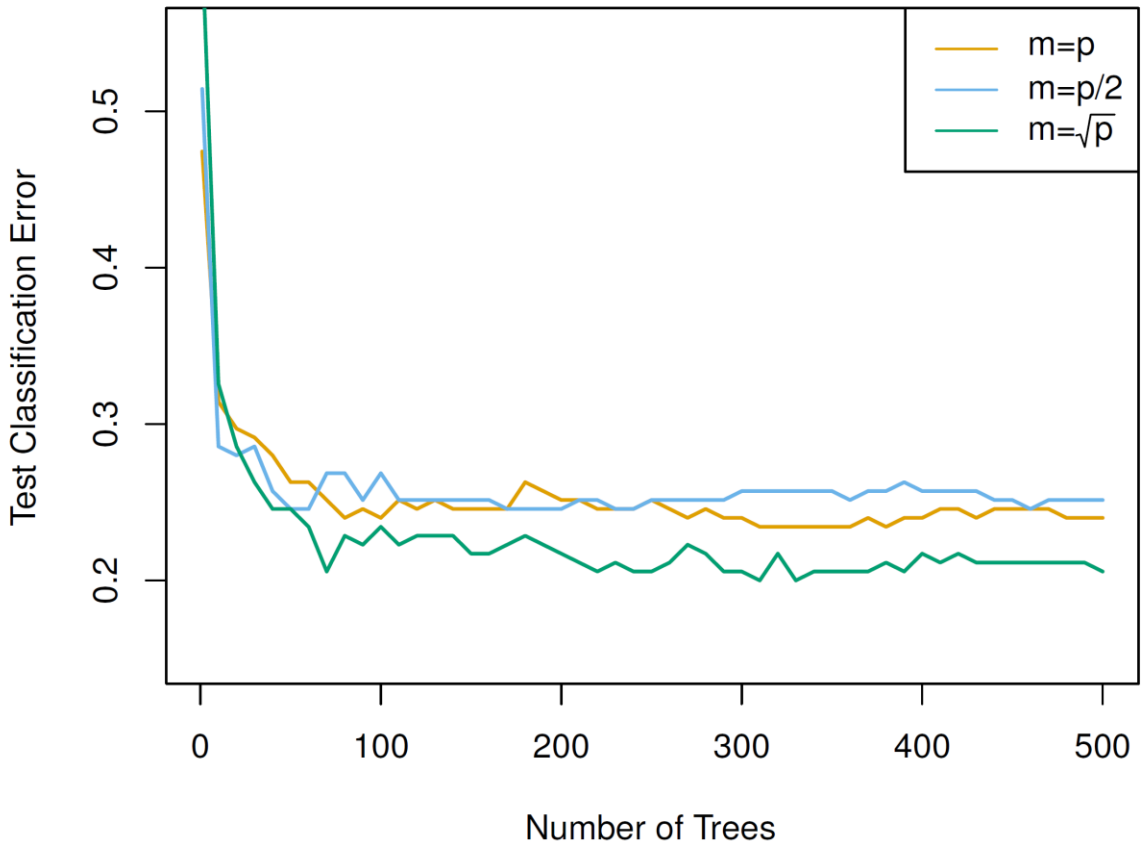
- Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split
  - Consequently, all of the bagged trees will look quite similar to each other
  - Averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities
  - Random forests overcome this problem by forcing each split to consider only a subset of the predictors
  - Using a small value of m in building a random forest will typically be helpful when we have a large number of correlated predictors

Example: gene expression data

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

Results: gene expression data

- As with bagging, random forests will not overfit if we increase B, so in practice we use a value of B sufficiently large for the error rate to have settled down



Details of previous figure

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

Ensemble method - Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification

- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model
  - Notably, each tree is built on a bootstrap data set, *independent* of the other trees

- Boosting works in a similar way, except that the trees are grown *sequentially:* each tree is grown using information from previously grown trees
  - Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set

Boosting algorithm for 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:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

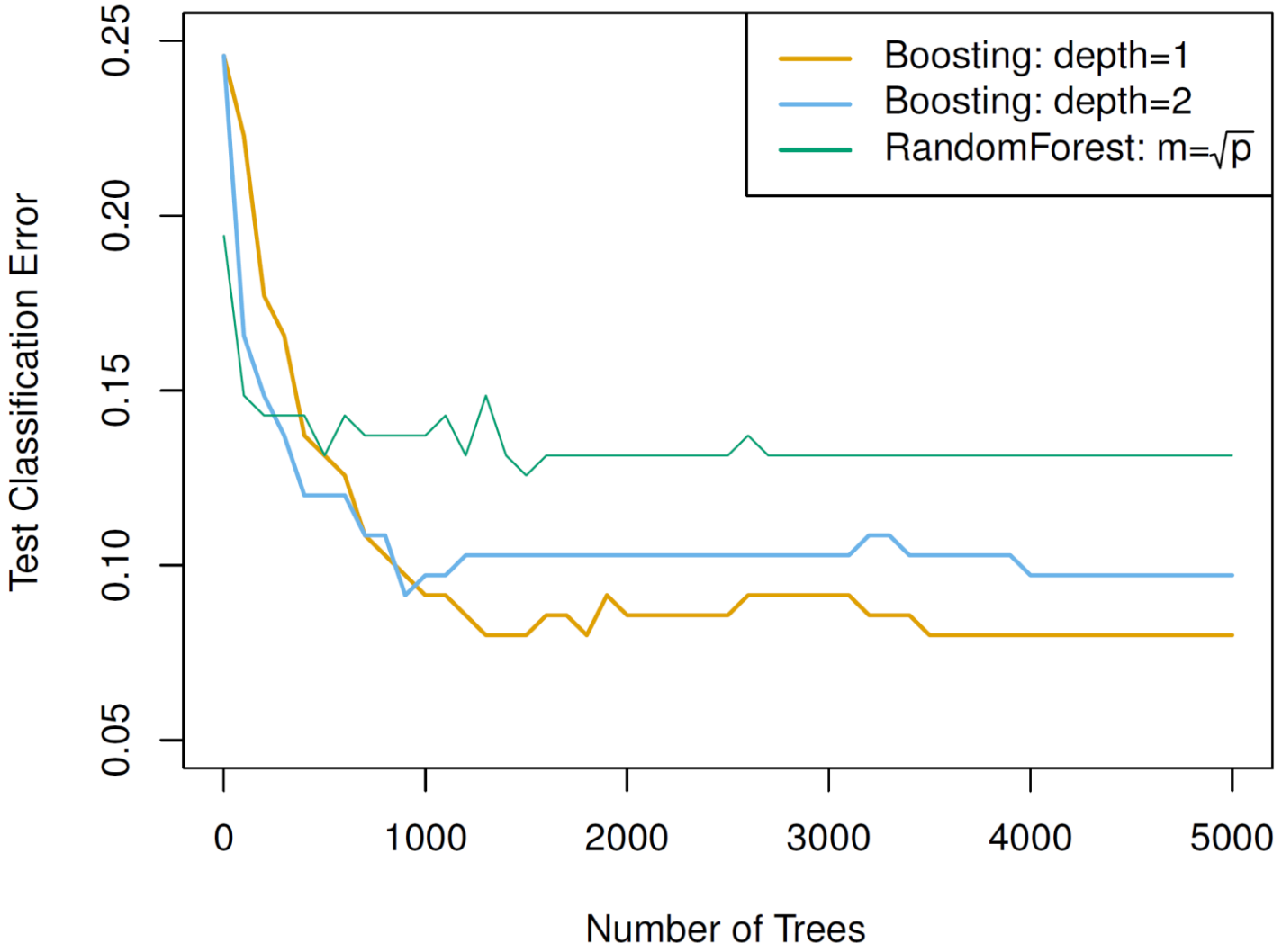
What is the idea behind this procedure?

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

Boosting for classification

- Boosting has three tuning parameters
  - The number of trees B (Choose by CV)
  - The shrinkage parameter λ (Typical values are 0.01 or 0.001)
  - The number d of splits in each tree (Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model)
- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here, nor do we in the text book
  - Can learn about the details in Elements of Statistical Learning, chapter 10
- The Python package XGboost (gradient boosted models) handles a variety of regression and classification problems using other tools besides decision tree

Gene expression data continued



Details of previous figure

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

Ensemble method - Bayesian Additive Regression Trees (BART)

- BART is related to the bagging and boosting approaches: each tree is constructed in a random manner as in bagging and random forests, and each tree tries to capture signal not yet accounted for by the current model, as in boosting
  - The main novelty in BART is the way in which new trees are generated
  - Let K denote the number of regression trees, and B the number of iterations for which the BART algorithm will be run. The notation $\hat{f}_k^b(x)$ represents the *prediction* at x for the kth regression tree used in the bth iteration
  - At the end of each iteration, the K trees from that iteration will be summed $\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x)$ for $b = 1, \ldots, B$

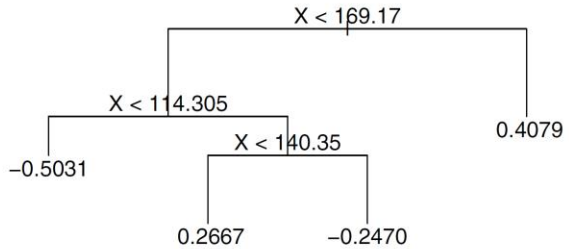
Bayesian Additive Regression Trees (BART)

- There are two components to this perturbation:
  1. We may change the structure of the tree by adding or pruning branches
  2. We may change the prediction in each terminal node of the tree
- Algorithm 8.3 can be viewed as a Markov chain Monte Carlo for fitting the BART model

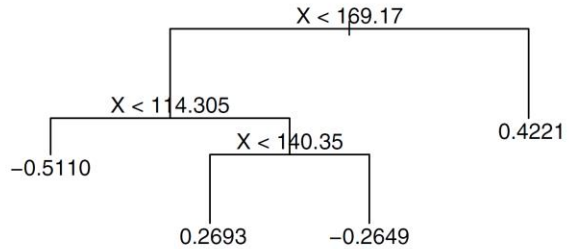
Algorithm 8.3 *Bayesian Additive Regression Trees*

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,
$$\hat{f}(x) = \frac{1}{B-L}\sum_{b=L+1}^{B} \hat{f}^b(x).$$

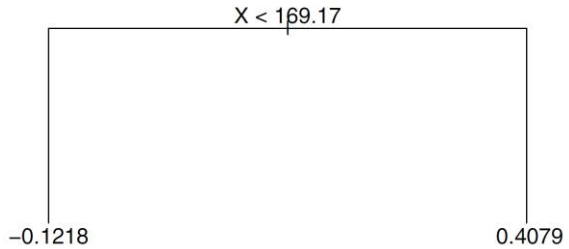
(a): $\hat{f}_k^{b-1}(X)$



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



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



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

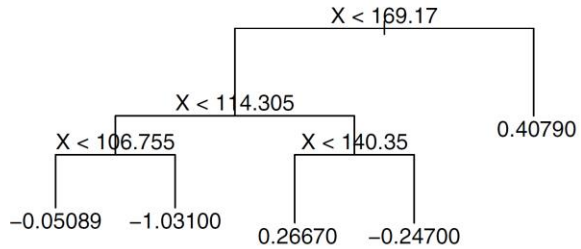


FIGURE 8.12. *A schematic of perturbed trees from the BART algorithm.* (a): *The kth tree at the* $(b-1)$*st iteration,* $\hat{f}_k^{b-1}(X)$*, is displayed. Panels (b)–(d) display three of many possibilities for* $\hat{f}_k^b(X)$*, given the form of* $\hat{f}_k^{b-1}(X)$*.* (b): *One possibility is 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* $\hat{f}_k^{b-1}(X)$*.*

Bayesian Additive Regression Trees (BART)

- We typically throw away the first few of these prediction models, since models obtained in the earlier iterations tend not to provide very good results
  - We can let L denote the number of burn-in iterations; for instance, we might take $L = 200$. Then, to obtain a single prediction, we simply take the average after the burn-in iterations, $\hat{f}(x) = \frac{1}{B-L}\sum_{b=L+1}^{B}\hat{f}^b(x)$
- A key element is that in Step 3(a)ii., we do not fit a fresh tree to the current partial residual: instead, we try to improve the fit to the current partial residual by slightly modifying the tree obtained in the previous iteration
- This guards against overfitting since it limits how "hard" we fit the data in each iteration. Furthermore, the individual trees are typically quite small. We limit the tree size in order to avoid overfitting the data, which would be more likely to occur if we grew very large trees

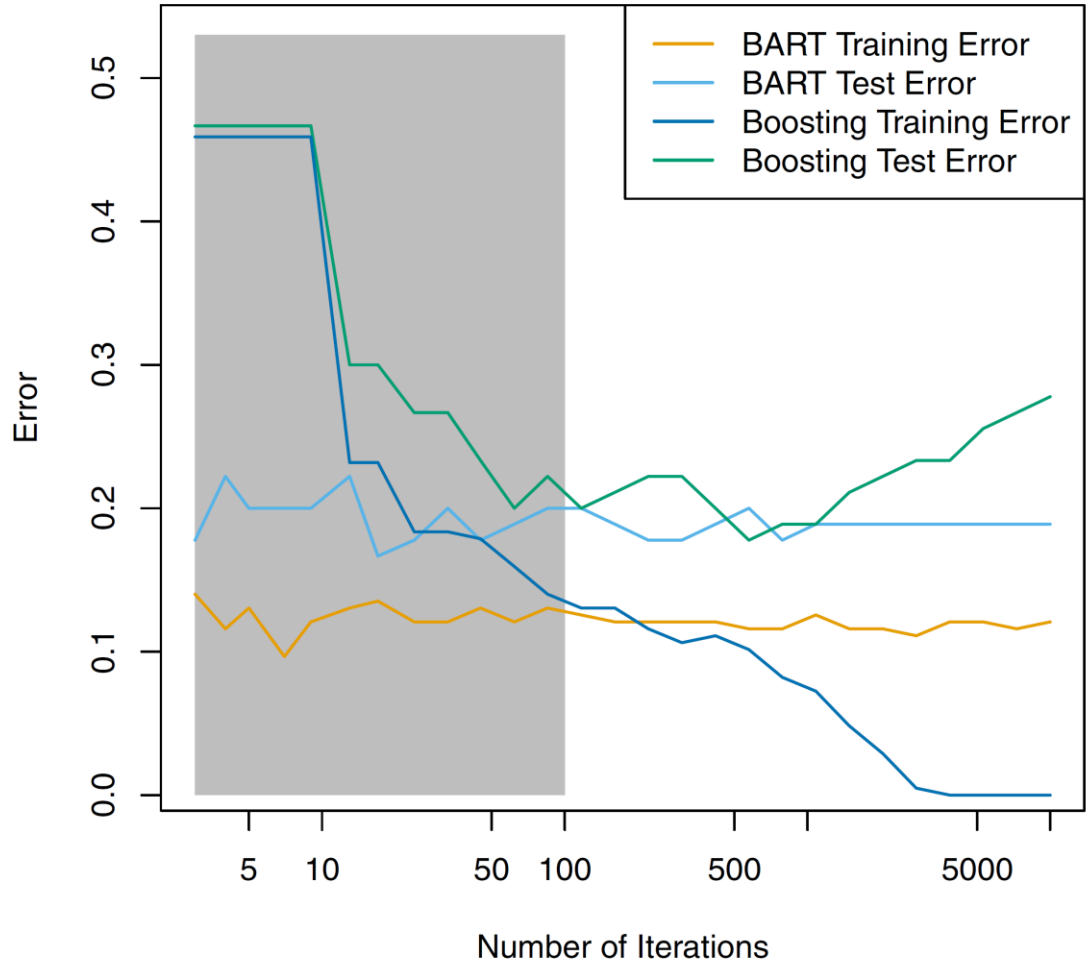


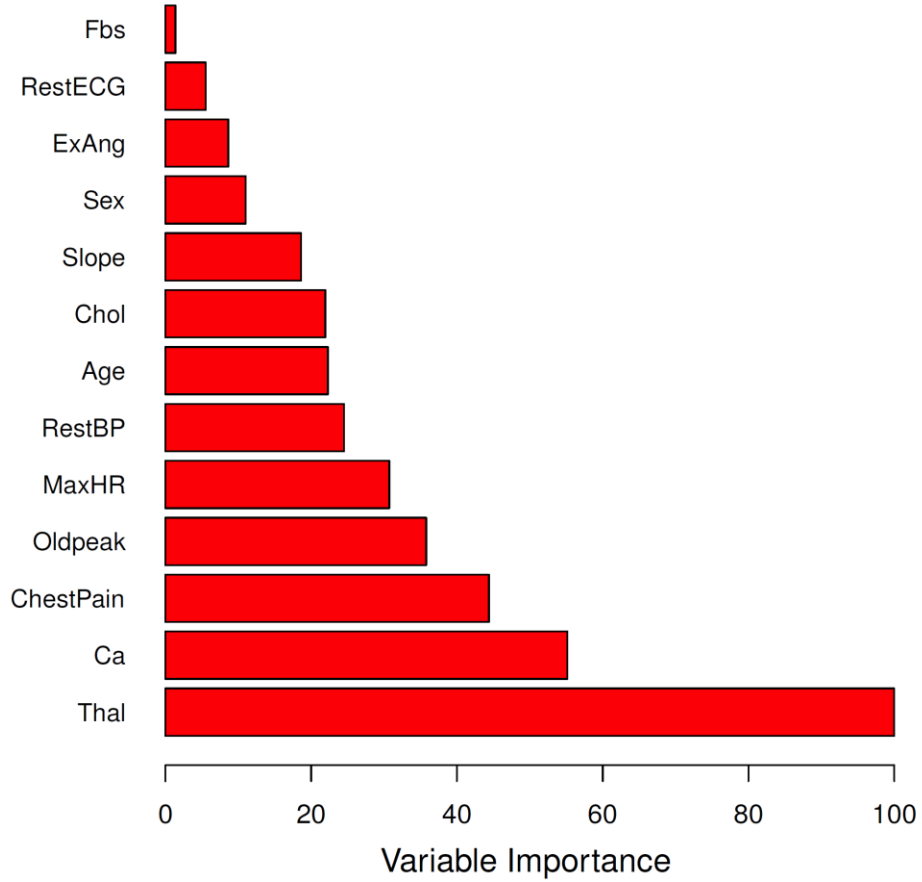
FIGURE 8.13. *BART and boosting results for the* `Heart` *data. Both training and test errors are displayed. After a burn-in period of* 100 *iterations (shown in gray), the error rates for BART settle down. Boosting begins to overfit after a few hundred iterations.*

Tuning parameters for BART

- When we apply BART, we must select the number of trees K, the number of iterations B, and the number of burn-in iterations L. We typically choose large values for B and K, and a moderate value for L
  - For instance, $K = 200$, $B = 1{,}000$, and $L = 100$ is a reasonable choice. BART has been shown to have very impressive out-of-box performance — that is, it performs well with minimal tuning

Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees



Summary

- Decision trees are simple and interpretable models for regression and classification
  - However they are often not competitive with other methods in terms of prediction accuracy
- In bagging, the trees are grown independently on random samples of the observations. Consequently, the trees tend to be quite similar to each other. Thus, bagging can get caught in local optima and can fail to thoroughly explore the model space
- In random forests, the trees are once again grown independently on random samples of the observations. However, each split on each tree is performed using a random subset of the features, thereby decorrelating the trees, and leading to a more thorough exploration of model space relative to bagging

Summary

- In boosting, we only use the original data, and do not draw any random samples. The trees are grown successively, using a “slow” learning approach: each new tree is fit to the signal that is left over from the earlier trees, and shrunken down before it is used
- In BART, we once again only make use of the original data, and we grow the trees successively. However, each tree is perturbed in order to avoid local minima and achieve a more thorough exploration of the model space

Appendix

The tree training algorithm

- ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data
- C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. These accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it

The tree training algorithm

- C5.0 is Quinlan's latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate
- CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node
- scikit-learn uses an optimised version of the CART algorithm; however, scikit-learn implementation does not support categorical variables for now
  - Scikit-learn's default max $\_features = n\_features$

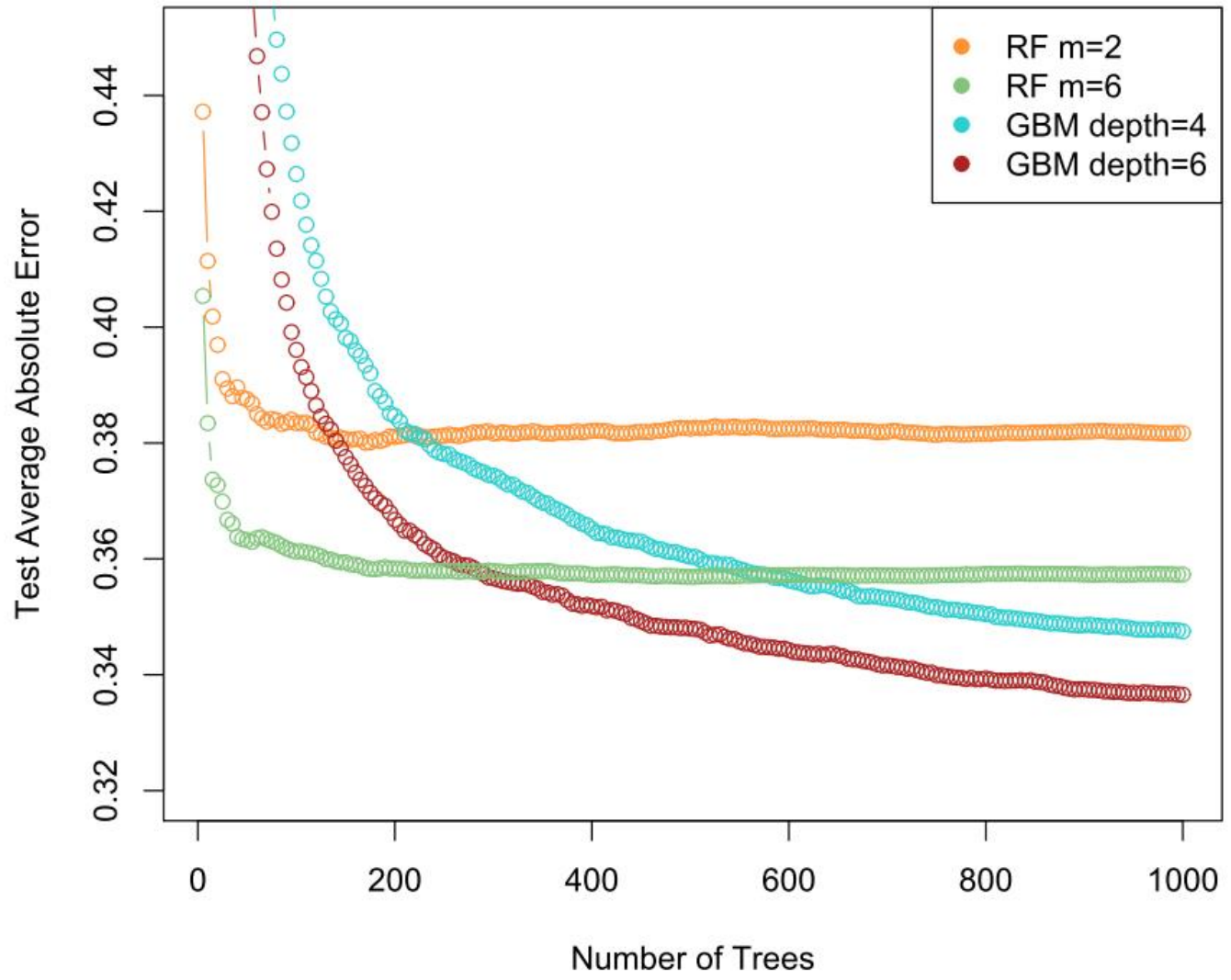
Other ensemble methods

- Extra-trees
  - In extremely randomized trees, randomness goes one step further in the way splits are computed
  - As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule
  - This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias
- AdaBoost
  - Scikit-learn's implementation
    - https://scikit-learn.org/stable/modules/ensemble.html#adaboost

Other ensemble methods

- Histogram-Based Gradient Boosting
  - These histogram-based estimators can be orders of magnitude faster
- Stacking
- Popular framework
  - XGBoost
  - CatBoost
  - LightGBM

Another regression example (California Housing Data)



Another classification example (Spam Data)