# 07 - Trees

CART, Bagging, and Random Forest

SYS 6018 | Fall 2020

07-trees.pdf

## Contents



---


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.

# 1 Classification and Regression Tree Intro

## 1.1 Classification and Regression Trees

Tree-based methods:

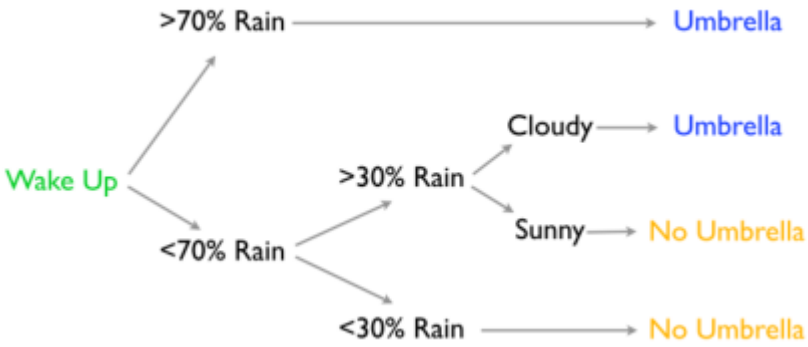
1. Partition the feature space into a set of (hyper) rectangles
2. Fit a simple model (e.g., constant) in each one.

They are conceptually simple yet powerful.

- Main Characteristics:
  - flexibility, intuitive, non-model based
  - natural graphical display, easy to interpret
  - building blocks of Random Forest and (Tree-based) Boosting
  - naturally includes feature interactions
  - reduces need for monotonic feature transformations
- Main Implementations:
  - CART (Classification and Regression Trees) by Breiman, Friedman, Olshen, Stone (1984)
  - C4.5 Quinlan (1993)
  - Conditional Inference Trees (`party` R package)

## 1.2 Decision Trees

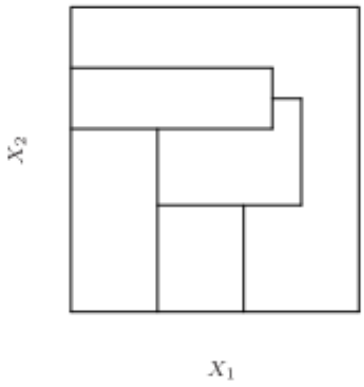
Example: http://www.20q.net/




## 1.3 Building Trees

As usual, we want find the tree that minimizes some loss.

- Classification trees have class probabilities at the leaves (e.g., the probability I'll be in heavy rain is 0.9).
  - E.g., Binomial likelihood.
- Regression trees have a mean response at the leaves. (e.g., the expected amount of rain is 2in).
  - E.g., squared error.

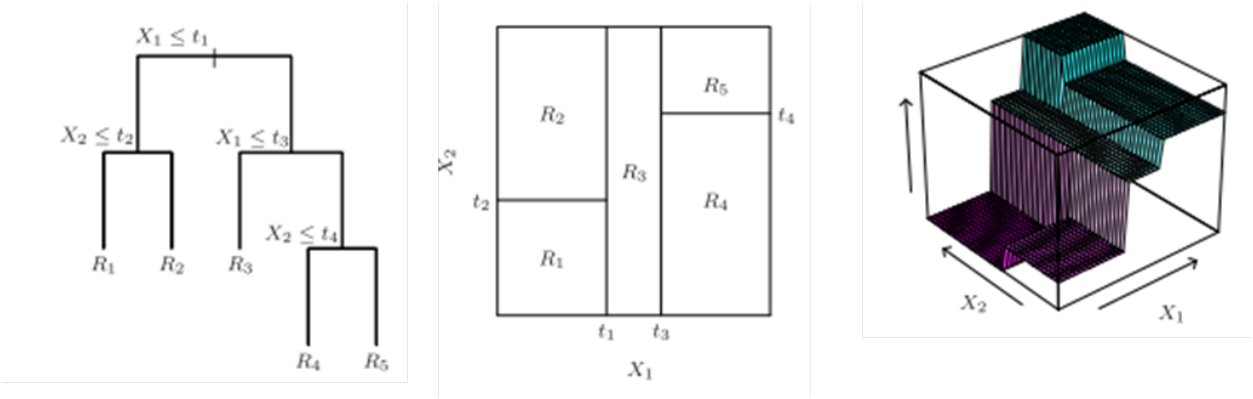



Note: This looks something like adaptive binning or nearest neighbor

## 1.4 Recursive Binary Partition (CART)

Because the number of possible trees is too large, we usually restrict attention to recursive binary partition trees (CART).

- These are also easy to interpret




Think of the *reverse* of agglomerative hierarchical clustering

- In hierarchical clustering, we started with all observations in clusters of size 1 and then sequentially grouped them together, according to some measure of *homogeneity/similiarity/distance/dissimilarity/loss*, until there was one big cluster.
  - The optimal clustering is usually somewhere between the two extremes
- In CART, all observations start in one big group and are split into two subgroups. Each subgroup is then split into two additional subgroups until there are some minimum number of nodes in each subgroup (leaf node).
  - The splitting is also based on some measure of homogeneity/similarity/loss.
  - Since we are in a supervised setting, the splitting criterion should be based on how well the new groups estimate the response.
  - There is another important difference: in CART only a single feature is used to determine the split

### 1.4.1 Model and Model Parameters

Model the response as a *constant* in each region

$$\hat{f}(x) = \sum_{m=1}^{M} \hat{c}_m \, \mathbb{1}(x \in \hat{R}_m)$$

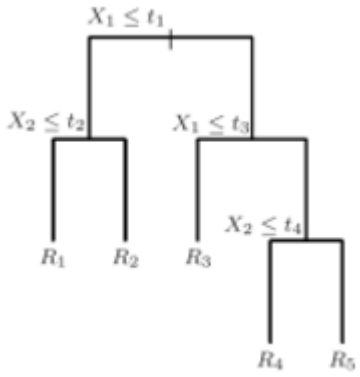
- The parameters of a tree, $T$, with $M$ leaf nodes, are:
  - The regions (*leaf nodes*) $R_1, \ldots, R_M$
  - The coefficients/scores for the regions $c_1, \ldots, c_M$
- The coefficients are based on the loss
  - Under Squared Error:

$$\begin{aligned} \hat{c}_m &= \text{Ave}(\{y_i : \mathbf{x}_i \in R_m\}) \\ &= \frac{1}{N_m} \sum_{i:\mathbf{x}_i \in R_m} y_i \end{aligned}$$

  - In classification models, the coefficients are *vectors* (one element for each class)

$$\begin{aligned} \hat{c}_{mk} &= \text{Proportion of class } k \text{ in region } R_m \\ &= \frac{1}{N_m} \sum_{i:\mathbf{x}_i \in R_m} \mathbb{1}(y_i = k) \end{aligned}$$

### 1.4.2 Basis Expansion Interpretation




$$f(x) = \sum_{m=1}^{M} \theta_m b_m(x)$$

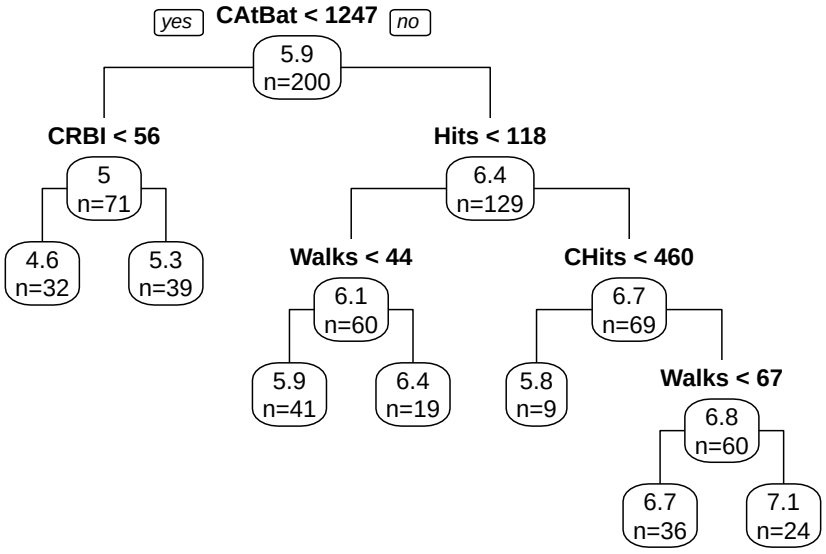
$$\begin{aligned} b_1(x_1, x_2) &= \mathbb{1}(x_1 \leq t_1) \, \mathbb{1}(x_2 \leq t_2) \\ b_2(x_1, x_2) &= \mathbb{1}(x_1 \leq t_1) \, \mathbb{1}(x_2 > t_2) \\ b_3(x_1, x_2) &= \mathbb{1}(x_1 > t_1) \, \mathbb{1}(x_1 \leq t_3) \\ b_4(x_1, x_2) &= \mathbb{1}(x_1 > t_1) \, \mathbb{1}(x_1 > t_3) \, \mathbb{1}(x_2 \leq t_4) \\ b_5(x_1, x_2) &= \mathbb{1}(x_1 > t_1) \, \mathbb{1}(x_1 > t_3) \, \mathbb{1}(x_2 > t_4) \end{aligned}$$

### 1.4.3 Example: Baseball Salaries

The `ISLR` R package (corresponding to the ISLR textbook), contains data (`Hitters`) on Major League Baseball players for the 1986-1987 season.

```
data(Hitters, package='ISLR')
```

Here is a CART tree for predicting the log of the salary (in thousands dollars):




## 1.5 Growing a Tree

CART uses a greedy algorithm to grow a tree.

- Split the feature space into two pieces and model response in each region
  - Find the predictor $j$ (out of $1, 2, \ldots, p$) and split point $t$ (from unique ordered values of $X_j$ or categories) to minimize the loss function
  - Produces two regions:

$$R_1(j,t) = \{x : x_j \leq t\} \text{ and } R_2(j,t) = \{x : x_j > t\} \qquad \textit{Numeric/Ordered Feature}$$

or

$$R_1(j,t) = \{x : x_j \in A_j\} \text{ and } R_2(j,t) = \{x : x_j \notin A_j\} \qquad \textit{Nominal/Categorical Feature}$$

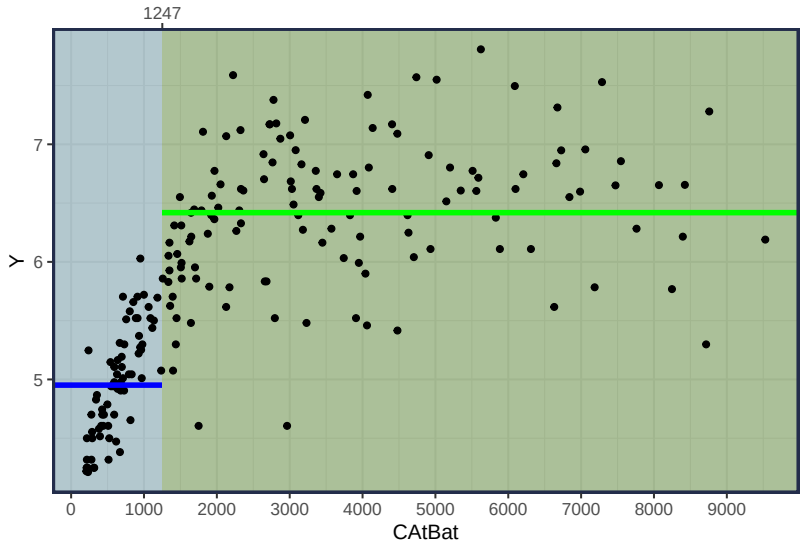
- Repeat this step for each child region
- Continue until stopping criteria met, e.g.
  - Minimum number of observations in region
  - Loss function has minimal improvement
- The final regions are called leaf nodes

## 1.6 Splitting Details

### 1.6.1 Regression Trees and Numeric Features

Notice in the fitted tree for the baseball data that the first split was based on a player's *Career At Bats* (`CAtBat`). Specifically, if a player has less than 1247 At Bats they go down the left side, otherwise they go down the right side.

Let's examine this first split:




- This is basically a univariate *change point model*
  - The split point (`CAtBat` < 1247) is the best change point (change in mean) using a Gaussian model
  - Alternatively, the reduction in MSE/SSE is maximized by splitting at (`CAtBat` < 1247) and fitting the data on each side of the split with a constant.

## Splitting Details

Notation

- $y \in \mathbb{R}$
- $\mathbf{x} = [x_1, \ldots, x_p]^\mathsf{T}$
- $n$ observations (in current node/region)

Consider a split on feature $j$:

- Before split, the quality of the model, based on SSE is:

$$Q_0 = \sum_{i=1}^{n}(y_i - \bar{y})^2 \qquad \text{where } \bar{y} = \sum_{i=1}^{n} y_i$$

- Consider a split at $s$ (on feature $j$)

**Left Region**

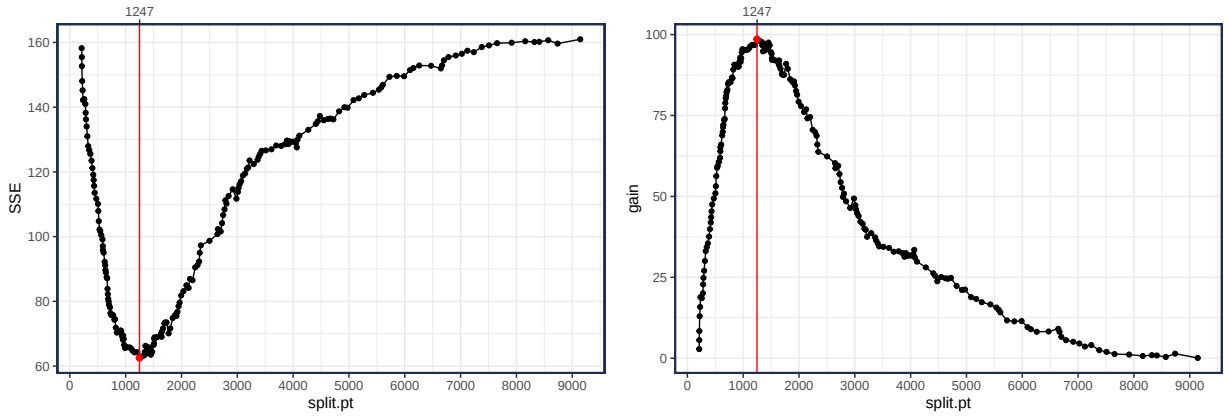
$$R_1(s) = \{x : x_j < s\}$$
$$\bar{y}_1(s) = \frac{1}{n_1} \sum_{\{i:x_i \in R_1(s)\}} y_i$$
$$Q_1(s) = \sum_{\{i:x_i \in R_1(s)\}} (y_i - \bar{y}_1(s))^2$$

**Right Region**

$$R_2(s) = \{x : x_j \geq s\}$$
$$\bar{y}_2(s) = \frac{1}{n_2} \sum_{\{i:x_i \in R_2(s)\}} y_i$$
$$Q_2(s) = \sum_{\{i:x_i \in R_2(s)\}} (y_i - \bar{y}_2(s))^2$$

- Updated SSE: $Q(s) = Q_1(s) + Q_2(s)$
- $\text{Gain}(s) = Q_0 - Q(s)$

We can examine the SSE (or Gain) for all possible split points:

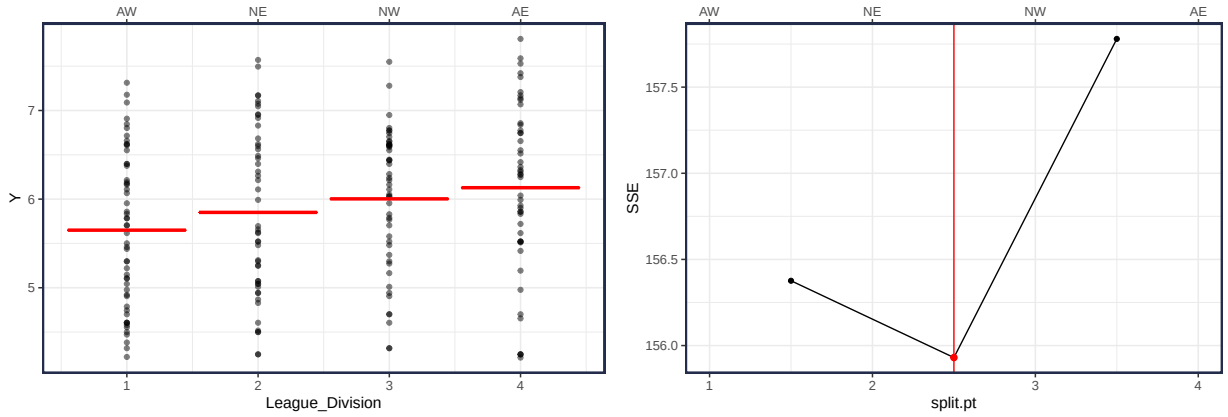



### 1.6.2 Regression Trees and Categorical (Nominal) Features

A categorical feature (with $k$ levels) can be split into two groups $2^{k-1} - 1$ different ways.

- $k = 3$: 3
- $k = 4$: 7
- $k = 10$: 511

The CART approach sorts the categories by the mean response (recodes to numeric) and then splits like its a numeric feature.




- Note: features with many levels will be split too often. Consider the quote from ESL (pg. 310)

  The partitioning algorithm tends to favor categorical predictors with many levels $k$; the number of partitions grows exponentially in $k$, and the more choices we have, the more likely we can find a good one for the data at hand. This can lead to severe overfitting if $k$ is large, and such variables should be avoided.

- An alternative is to use one-hot-encoding to split a categorical feature into $k$ new features.
  - As done by XGBoost
- There are other ways to *encode* categorical data so they can be treated like numeric (i.e., ordered data)
  - See e.g., CatBoost

### 1.6.3 Classification Trees

A classification tree is used when the response is categorical $y \in \mathcal{G} = (1, 2, \ldots, K)$.

- In region $R_m$, the probability of class $k$ can be estimated:

$$\begin{aligned}\bar{p}_m(k) &= \hat{\Pr}(y = k \mid \mathbf{x} \in R_m) \\ &= \frac{1}{n_m} \sum_{\{i:\mathbf{x}\in R_m\}} \mathbb{1}(y_i = k)\end{aligned}$$

- Each region has $K$-vector of probabilty estimates: $\bar{p}_m = [\bar{p}_m(1), \ldots, \bar{p}_m(K)]$

There are three primary measures of *node impurity* in this setting:

1. **Misclassification Error**:
$$Q_m = 1 - \max_k \bar{p}_m(k)$$

2. **Gini Index**:
$$Q_m = \sum_{k=1}^{K} \bar{p}_m(k)(1 - \bar{p}_m(k))$$

3. **Cross-entropy/Deviance**:
$$\begin{aligned}Q_m &= -\sum_{k=1}^{K} \bar{p}_m(k) \log \bar{p}_m(k) \\ &= \sum_{k=1}^{K} \bar{p}_m(k) \log \frac{1}{\bar{p}_m(k)}\end{aligned}$$

### 1.6.4 Splitting Summary

For each iteration, we calculate the Gain/Loss for *all* features $j = 1, 2, \ldots, p$ and *all* possible split points and choose the pair that maximizes the gain (or minimizes the overall loss):

$$\begin{aligned}j^* &= \arg\max_{j,s} \text{Gain}(j, s) \\ s^* &= \arg\max_{s} \text{Gain}(j^*, s)\end{aligned}$$

## 1.7 Stopping and Pruning

- Tree Size
  - A large tree (many leaf nodes with few observations) can overfit
  - A small tree can not capture important structure
  - Tree size is a tuning parameter governing the model's complexity, and the optimal tree size should be adaptively chosen from the data
- Early Stopping
  - Stop splitting when loss function has insignificant improvement (like forward stepwise)

  - However, a seemingly worthless initial split can lead to a good split further down the tree (short-sighted)

- Pruning
  - Grow a full tree (minimum node size) and prune back (like backwards stepwise)

### 1.7.1 Cost Complexity Pruning

- Let $N_m$ be the number of observations in node $R_m$ and $Q_m(T)$ be the loss in region $m$ for a given tree $T$. E.g.,

$$Q_m(T) = \frac{1}{N_m} \sum_{\{i:x_i \in R_m\}} (y_i - \hat{c}_m)^2$$

- *Weakest link pruning*: Successively collapse the internal node that produces the smallest per-node increase in $\sum_{m=1}^{|T|} N_m Q_m(T)$ until you reach the single node (root) tree. This produces a finite sequence of sub-trees.

- For each sub-tree $T$, define its cost complexity

$$C_\lambda(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \lambda |T|$$

  where the $m$ cover all terminal nodes in $T$ and $\lambda$ is a penalty on tree size $|T|$.

  - Note: The *complexity* of a tree in this setting is measured by the *number of leaf nodes*, $|T|$

### 1.7.2 Penalty Tuning

- For each $\lambda$, there is a unique smallest sub-tree $T_\lambda$ that minimizes $C_\lambda(T)$.
- The sequence of sub-trees from weakest link pruning contains every $T_\lambda$
- The tuning parameter, $\lambda$ can be chosen from cross-validation

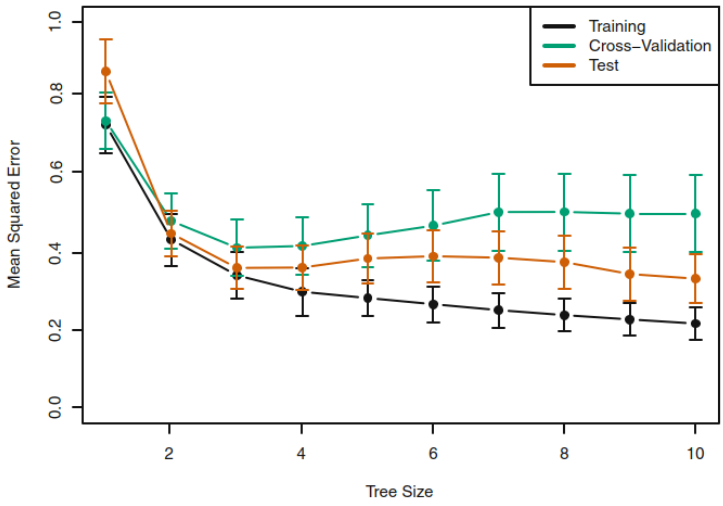



**FIGURE 8.5.** *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

## 1.8 Special Considerations

### 1.8.1 Missing Predictor Values

- For categorical predictors, create an additional level “missing"
  - This can reveal important patterns if *missing* is not at random.
- Surrogate splits
  - At every split, create a list of *surrogate splits* that mimics the original splits
    - For prediction, if an observation has a missing value use the surrogate splits to determine which child group it should go into
- Alternatively, we could omit observations with missing values or impute
  - The surrogate approach is similar to imputation, but is based on the “nearest neighbors”; the observations in the same branch of the tree.

### 1.8.2 Binary Splitting

- Multiway splits are possible, but could partition the data too quickly (overfit)
- Multiway splits can be achieved from a combination of binary splits
  - I.e., split on $X_1$ at $s_1$ and then split again on $X_1$ at $s_2$
  - This will/should happen when the true response is not a constant.

### 1.8.3 Variable/Feature Importance

There are several ways to measure the *importance* of a feature in a tree. Here are only a few:

- The number of times the feature was used to make a split
- The total reduction in loss (or increase in gain) for all the splits made by the feature
  - This is a weighted version of number of times feature used to make a split

- In CART, the features used to make the surrogate splits are also included
- Permutation based importance: re-run the tree, but include a *permuted* version of the feature. Any decrease in performace indicates the feature was important.
  - Note: consider what happens to features that are highly correlated/associated with other features

## 1.9 Tree Advantages

- Handles categorical and continuous data in consistent manner
- Automatic variable selection (any predictor not split)
- Automatically discover interactions between multiple predictors
  - The tree depth determines the possible number of interactions
- Because the partitions are made from the data, trees give a *locally adaptive* estimate
- Invariant to monotone transformations (some caveats)
- Can be robust to outliers in the feature space (splitting on sample quantiles)
- Easy to interpret

## 1.10 Tree Limitations

- Instability (high variance) due to greedy hierarchical structure; one change at top split can change remaining tree.
- Would be difficult to use trees for making inferences due to stepwise search (see above)
- Difficulty capturing additive structure

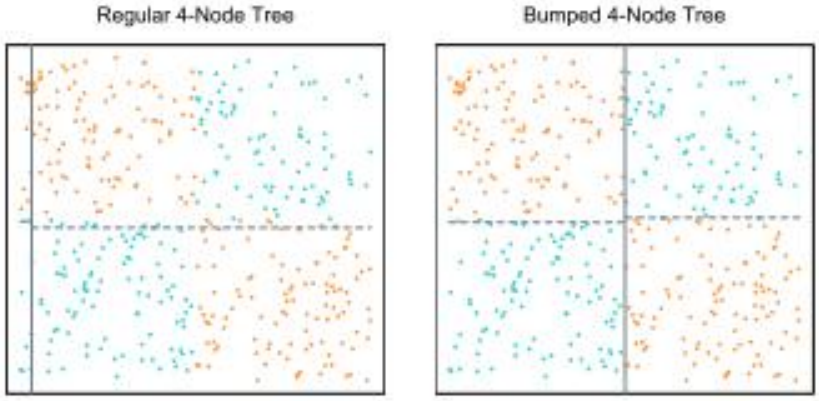



**FIGURE 8.13.** *Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two subsequent splits. The algorithm has no idea where to make a good initial split, and makes a poor choice. The right panel shows the near-optimal splits found by bumping the tree-growing algorithm 20 times.*

## 1.11 Trees in R

Main R packages: `tree` and `rpart` and `party`

# 2 Bagging Trees

## 2.1 Better Trees

- Because of the instability of trees, they are great candidates for methods like bagging that will reduce the variance.
- Grow a set of $B$ trees from a bootstrap samples and average their predictions:

$$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} T(x; \hat{\theta}_b)$$

  where $\hat{\theta}_b$ are all the estimated parameters for fitting the tree (split variables, cutpoints, and terminal (leaf) node values) from the bootstrap sample $\mathcal{D}_b$.
- Lots of detail and discussion can be found in Breiman's article "Bagging Predictors" (1996, *Machine Learning*).
  - Bagging = **B**ootstrap **Agg**regat**ing**
  - Lots of advice on when Bagging will help and when it won't
  - Bagging will help with variance reduction, but not bias
- Bagging produces an *ensemble model*
- Aggregation of Bagged Predictors:
  a. For regression: use the average predictions
  b. For classification: use the average of the predicted class probabilities (majority vote is possible too, but be careful about class imbalance or unequal misclassification costs)

### 2.1.1 Variance Reduction with Bagging

A helpful probability cheatsheet can be found here: https://github.com/wzchen/probability_cheatsheet/blob/master/probability_cheatsheet.pdf

**Properties of Variance/Covariance**

$$\begin{aligned}
\mathrm{V}(X) &= \mathrm{E}(X^2) - (\mathrm{E}(X))^2 \\
&= \mathrm{Cov}(X, X) \\
\mathrm{Cov}(X_1, X_2) &= \mathrm{E}(X_1 X_2) - \mathrm{E}(X_1)\,\mathrm{E}(X_2) \\
\mathrm{Cor}(X_i, X_j) &= \frac{\mathrm{Cov}(X_i, X_j)}{\sqrt{\mathrm{V}(X_i)\,\mathrm{V}(X_j)}} \\
\mathrm{V}(X_1 + X_2) &= \mathrm{V}(X_1) + \mathrm{V}(X_2) + 2\,\mathrm{Cov}(X_1, X_2) \\
\mathrm{V}\left(\sum_{i=1}^{p} X_i\right) &= \sum_{i=1}^{p} \mathrm{V}(X_i) + 2\sum_{i<j} \mathrm{Cov}(X_i, X_j)
\end{aligned}$$

**Variance Reduction**

- Let $\theta$ be something we want to estimate (e.g., $\theta = f(x)$) and $\hat{\theta}$ an estimate.
- Suppose we have $M$ models to estimate $\theta$ which produces the estimates $\{\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_M\}$
- One way to make an *ensemble prediction* is from the average

$$\bar{\theta} = \frac{1}{M}\sum_{i=1}^{M} \hat{\theta}_i$$

- The **expected value** of the ensemble is:

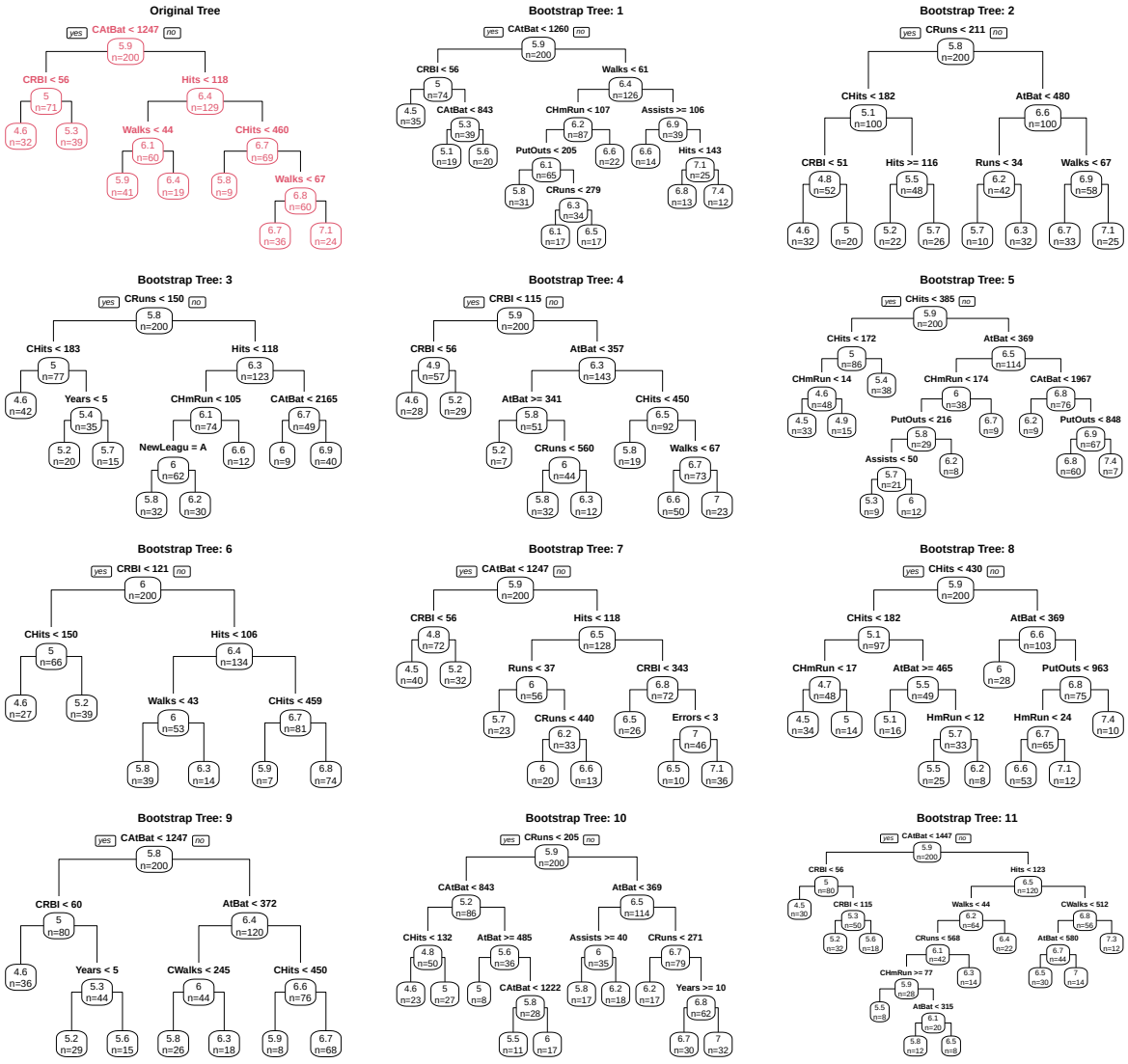
$$\mathrm{E}(\bar{\theta}) = \frac{1}{M}\sum_{i=1}^{M}\mathrm{E}(\hat{\theta}_i)$$

- The **variance** of the ensemble is:

$$\begin{aligned}\mathrm{V}(\bar{\theta}) &= \frac{1}{M^2}\sum_{i=1}^{M}\mathrm{V}(\hat{\theta}_i) + \frac{2}{M^2}\sum_{i<j}\mathrm{Cov}(\hat{\theta}_i,\hat{\theta}_j) \\ &= \frac{1}{M^2}\sum_{i=1}^{M}\mathrm{V}(\hat{\theta}_i) + \frac{2}{M^2}\sum_{i<j}\sqrt{\mathrm{V}(\hat{\theta}_i)\,\mathrm{V}(\hat{\theta}_j)}\,\mathrm{Cor}(\hat{\theta}_i,\hat{\theta}_j)\end{aligned}$$

- Thus to reduce the variance, we want to use models that have low correlation.
  - If $\mathrm{Cor}(\hat{\theta}_i,\hat{\theta}_j) = 0 \quad \forall i,j$, then variance is minimized (for example, when the models are *independent*)
  - If $\mathrm{Cor}(\hat{\theta}_i,\hat{\theta}_j) = 1 \quad \forall i,j$, then there is no (variance reduction) benefit of using an ensemble.
  - In Bagging, each *model* is a tree fit with a bootstrap sample.
  - For unstable models, like trees, the bagged models will have low correlation, but for more stable models, like linear regression, the bagged models will maintain high correlation.

## 2.2 Bagging Trees




(ESL pg 587) "The essential idea in bagging is to average many noisy but approximately unbiased models, and hence reduce the variance."

- Thus when Bagging trees, grow deep trees to reduce bias and use many bootstrap samples to reduce variance

# 3 Random Forest

## 3.1 Random Forest

Random Forest is a modification of bagging that attempts to build de-correlated trees and then average them

**Algorithm 15.1** *Random Forest for Regression or Classification.*

1. For $b = 1$ to $B$:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
   (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
      i. Select $m$ variables at random from the $p$ variables.
      ii. Pick the best variable/split-point among the $m$.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point $x$:

*Regression:* $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

*Classification:* Let $\hat{C}_b(x)$ be the class prediction of the $b$th random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

- Note: I recommend aggregating the probabilities for classification trees instead of majority vote.

## 3.2 Random Forest Tuning

There are two primary tuning parameters for Random Forest:

1. $m$ controls the number of predictors that are evaluated for each split
2. $min.obs$ or $depth$ controls how large the tree grows

How do these relate to the bias/variance trade-off?

Note:

- For classification, the default value is $m = \lfloor \sqrt{p} \rfloor$ and $min.obs = 1$.
- For regression, the default value is $m = \lfloor p/3 \rfloor$ and $min.obs = 5$.
- The tuning parameters can be determined from cross-validation or OOB error
- The *number of trees* is another tuning parameter, but want this to be as large as possible (computational and memory constraints)

### 3.3 OOB error

For each observation $(x_i, y_i)$, construct its OOB prediction by averaging only those trees corresponding to bootstrap samples in which observation $i$ did not appear.

$$\hat{f}(x_i) = \frac{1}{N_B(i)} \sum_{b=1}^{B} \mathbb{1}(x_i \in \mathrm{OOB(b)}) \cdot \mathrm{T(x_i; \theta_b)}$$

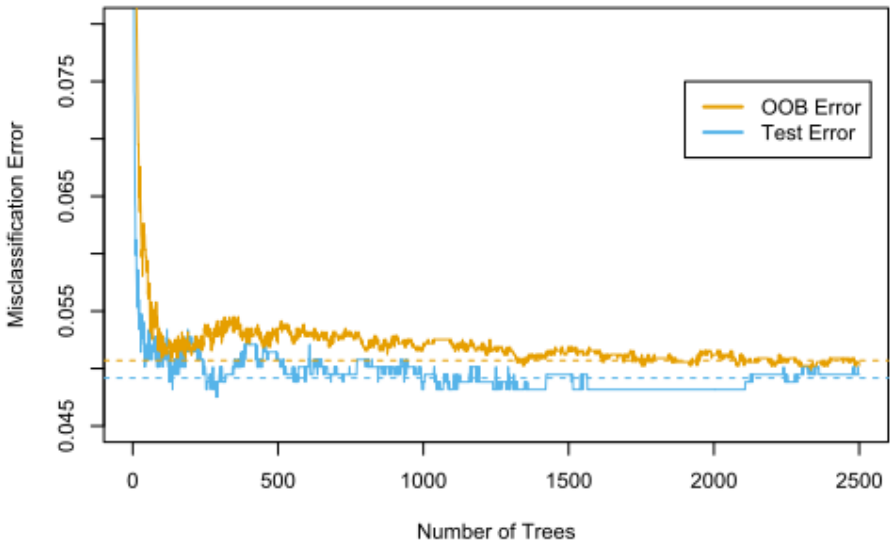



Figure 15.4 in ESL

### 3.4 Variable Importance

At each split in each tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable, and is accumulated over all the trees in the forest separately for each variable.

The importance of predictor $j$ in a single tree $T$:

$$\mathcal{I}_j^2(T) = \sum_t i_j^2 \cdot \mathbb{1}(v(t) = j)$$

The importance of predictor $j$ in a forest:

$$\mathcal{I}_j^2 = \frac{1}{B} \sum_{b=1}^{B} \mathcal{I}_j^2(T_b)$$

### 3.5 Random Forest and k-NN

Random Forests (especially with fully grown trees) are similar to $k$-NN methods, but adaptively determines the neighbors.

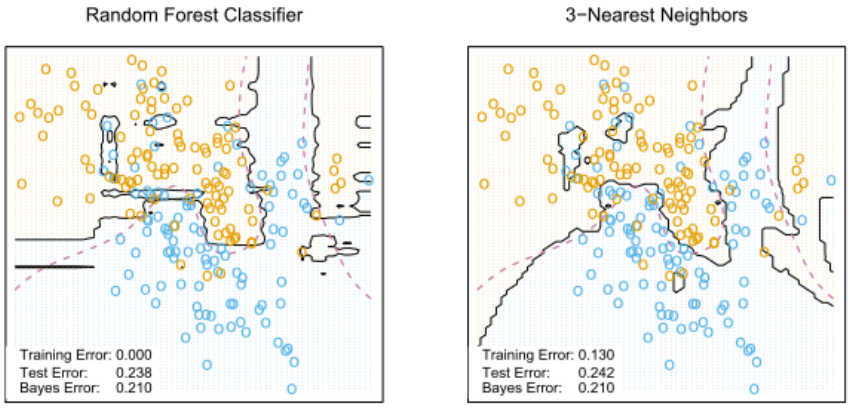



**FIGURE 15.11.** *Random forests versus 3-NN on the mixture data. The axis-oriented nature of the individual trees in a random forest lead to decision regions with an axis-oriented flavor.*