Chapter8 Tree based Methods

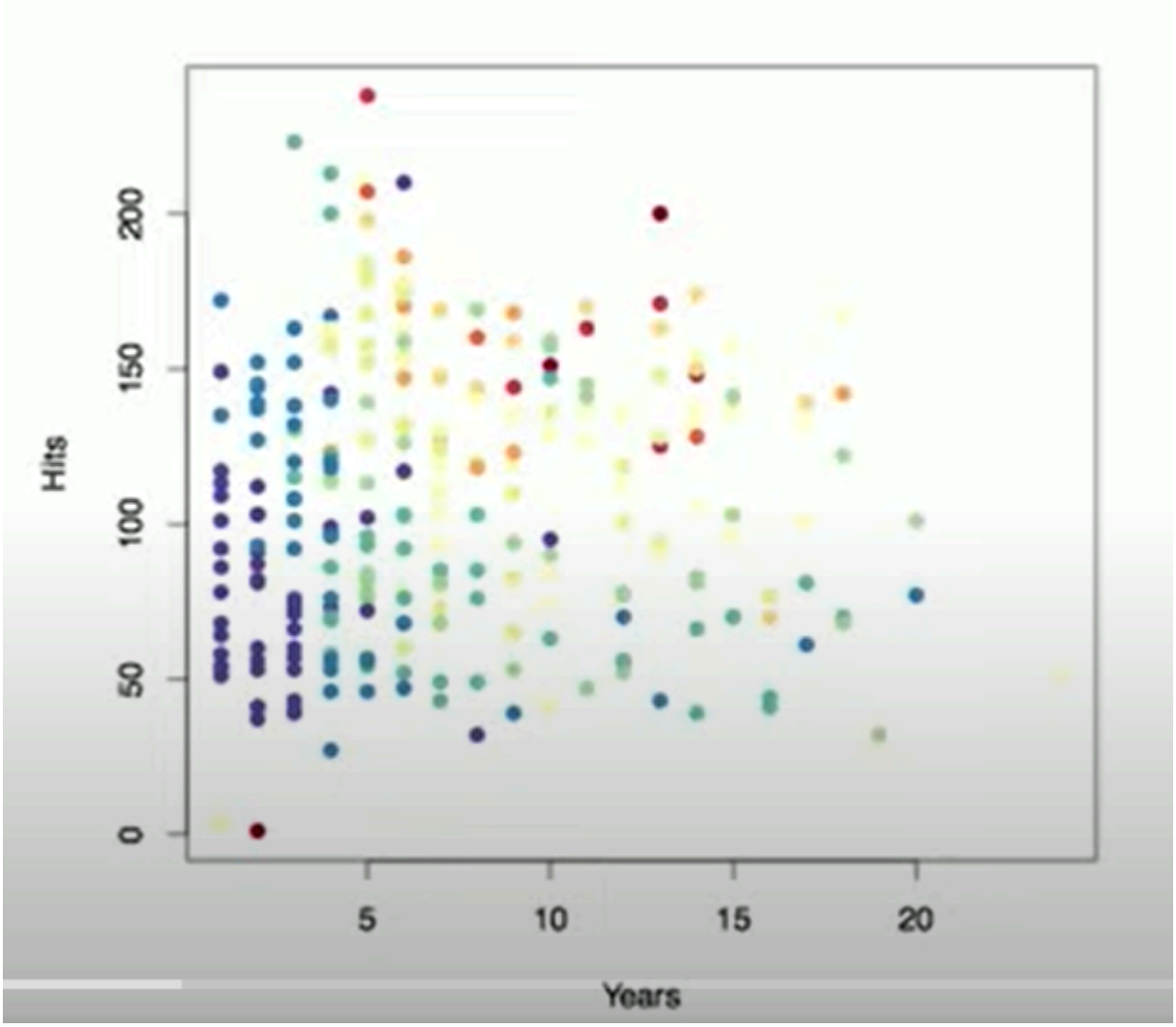
These methods involve **stratifying** or **segmenting** the predictor space into a number of simple regions. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as **decision-tree** methods.

Some pros and cons:

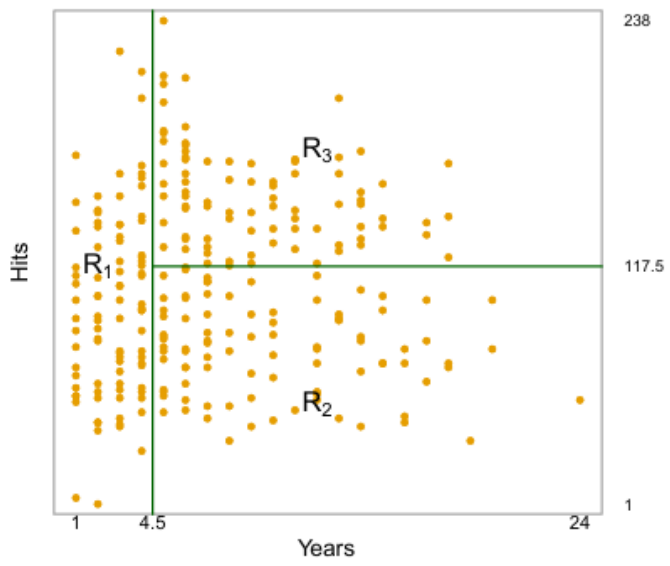
- They are simple and useful for interpretation.
- However they typically aren't competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss **bagging, random forests,** and **boosting** . These methods grow multiple trees which are then combined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

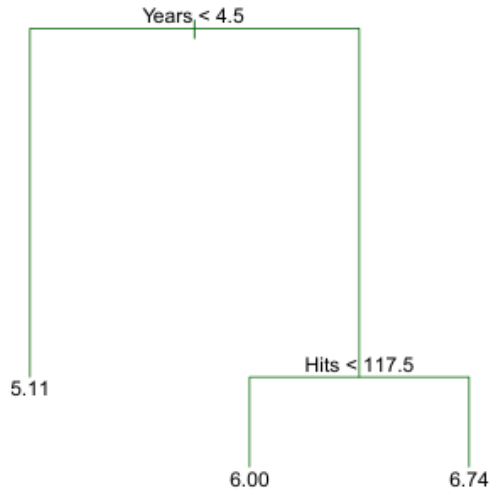
The basics of Decision Trees

You have a dataset containing the salary of baseball players, with predictors: hits per season and number of years he has been in the game. How would you stratify it? Separate high salary players from low salary players?
Salary is color coded from low (blue, green) to high (yellow, red).



This data can be divided into three regions. And the decision tree for this is in the right 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 $Years \geq 4.5$.
- 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.
- Overall the tree stratifies or segments the players into three regions of predictor space: $R_1 = \{X|Years < 4.5\}$, $R_2 = \{X|Years \geq 4.5, Hits < 117.5\}$ and $R_3 = \{X|Years \geq 4.5, Hits \geq 117.5\}$,as shown in the left figure above.

Terminology for Trees

- In keeping with the $tree$ analogy, the regions R_1, R_2, 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 $Years < 4.5$ $Hits < 117.5$.

Interpretation of the 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 effect **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's easy to display , interpret and explain.

Details of the tree-building process

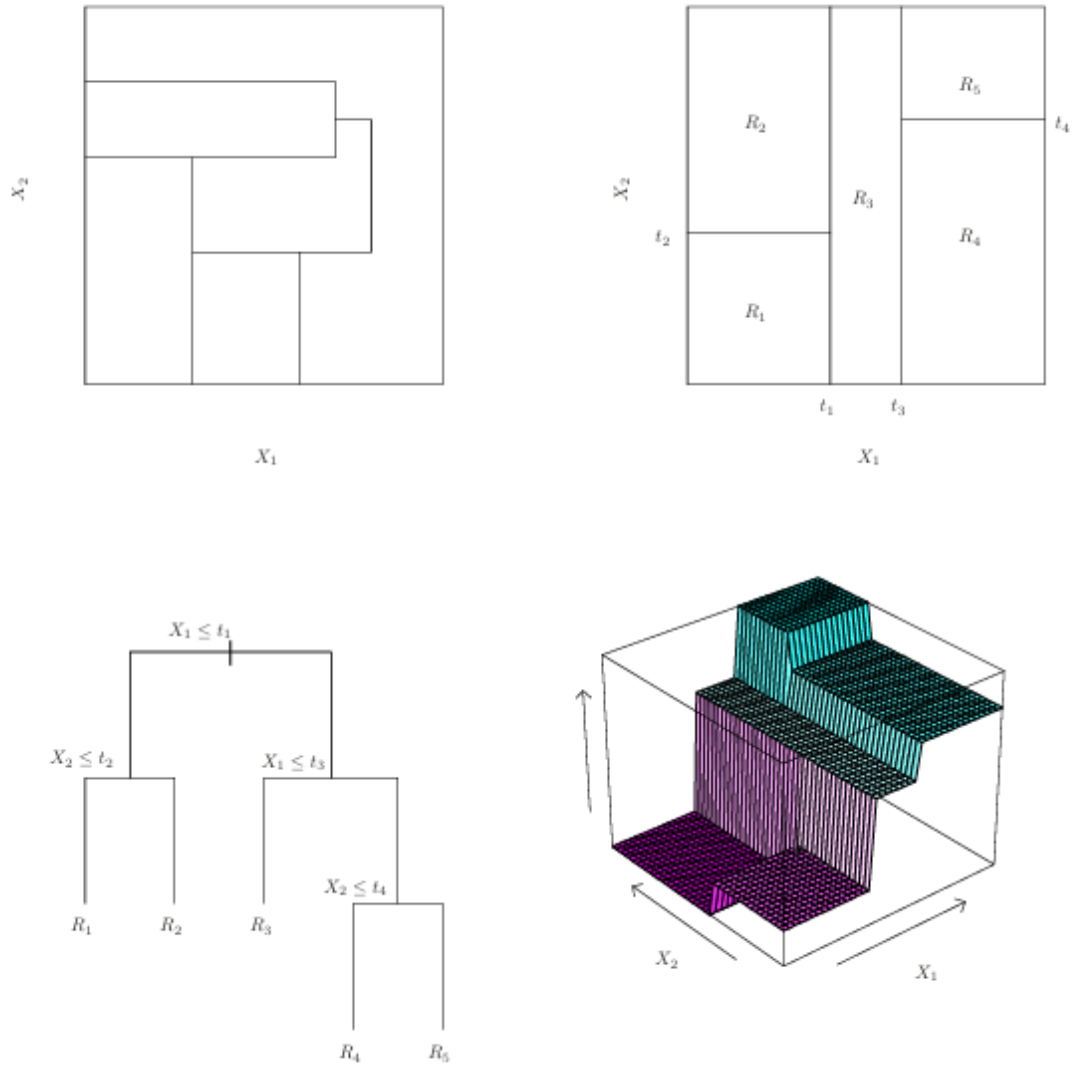
- We divide the predictor space - that is the set of possible values for $X_1, X_2, \ldots, X_p$ - into J distinct non-overlapping regions, $R_1, R_2, \ldots, R_J$.
- For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j .
- In theory the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or **boxes** , for simplicity and for ease of interpretation of the resulting predictive model
- The goal is to find the boxes $R_1, R_2, \ldots, R_J$ that minimizes the RSS given by

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

  where $\hat{y}_{R_j}$ is the mean response for the training observations within the j th box.
- Unfortunately it's 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 the particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.
- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $X|X_j < s$ and $X|X_j \geq s$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cut point in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Predictions

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



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

Pruning a Tree

- The process described above may produce good predictions on the training set, but likely to overfit the data, leading to poor test performance. Why? Well ! if you have a tree that's so large that each observation has its own terminal node , it's gonna have training error of zero, it's gonna overfit.
- A smaller tree with fewer splits (that is , fewer regions $R_1, R_2, \ldots, R_J$) .It 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 in the tree might be followed by a very good split - that is , a split that leads to a large reduction in RSS later on.
- A better strategy is to grow a very large tree T_0 , and then $prune$ it back in order to obtain a $subtree$
- **Cost complexity pruning** - also known as **weakest link pruning** - is used to do this.
- We consider a sequence of trees indexed by a non-negative tuning parameter α . For each value 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 .
- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We can find the best value of α by cross-validation.
- We then return to the full dataset and obtain the subtree corresponding to α .

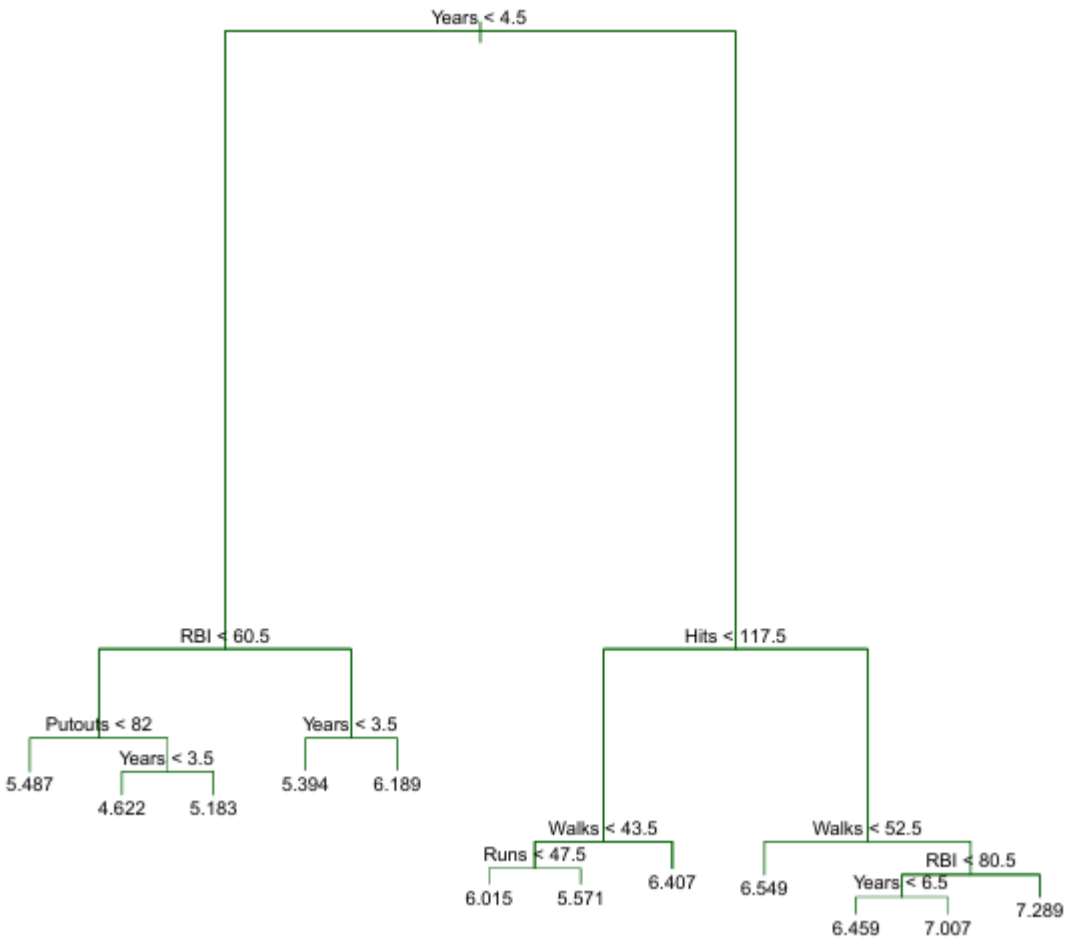
Summary : Tree algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
   3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.
4. Average the results, and pick α to minimize the average error.
5. Return the subtree from Step 2 that corresponds to the chosen value of α.

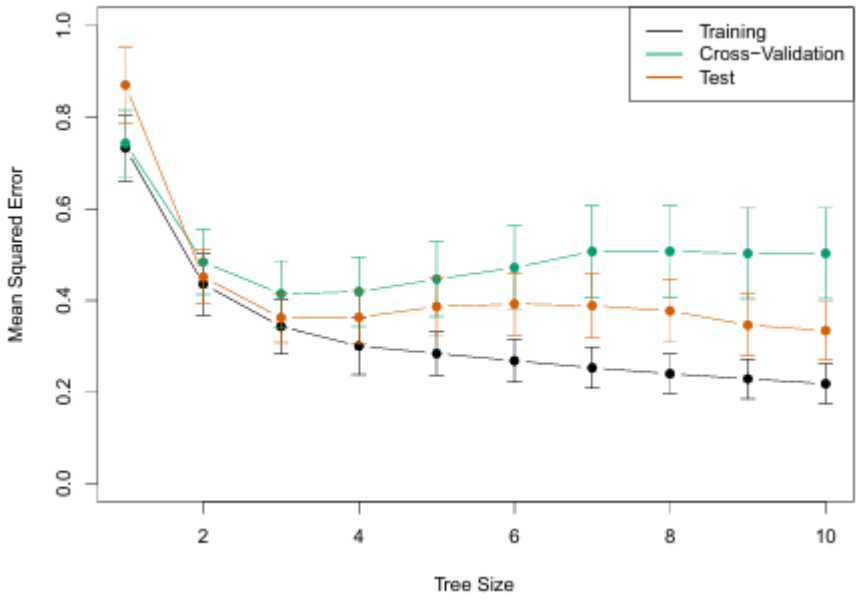
Now let us look at the baseball data again:

- First, we randomly divided the dataset 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-validation MSE of the trees as a function of α .



- Above is an unpruned tree with a rule that we don't split the region having less than 5 observations.
- Why are some of the arms represented longer than other? They denote the magnitude of decrement in RSS , as we go down the error reduction keeps on decreasing.



- 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

- Very similar to regression trees , 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 the training observations in the region to which it belongs to.
- We use similar recursive binary splitting to grow a classification tree.
- In classification, RSS can't be used as a criterion. A natural alternative is **classification error rate.** This is simply the fraction of the training observations in that region that don't 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 isn't sufficiently sensitive for tree-growing , and in practice two other measures are preferable.

1. $Gini\ index$:

- It is defined by

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

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

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

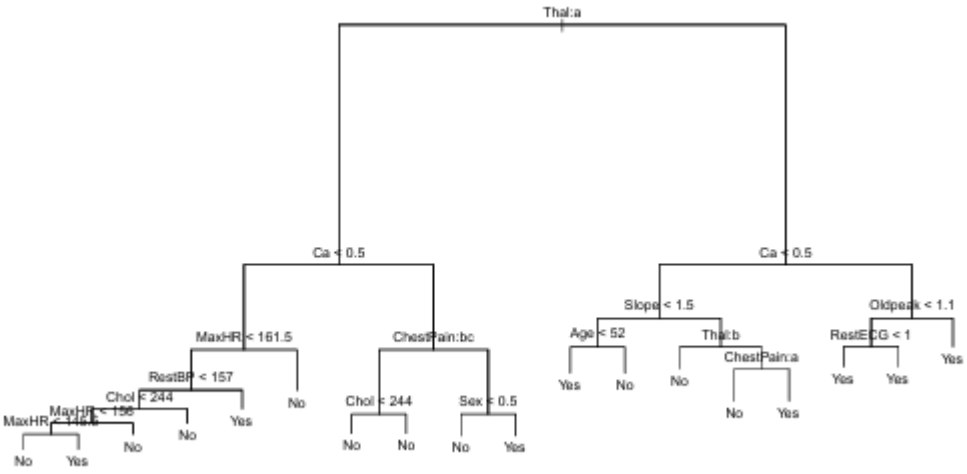
2. $Cross\ Entropy$:
    - It is given by

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

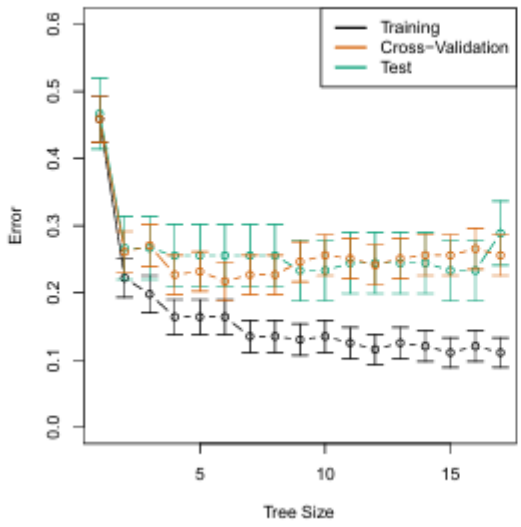
    - It turns out that Gini index and cross-entropy are very similar numerically.

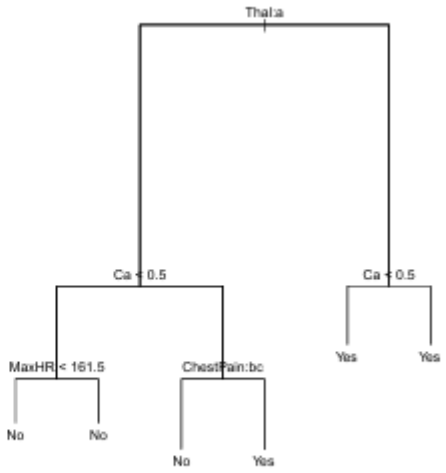
Let us look at an example of heart data:

- 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 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 measurement.
- Cross-validation yields a tree with six terminal nodes.



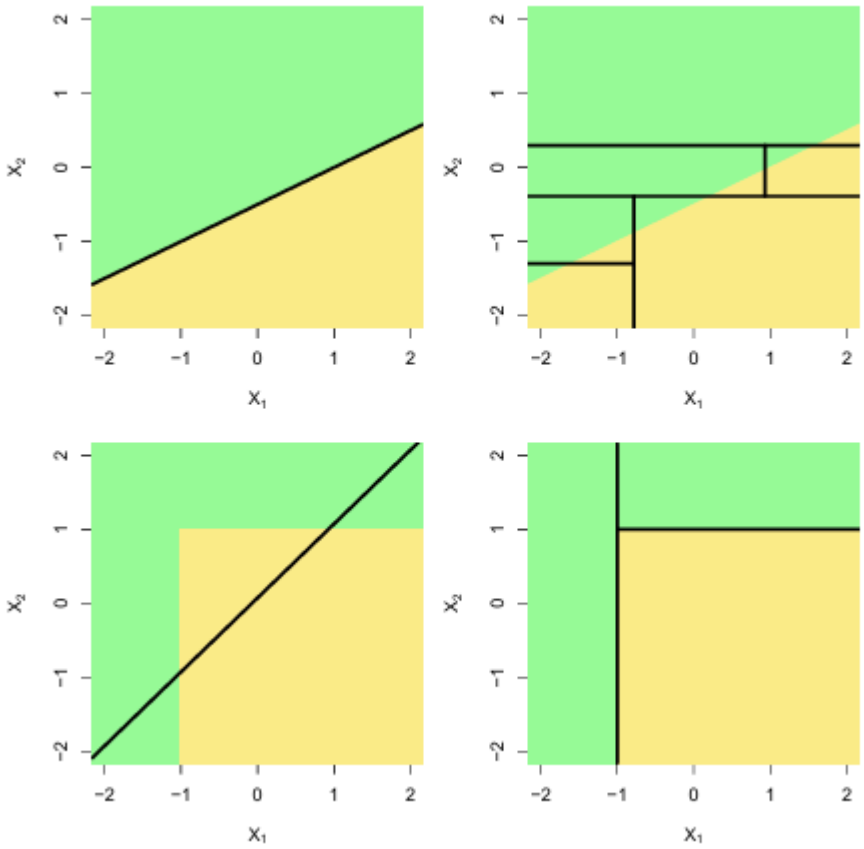
- The above figure shows an unpruned tree.
- In the figure below, on left we have Cross-validation error, training, and test error, for different sizes of the pruned tree. Remember that tree size is in one-one correspondence with α . And on the right, we got the pruned tree corresponding to the minimal cross-validation error.





- We see that the cross-validation curve and test curve almost follow each other , and we get the best tree size around six . Also remember that bars show the standard errors.

Trees versus Linear Models



- 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.

$Disadvantages\ of\ Trees$:

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.

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

Bagging

- **Bootstrap aggregation** or **bagging** is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with a 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 don't have access to multiple training sets.
- Instead , we can bootstrap, by taking repeated samples from the single training set.
- In this approach we generate B different bootstrapped training datasets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^*(x)$, 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 $bagging$.

- Here we don't prune the trees when fitting each bootstrapped training dataset.
- The old idea of pruning was to reduce the variance (the larger/bushy the tree, the larger the variance) but you get a lot of bias when you prune it back because you make a much coarser tree.
- What we are doing here is that we aren't pruning the trees, using bushy trees which tend to have a low bias and getting rid of variance by doing this averaging.
- The above prescription is applied to regression trees. For classification setting: 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.

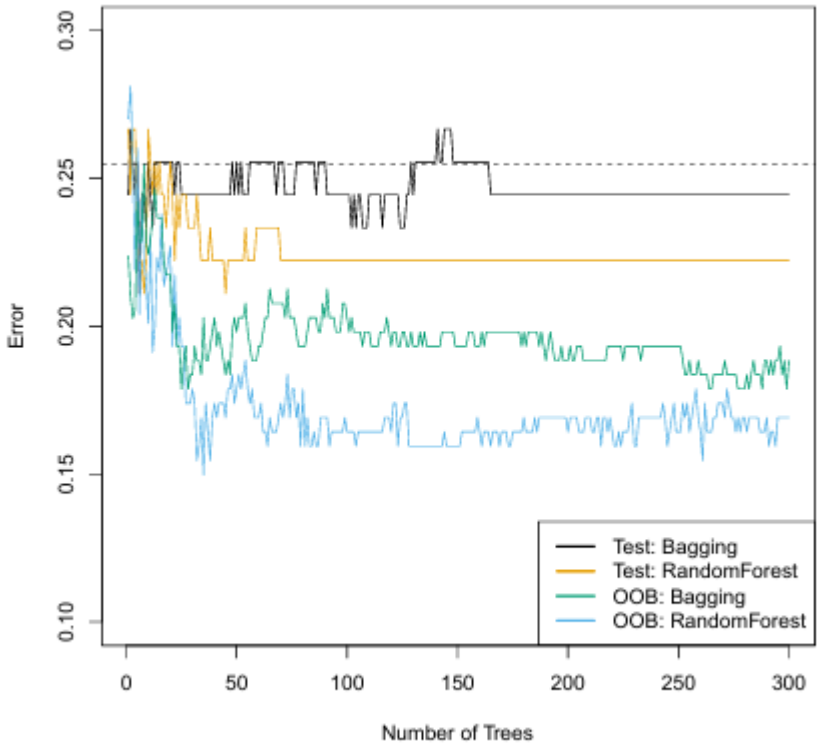
Out-of-bag error estimation

- 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.
- This estimate is essentially the LOO cross-validation error for bagging ,if B is large.

Random Forests

- **Random forests** provide an improvement over bagged trees by way of a small tweak that **decorrelates** the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, **a random selection of m predictors** is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.

- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).

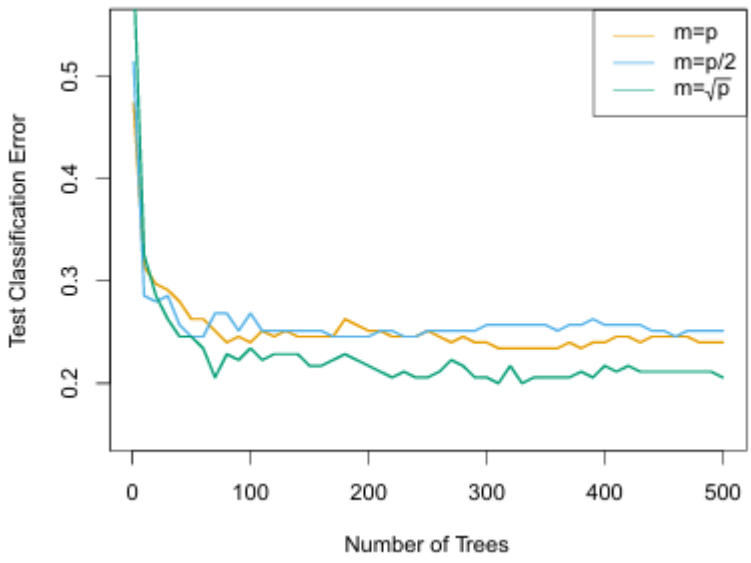


- Shown above is bagging and random forest results for the Heart data.
- The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used.
- Random forests were applied with $m = p$.
- The dashed line indicates the test error resulting from a single classification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

Now let us look at the 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. Is this a problem? No, because we did an unsupervised screening, if it would been supervised then it would have added bias to it.

- 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.



Note: By adding more trees in bagging or random forest, the only benefit you have is that it brings the variance down more but at some point the variance just stops decreasing and adding more trees doesn't help you but it'll never hurt you.

Boosting

- Like bagging , boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy , and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap dataset , 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 tree.

$Boosting\ Algorithm\ for\ regression\ trees$

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

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

2.3 Update the residuals,

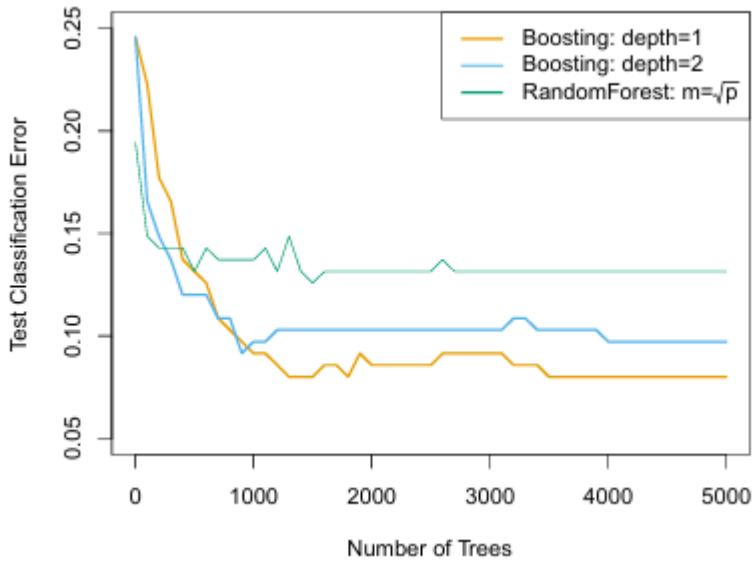
$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model,

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

- Unlike fitting in 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.

$Boosting\ for\ Classification$: Similar in spirit but bit more complex , so not covered. You can look into in **Elements of Statistical Learning , chapter 10** if you want.



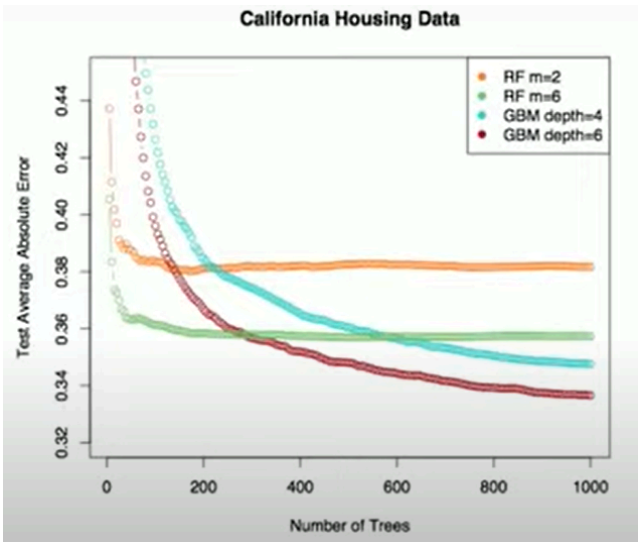
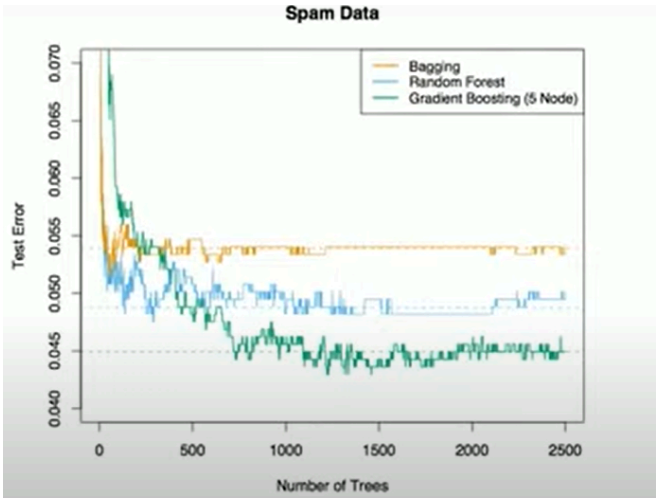
Results from performing boosting and random forests on the 15-class gene expression data set in order to predict cancer versus normal. The test error is displayed as a function of the number of trees. For the two boosted models, =0.01. Depth-1 trees slightly outperform depth-2 trees,

and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant. The test error rate for a single tree is 24%.

Tuning parameters for boosting

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

$Few\ other\ examples$:

California housing example to predict house prices	**Spam data**
	

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 classification trees , we add up the total amount the Gini index is decreased by splits over a given predictor , averaged over all B trees.

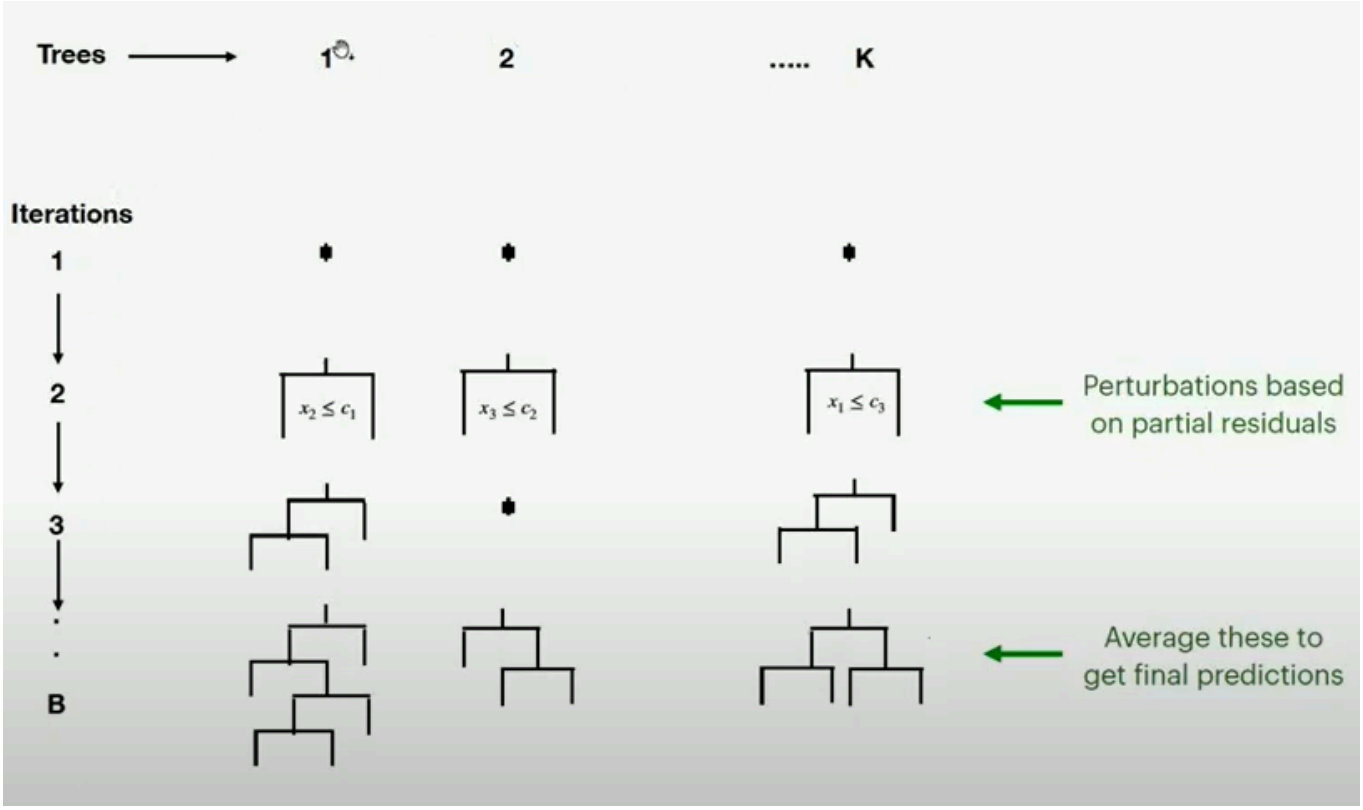
Summary of the discussion above

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

Bayesian Additive Regression Trees (BART)

- It uses decision trees as a building block.
- Recall that bagging and random forests make predictions from an average of regression trees, each of which is built using a random sample of data and/or predictors. Each tree is built separately from the others.
- By contrast, boosting uses a weighted sum of trees, each of which is constructed by fitting a tree to the residual of the current fit. Thus, each new tree attempts to capture signal that is not yet accounted for by the current set of trees.
- BART is related to both random forests and boosting: 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.
- BART can be applied to regression, classification and other problems; we will focus here just on regression.

$BART\ Algorithm$:



- We 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 the iteration will be summed i.e. , $\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x)$ for $b = 1, 2, \ldots, B$.

$BART\ iteration$:

- **In the first iteration** of the BART algorithm, all trees are initialized to have a single root node, with $f_k^1(x) = \frac{1}{nK} \sum_{i=1}^{n} y_i$, the mean of the response values divided by the total number of trees. Thus,

$$\hat{f}^1(x) = \sum_{k=1}^{K} \hat{f}_k^1(x) = \frac{1}{n} \sum_{i=1}^{n} y_i$$

- **In subsequent iterations**, BART updates each of the K trees, one at a time. In the bth iteration, to update the kth tree, we subtract from each response value the predictions from all but the kth tree, in order to obtain a **partial residual**

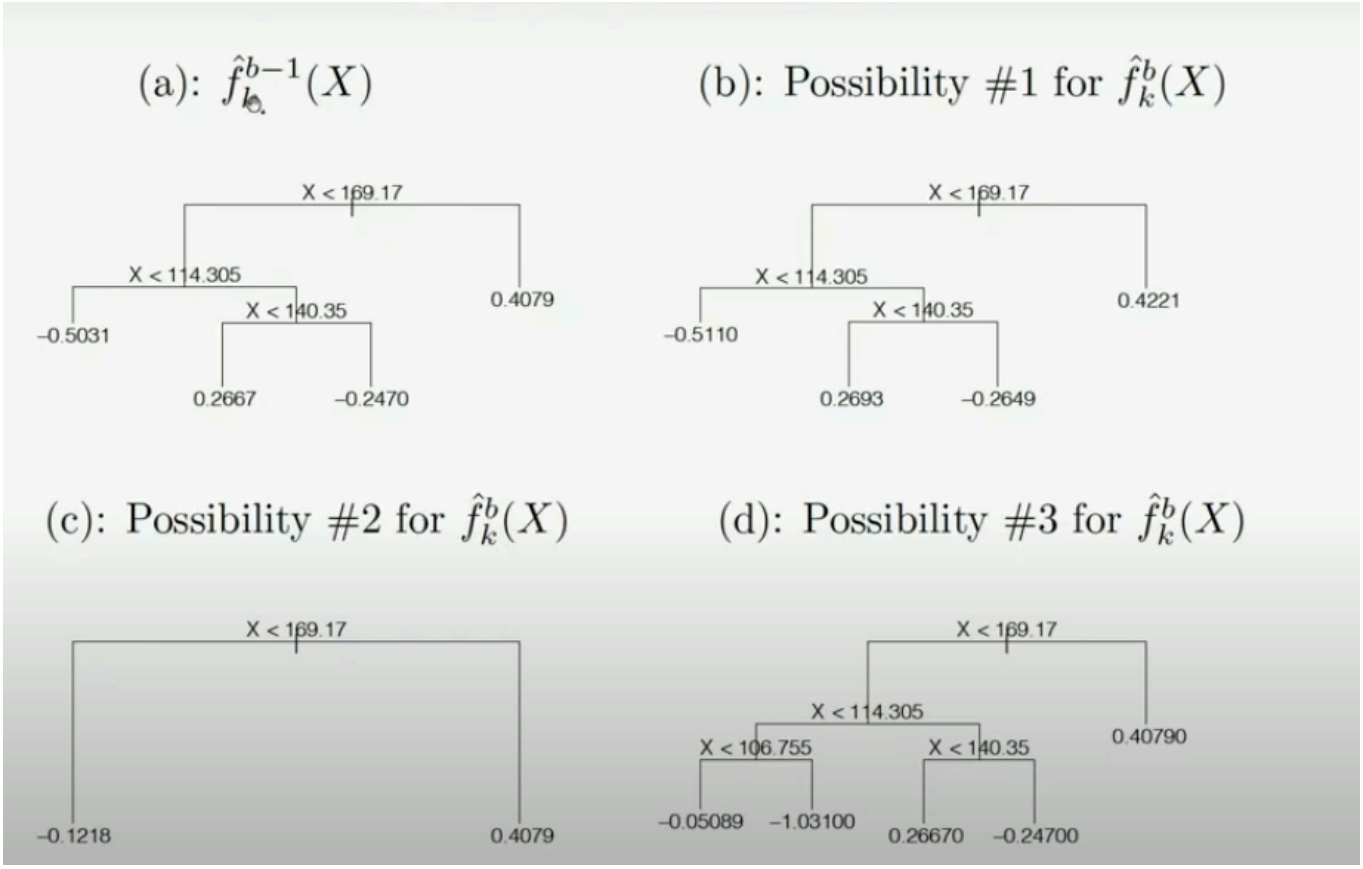
$$r_i = y_i - \sum_{k'<k} \hat{f}_{k'}^b(x_i) - \sum_{k'>k} \hat{f}_{k'}^{b-1}(x_i), i = 1, \ldots, n$$

Rather than fitting a fresh tree to this partial residual, BART randomly chooses a perturbation to the tree from the previous iteration ($\hat{f}_k^{b-1}$) from a set of possible perturbations, favoring ones

that improve the fit to the partial residual.
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.

Below are some examples of possible perturbations to the tree:



What does BART deliver?

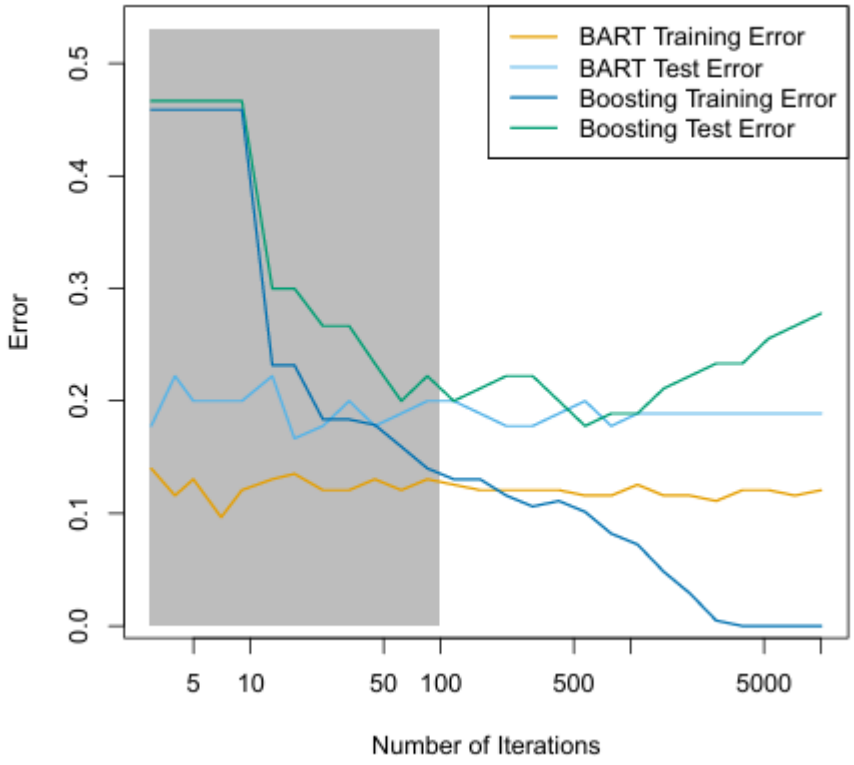
- The output of BART is a collection of prediction models,

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x), for, b = 1, 2, \ldots, B$$

- To obtain a single prediction, we simply take the average after some L **burn-in iterations**, $\hat{f}(x) = \frac{1}{B-L} \sum_{b=L+1}^{B} \hat{f}^b(x)$.
- The perturbation-style moves guard against overfitting since they limit how **hard** we fit the data in each iteration.
- We can also compute quantities other than the average: for instance, the **percentiles** of $f^{L+1}(x), \cdots f^B(x)$ provide a measure of uncertainty of the final prediction.
- For any given b , the BART prediction is like a boosting model; it's a sum of trees but the difference is that you got many of those and you've created these many by having this

randomization in tree selection each time. Also you are not building the tree by a simple tree fit to the residual , you are actually taking a existing tree and doing a perturbation .It's even slower than boosting; overfitting is reduced by this kind of slower learning.

- Let us look at an example: BART applied to the Heart Data



$K = 200$ trees; the number of iterations is increased to 10,000. During the initial iterations(in gray) , the test and training errors jump around a bit . After this initial burn-in-period , the error rates settle down. The tree perturbation process largely avoids overfitting.
Also notice that after a number of iterations, boosting is overfitting and test error goes up but BART is behaving like Random Forest, after you reach certain number of trees, it just flatten off and it doesn't seem to overfit.

$BART\ is\ a\ Bayesian\ Method$

- It turns out that the BART method can be viewed as a Bayesian approach to fitting an ensemble of trees: each time we randomly perturb a tree in order to fit the residuals, we are in fact drawing a new tree from a posterior distribution.
- Furthermore, the BART algorithm can be viewed as a **Markov chain Monte Carlo** procedure for fitting the BART model.

- We typically choose large values for B and K, and a moderate value for L: for instance, $K = 200$, $B = 1,000$, and $L = 100$ are reasonable choices. BART has been shown to have impressive out- that is , it performance performs well with minimal tuning.