

Prediction Trees

CART, Bagging, and Random Forest

DS 6030 | Fall 2023

trees.pdf

Contents

1 Classification and Regression Tree Intro	2
1.1 Decision Trees	2
1.2 Building Trees	2
1.3 Recursive Binary Partition (CART)	2
1.4 Growing a Tree	5
1.5 Splitting Details	6
1.6 Stopping and Pruning	9
1.7 Special Considerations	10
1.8 Tree Advantages	11
1.9 Tree Limitations	13
1.10 Trees in R	13
2 Trees Demo	14
2.1 Required R Packages	14
2.2 Baseball Salary Data	14
2.3 Regression Tree	14
2.4 Details of Splitting (for Regression Trees)	21
3 Bagging Trees	26
3.1 Better Trees	26
3.2 Bagging Trees	29
4 Random Forest	31
4.1 Random Forest	31
4.2 Random Forest Tuning	34
4.3 OOB error	34
4.4 Variable Importance	35
4.5 Random Forest and k-NN	35
4.6 Random Forests in R	36

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

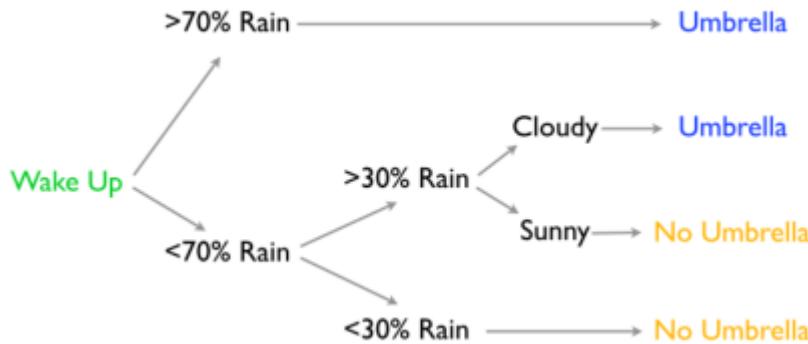
Tree-based methods:

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)

1.1 Decision Trees



1.2 Building Trees

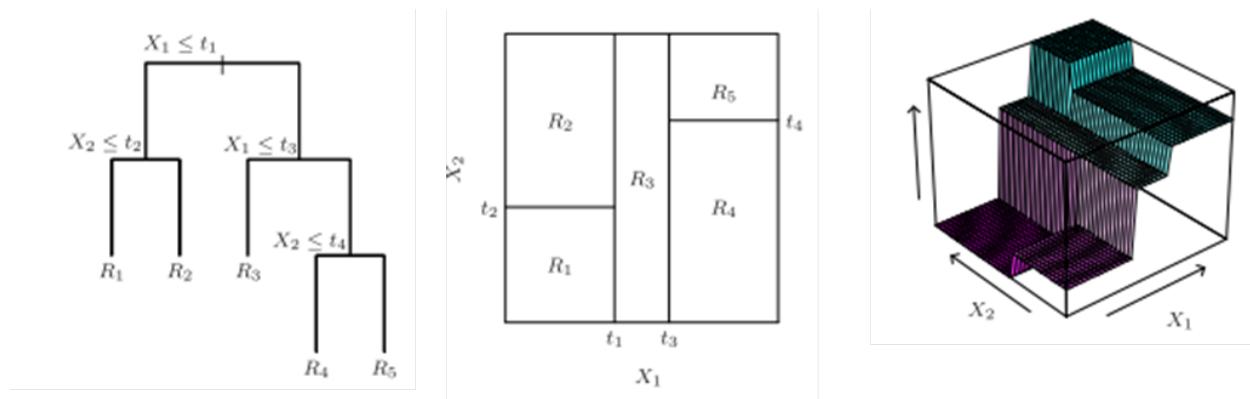
As usual, we want find the tree that minimizes some loss.

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

1.3 Recursive Binary Partition (CART)

Because the number of possible trees is too large to exhaustively search, 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/similarity/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. This is repeated until some stopping criteria is met (e.g., not enough observations in to split further). The terminal subgroup (leaf nodes) are used to make predictions.
 - 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 outcome variable.
 - There is another important difference: in CART **only a single feature** is used to determine the split into subgroups.

1.3.1 Model and Model Parameters

Trees model the outcome as a *constant* in each region

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

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

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

- Under log-loss (soft classification), the coefficients are probability *vectors* (one element for each class; sums to one).

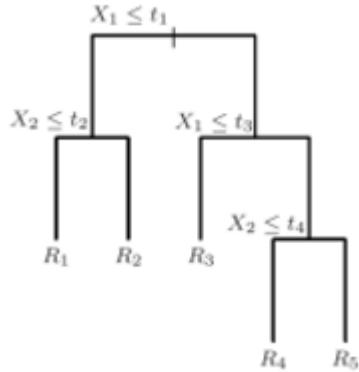
$$\begin{aligned} \hat{c}_{mk} &= \text{Proportion of class } k \text{ in region } R_m \\ &= \frac{1}{N_m} \sum_{i:\mathbf{x}_i \in R_m} \mathbb{1}(y_i = k) \end{aligned}$$

- Under 0-1 loss (hard classification), the coefficients are one-hot vectors.

$$\begin{aligned}\hat{c}_{mk} &= \text{One hot for majority class} \\ &= \mathbb{1}(k \text{ is majority class in region } R_m)\end{aligned}$$

- Other options possible; choose the coefficients to optimize your particular objective function.
Note: check the loss (implicitly) used in growing the tree!

1.3.2 Basis Expansion Interpretation



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

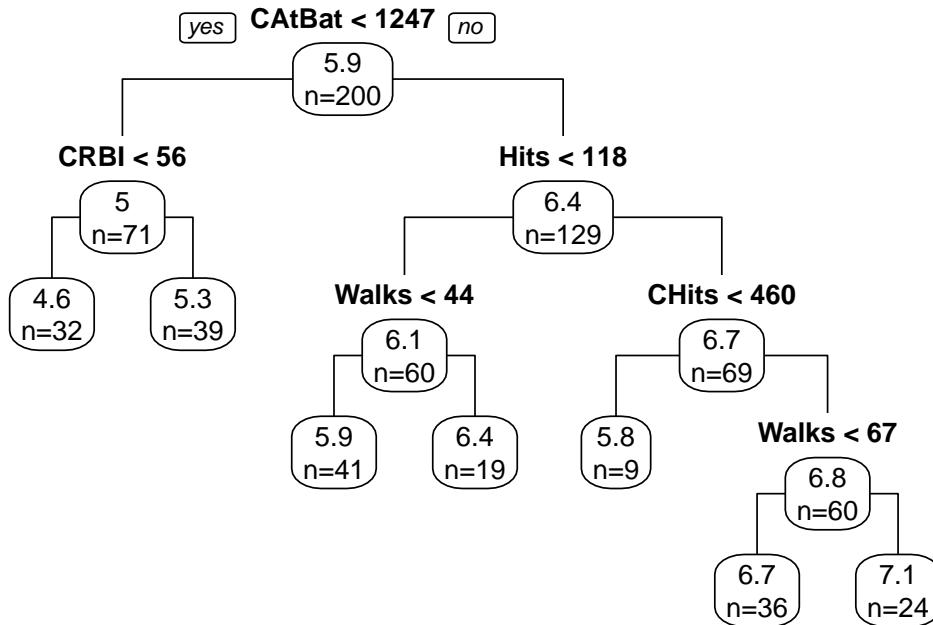
$$\begin{aligned}R_1 : b_1(x_1, x_2) &= \mathbb{1}(x_1 \leq t_1) \mathbb{1}(x_2 \leq t_2) \\R_2 : b_2(x_1, x_2) &= \mathbb{1}(x_1 \leq t_1) \mathbb{1}(x_2 > t_2) \\R_3 : b_3(x_1, x_2) &= \mathbb{1}(x_1 > t_1) \mathbb{1}(x_1 \leq t_3) \\R_4 : b_4(x_1, x_2) &= \mathbb{1}(x_1 > t_1) \mathbb{1}(x_1 > t_3) \mathbb{1}(x_2 \leq t_4) \\R_5 : b_5(x_1, x_2) &= \mathbb{1}(x_1 > t_1) \mathbb{1}(x_1 > t_3) \mathbb{1}(x_2 > t_4)\end{aligned}$$

1.3.3 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):



Your Turn #1 : Tree Interpretation

1. How many leaves are on the tree?
2. What do the numbers in the boxes mean?
3. How could you evaluate the prediction in a leaf node?

1.4 Growing a Tree

CART uses a greedy algorithm to grow a tree.

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

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

or

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

- 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

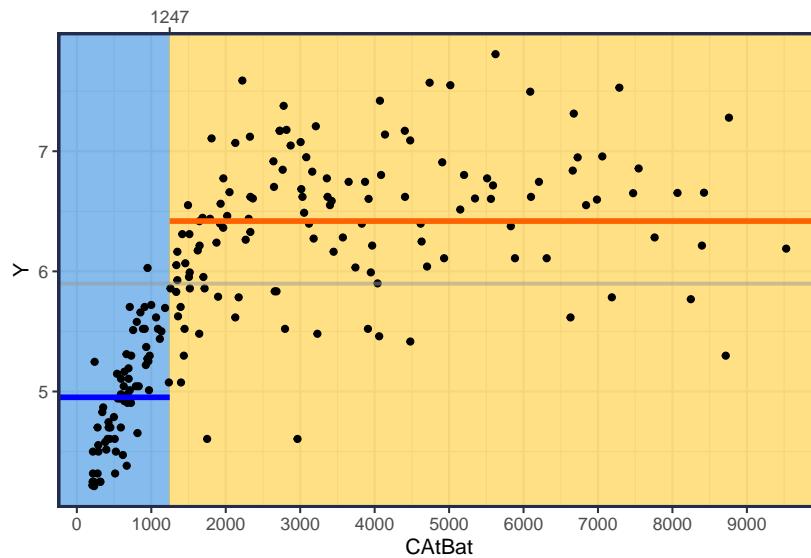
- Maximum depth (number of interactions)
- The final regions are called **leaf** nodes

1.5 Splitting Details

1.5.1 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 ($\text{CAtBat} < 1247$) is the best change point (change in mean) using a Gaussian model
 - An alternative perspective is to see that the reduction in MSE/SSE is maximized by splitting at ($\text{CAtBat} < 1247$) and fitting the data on each side of the split with a constant.

Splitting Details: Squared Error Loss

Notation

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

Consider a split on feature j :

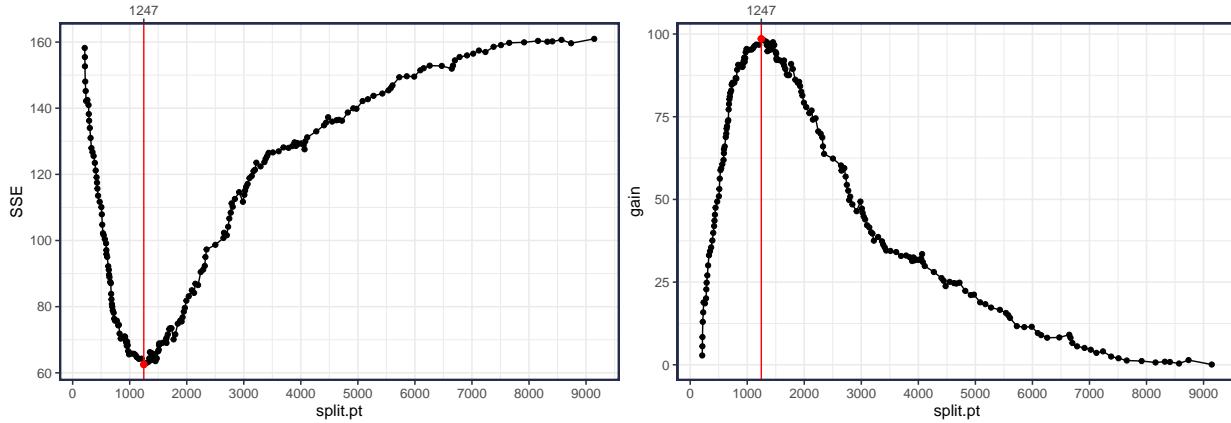
- Before split, the quality of the model, based on SSE is:

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

- Consider a split at s (on feature j)

<u>Left Region</u>	<u>Right Region</u>
$R_1(s) = \{x : x_j < s\}$	$R_2(s) = \{x : x_j \geq s\}$
$\bar{y}_1(s) = \frac{1}{n_1} \sum_{\{i : x_i \in R_1(s)\}} y_i$	$\bar{y}_2(s) = \frac{1}{n_2} \sum_{\{i : x_i \in R_2(s)\}} y_i$
$Q_1(s) = \sum_{\{i : x_i \in R_1(s)\}} (y_i - \bar{y}_1(s))^2$	$Q_2(s) = \sum_{\{i : x_i \in R_2(s)\}} (y_i - \bar{y}_2(s))^2$
<ul style="list-style-type: none"> • Updated SSE: $Q(s) = Q_1(s) + Q_2(s)$ • Gain(s) = $Q_0 - Q(s)$ 	

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

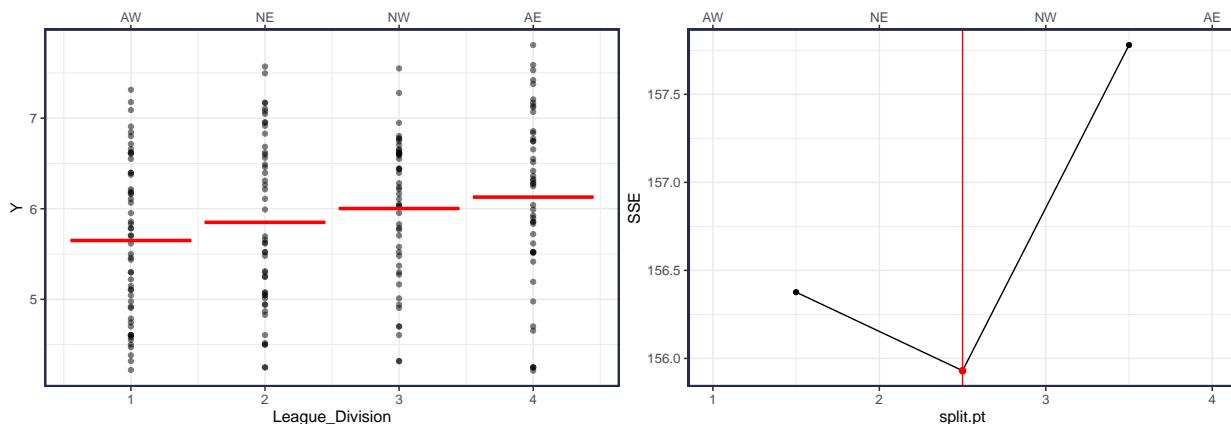


1.5.2 Regression Trees and Categorical (Nominal) Features

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

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

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



Note

- 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](#)

1.5.3 Classification Trees

A [classification tree](#) is used when the outcome is categorical $y \in \mathcal{G} = (1, 2, \dots, K)$.

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

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

- Each region has K -vector of probability estimates: $\bar{p}_m = [\bar{p}_m(1), \dots, \bar{p}_m(K)]$
 - $\sum_{k=1}^K \bar{p}_m(k) = 1$

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

1. **Misclassification Error:**

$$Q_m = 1 - \max_k \bar{p}_m(k)$$

2. **Gini Index:**

$$\begin{aligned}Q_m &= \sum_{k=1}^K \bar{p}_m(k)(1 - \bar{p}_m(k)) \\ &= 1 - \sum_{k=1}^K \bar{p}_m^2(k)\end{aligned}$$

3. **Cross-entropy/Deviance:**

$$\begin{aligned}Q_m &= - \sum_{k=1}^K \bar{p}_m(k) \log \bar{p}_m(k) \\ &= \sum_{k=1}^K \bar{p}_m(k) \log \frac{1}{\bar{p}_m(k)}\end{aligned}$$

1.5.4 Splitting Summary

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

$$(j^*, s^*) = \arg \min_{j,s} \text{Loss}(j, s) = \arg \max_{j,s} \text{Gain}(j, s)$$

- where $\text{Loss}(j, s)$ is the loss of splitting the current node on the j predictor at split point s .

1.6 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)

1.6.1 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. using sum of squared error loss,

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

- Weakest link pruning:** Successively collapse the internal node that produces the smallest per-node increase in $\sum_{m=1}^{|T|} Q_m(T)$ 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**

$$\begin{aligned} C_\lambda(T) &= \frac{1}{n} \sum_{m=1}^{|T|} Q_m(T) + \lambda|T| \\ &= \text{Loss}(T) + \lambda \text{Penalty}(T) \end{aligned}$$

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

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

1.6.2 Penalty Tuning

- For each λ , there is a unique smallest sub-tree T_λ that minimizes $C_\lambda(T)$.
- The sequence of sub-trees from weakest link pruning contains every T_λ
- The tuning parameter, λ can be chosen by: cross-validation, AIC/BIC, Out-of-Bag (OOB), etc.

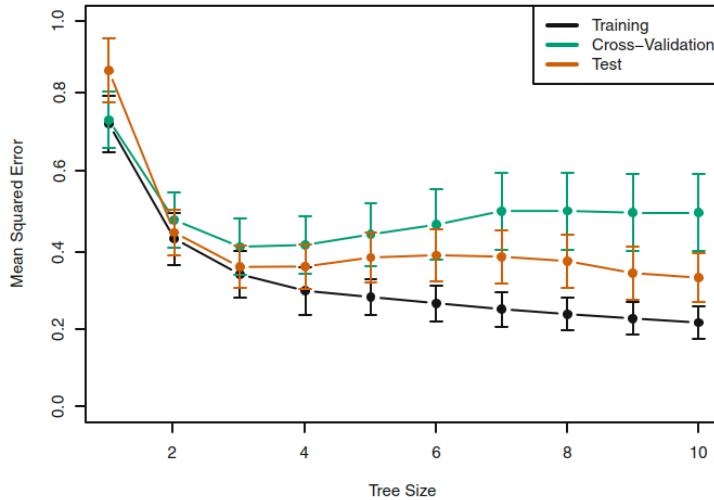


FIGURE 8.5. Regression tree analysis for the **Hitters** data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.

1.7 Special Considerations

1.7.1 Missing Predictor Values

- Omit observations with missing 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, before splitting we could impute (replace missing values with an estimate)
 - The surrogate approach is similar to imputation, but is based on the “nearest neighbors”; the observations in the same branch of the tree.

1.7.2 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.

1.7.3 Variable/Feature Importance

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

1. The number of times the feature was used to make a split

$$\mathcal{I}_j(T) = \sum_t \mathbb{1}(\text{split } t \text{ uses feature } j)$$

The sum is over all splits t in tree T .

2. 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

The importance of predictor j in a single tree T :

$$\mathcal{I}_j(T) = \sum_t \text{gain}(t) \cdot \mathbb{1}(\text{split } t \text{ uses feature } j)$$

That is, the importance of feature j in tree T is the total *gain* from all splits involving feature j . In the equation, the sum is over all splits t in tree T .

3. Permutation based importance: re-run the tree, but include a *permuted* version of the feature. Any decrease in performance indicates the feature was important.
 - Note: this is not perfect; consider what happens to features that are highly correlated/associated with other features

$$\mathcal{I}_j(T) = \text{Loss(Tree with permuted column } j) - \text{Loss(Tree)}$$

We will explore other ways to assess variable importance later in the course.

1.8 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. This is because trees split on sample quantiles rather than raw values.
- Easy to interpret

1.9 Tree Limitations

- Instability (high variance) due to greedy hierarchical structure; one change at top split can change remaining tree.
- Difficulty capturing additive structure

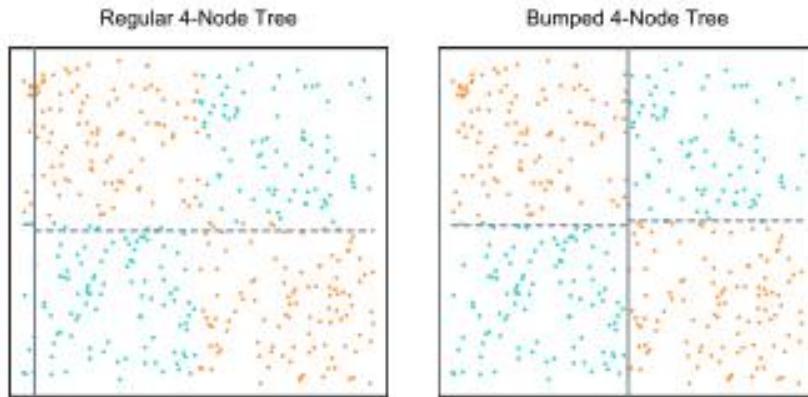


FIGURE 8.13. Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two 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.

1.10 Trees in R

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

2 Trees Demo

2.1 Required R Packages

```
library(ISLR)           # Hitters baseball data
library(rpart)          # classification and regression trees (CART)
library(rpart.plot)     # for `rpart()` which allows more plotting control for trees
library(randomForest)   # for `randomForest()` function
library(tidyverse)       # data manipulation and visualization
```

2.2 Baseball Salary Data

The goal is to build models to predict the (log) salary of baseball players

```
##-- Make Baseball Data
# Goal is to predict the log Salary

library(ISLR)
Hitters = ISLR::Hitters %>%
  filter(!is.na(Salary)) %>%      # remove missing Salary
  mutate(Salary = log(Salary)) %>%  # convert to log Salary
  rename(Y = Salary)

set.seed(2019) # choose 200 samples for training (leaving only 63 for testing)
train.ind = sample(nrow(Hitters), size=200)
bball = Hitters[train.ind, ]

#- test data
X.test = Hitters[-train.ind, ] %>% select(-Y)
Y.test = Hitters[-train.ind, ] %>% pull(Y)

bball %>% arrange(-Y) %>% head() %>% as_tibble(rownames = "name")
#> # A tibble: 6 x 21
#>   name      AtBat    Hits HmRun   Runs    RBI Walks Years CAtBat CHits CHmRun CRuns
#>   <chr>     <int>   <int> <int>   <int> <int> <int> <int> <int> <int> <int> <int>
#> 1 -Eddie Mu~    495    151     17     61     84     78     10    5624   1679    275    884
#> 2 -Jim Rice     618    200     20     98    110     62     13    7127   2163    351   1104
#> 3 -Mike Sch~     20      1      0      0      0      0      2      41      9      2      6
#> 4 -Don Matt~    677    238     31    117    113     53      5    2223    737     93    349
#> 5 -Ozzie Sm~    514    144      0     67     54     79      9    4739   1169     13    583
#> 6 -Gary Car~    490    125     24     81    105     62     13    6063   1646    271    847
#> # i 9 more variables: CRBI <int>, CWalks <int>, League <fct>, Division <fct>,
#> # PutOuts <int>, Assists <int>, Errors <int>, Y <dbl>, NewLeague <fct>
```

2.3 Regression Tree

```
#####
##-- Regression Trees in R
# trees are in many packages: rpart, tree, party, ...
# there are also many packages to display tree results
#
# Formulas: you don't need to specify interactions as the tree does this
# naturally.
#####
##-- Build Tree
```

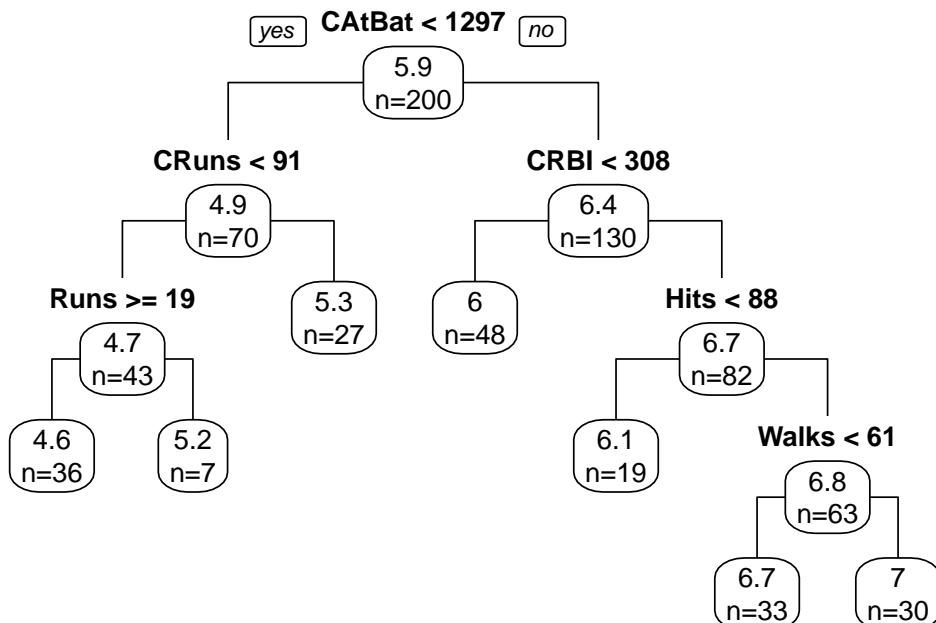
```

library(rpart)
tree = rpart(Y~., data=bball)
summary(tree, cp=1)
#> Call:
#> rpart(formula = Y ~ ., data = bball)
#> n= 200
#>
#>      CP nsplit rel error xerror      xstd
#> 1 0.59642      0    1.0000 1.0095 0.07330
#> 2 0.07158      1    0.4036 0.4329 0.05933
#> 3 0.04991      2    0.3320 0.4145 0.06130
#> 4 0.03923      3    0.2821 0.4044 0.06976
#> 5 0.01258      4    0.2429 0.3365 0.07088
#> 6 0.01051      5    0.2303 0.3419 0.06920
#> 7 0.01000      6    0.2198 0.3418 0.06836
#>
#> Variable importance
#> CAtBat   CHits   CRuns    CRBI   CWalks   CHmRun    Hits   AtBat    Runs    RBI    Walks
#>     18       17      17       16      14       11        2       1       1       1       1
#>
#> Node number 1: 200 observations
#> mean=5.911, MSE=0.8476
length(unique(tree$where))           # number of leaf nodes
#> [1] 7

-- Plot Tree
library(rpart.plot)    # for prp() which allows more plotting control
prp(tree, type=1, extra=1, branch=1)

# rpart() functions can also plot (just not as good):
# plot(tree, uniform=TRUE)
# text(tree, use.n=TRUE, xpd=TRUE)

```



2.3.2 Evaluate Tree

```
#- mean squared error function
mse <- function(yhat, y){
  yhat = as.matrix(yhat)
  apply(yhat, 2, \((f) mean((y-f)^2)) )
}

mse(predict(tree), bball$Y)           # training error
#> [1] 0.1863
mse(predict(tree, X.test), Y.test)    # testing error
#> [1] 0.2942
```

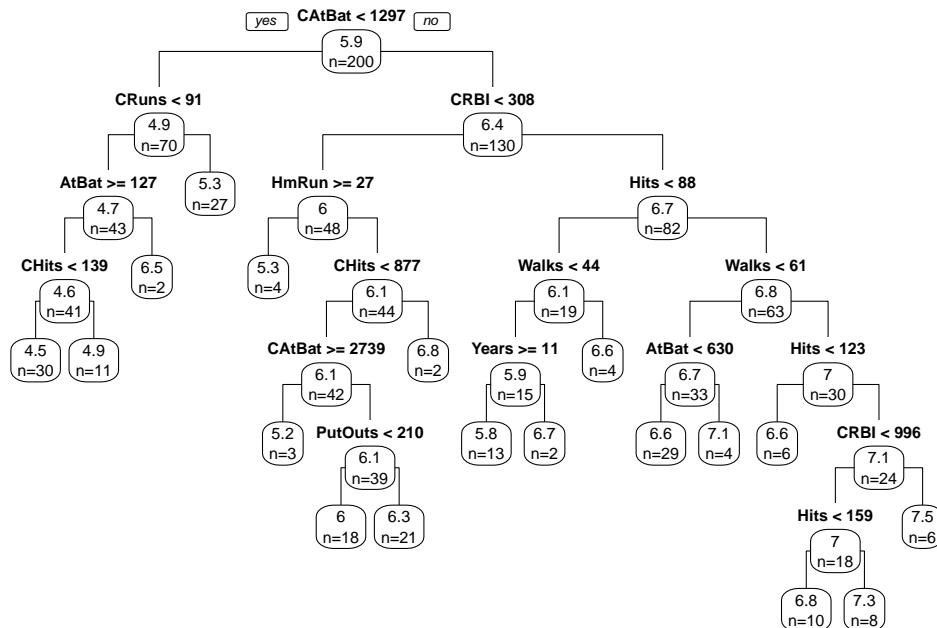
Build a more complex tree

```
-- More complex tree
# see ?rpart.control() for details
# xval: number of cross-validations
# minsplit: min obs to still allow a split
# cp: complexity parameter

tree2 = rpart(Y ~ ., data=bball, xval=0, minsplit=5, cp=0.005)
summary(tree2, cp=1)
#> Call:
#> rpart(formula = Y ~ ., data = bball, xval = 0, minsplit = 5,
#>       cp = 0.005)
#>     n= 200
#>
#>          CP nsplit rel error
#> 1  0.596422      0   1.0000
#> 2  0.071577      1   0.4036
#> 3  0.049912      2   0.3320
#> 4  0.039235      3   0.2821
#> 5  0.038202      4   0.2429
#> 6  0.012689      5   0.2047
#> 7  0.012583      6   0.1920
#> 8  0.010402      7   0.1794
#> 9  0.008953      9   0.1586
#> 10 0.008834     10  0.1496
#> 11 0.008015     11  0.1408
#> 12 0.007319     12  0.1328
#> 13 0.005964     13  0.1255
#> 14 0.005708     14  0.1195
#> 15 0.005103     15  0.1138
#> 16 0.005031     16  0.1087
#> 17 0.005000     17  0.1036
#>
#> Variable importance
#>   CATBat   CRuns   CHits    CRBI   CWalks   CHmRun    Hits   AtBat   Runs   Walks
#>     17      17      16      15      13      10        3       3      2       2
#>   RBI   Years   HmRun PutOuts
#>     1       1      1       1
#>
#> Node number 1: 200 observations
#>   mean=5.911, MSE=0.8476
length(unique(tree2$where))
#> [1] 18
```

```
prp(tree2, type=1, extra=1, branch=1)

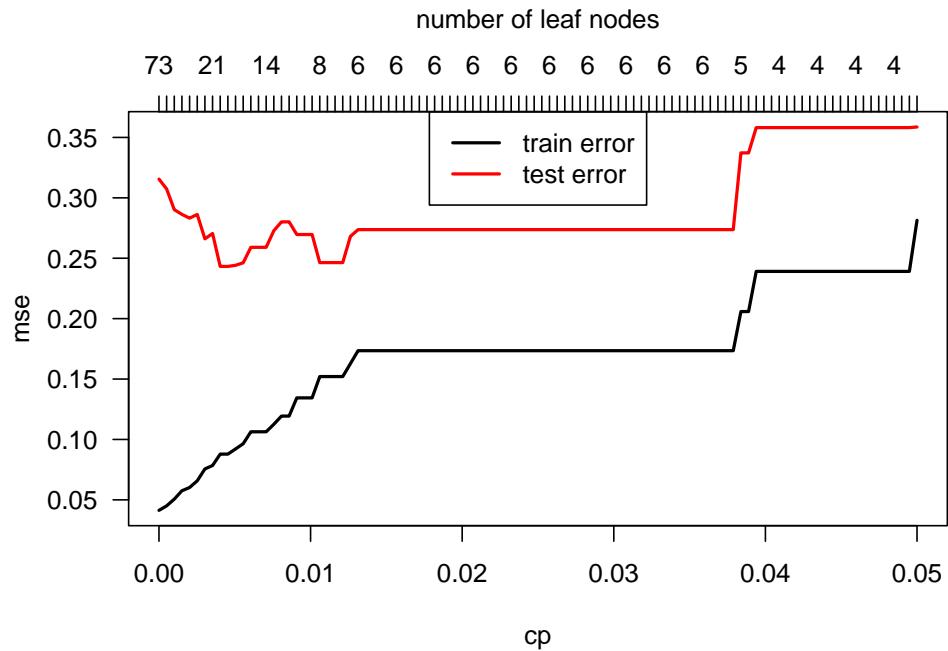
mse(predict(tree2), bball$Y)                      # training error
#> [1] 0.08785
mse(predict(tree2, X.test), Y.test)               # testing error
#> [1] 0.2432
```



Now, fit a set of trees for sequence of cp values.

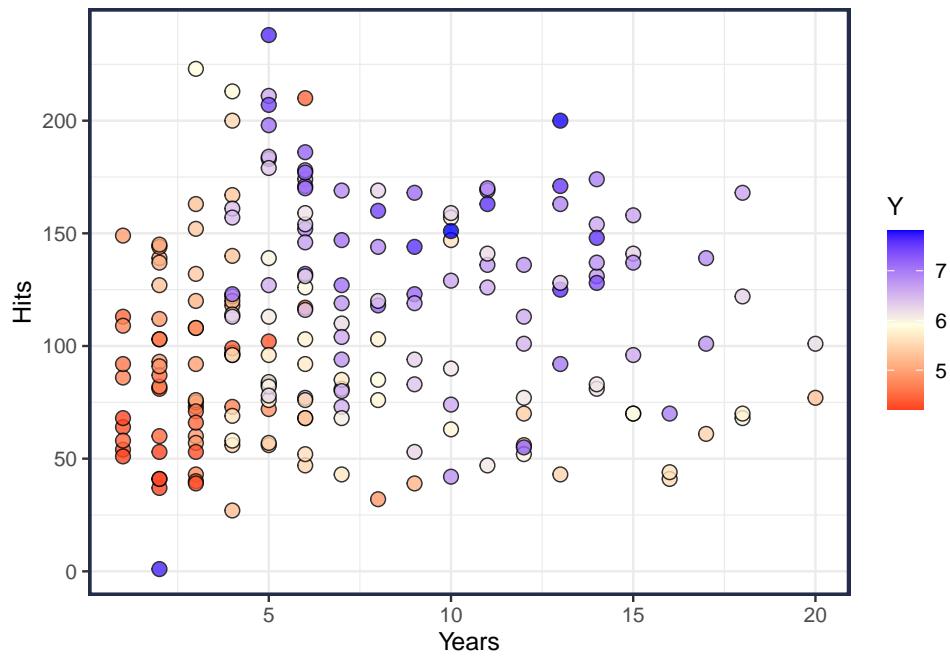
```
cp = seq(.05,0,length=100)  # cp is like a penalty on the tree size
for(i in 1:length(cp)){
  if(i == 1){train.error = test.error = nleafs = numeric(length(cp)) }
  tree.fit = rpart(Y~.,data=bball, xval=0, minsplit=5, cp=cp[i])
  train.error[i] = mse(predict(tree.fit),bball$Y)                  # training error
  test.error[i] = mse(predict(tree.fit,X.test),Y.test)             # testing error
  nleafs[i] = length(unique(tree.fit$where))
}

plot(range(cp),range(train.error,test.error),typ='n',xlab="cp",ylab="mse",las=1)
lines(cp,train.error,col="black",lwd=2)
lines(cp,test.error,col="red",lwd=2)
legend("top",c('train error','test error'),col=c("black","red"),lwd=2)
axis(3,at=cp,labels=nleafs)
mtext("number of leaf nodes",3,line=2.5)
```

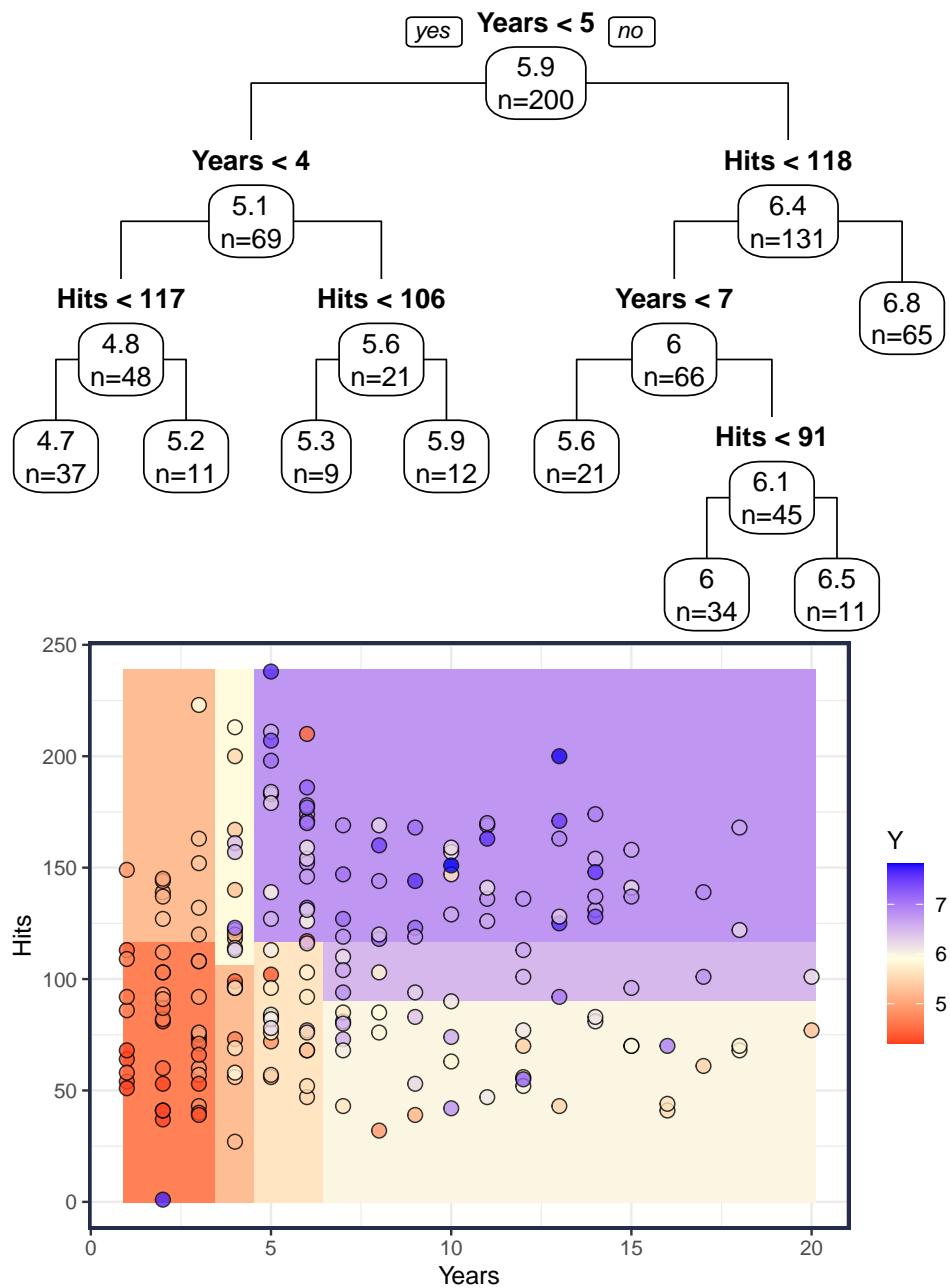


2.3.3 Regression Tree example with 2 dimensions only

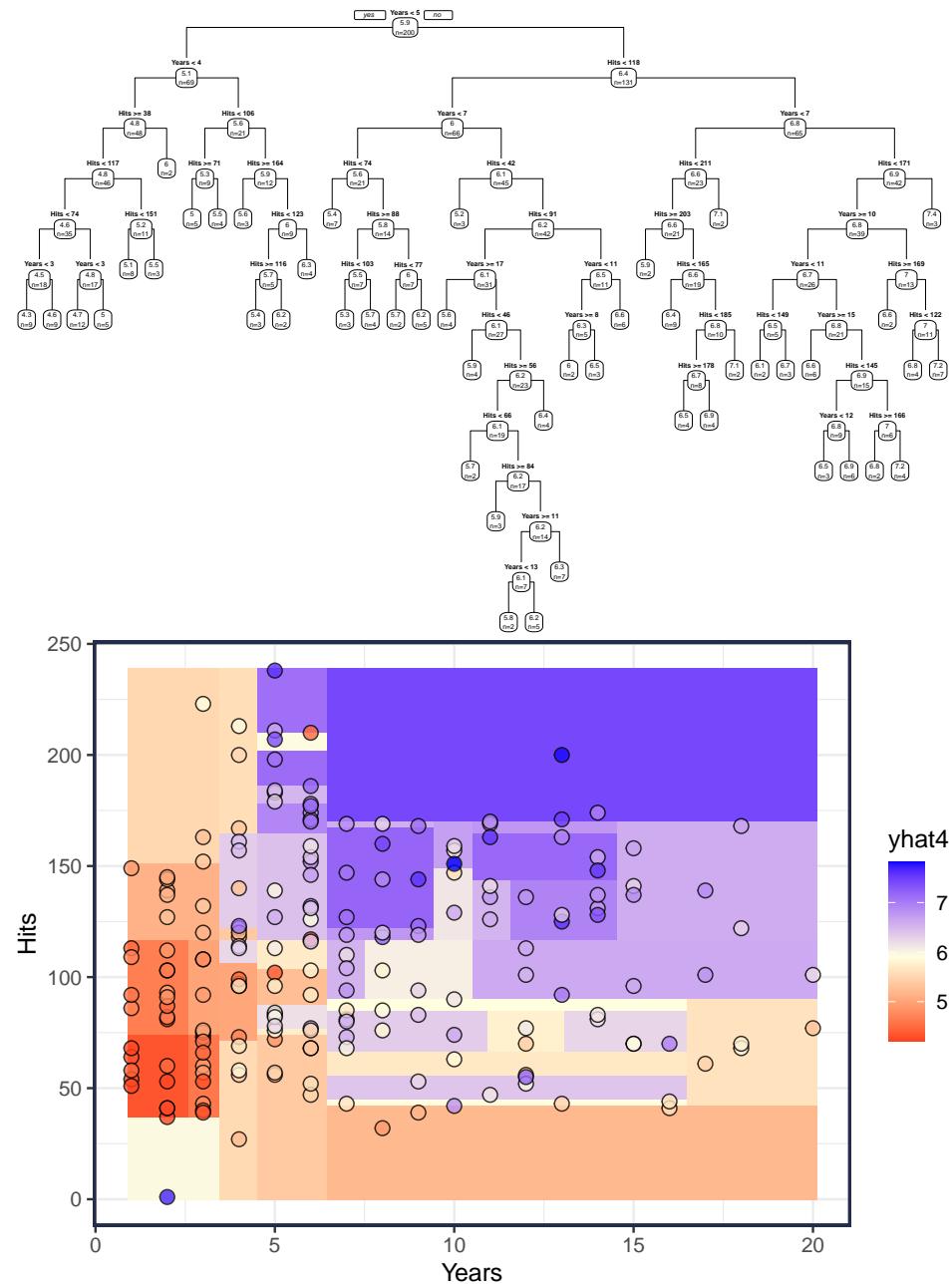
Consider the two variables Years and Hits and their relationship to Y.



Let's fit a tree with the two predictors

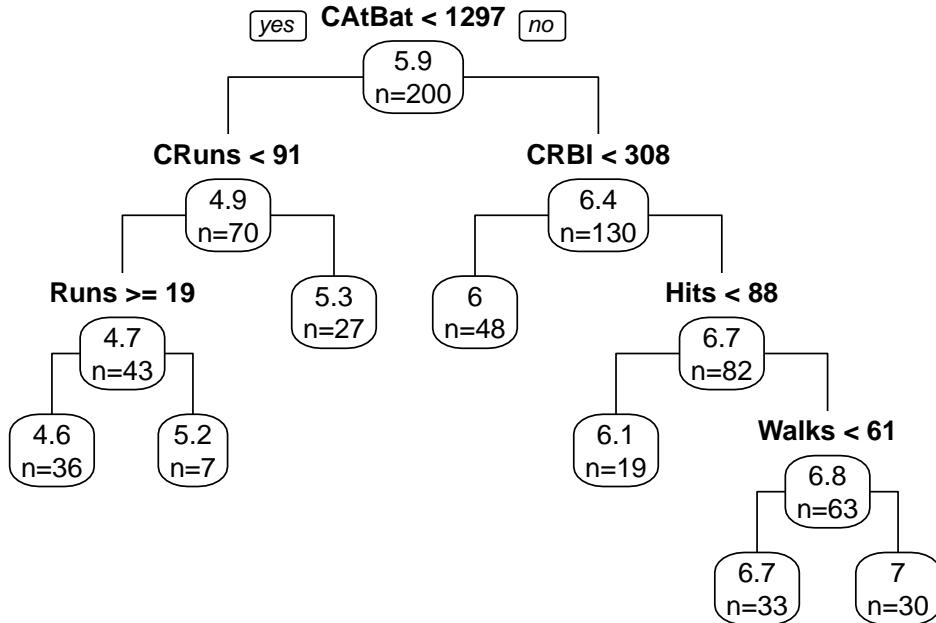


And we can also use more complex trees:



2.4 Details of Splitting (for Regression Trees)

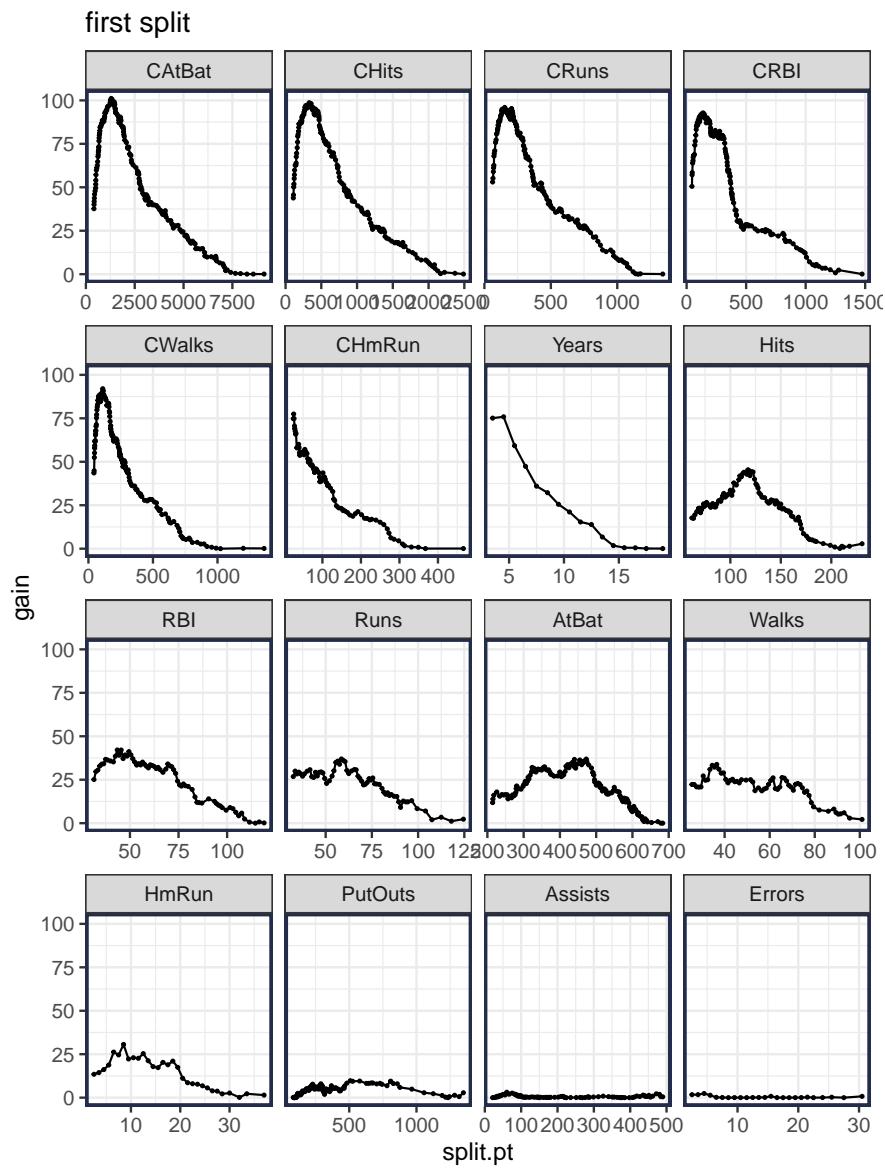
Going back to building a tree with all predictor variables. Notice that `CAtBat` is split at 1296.5.



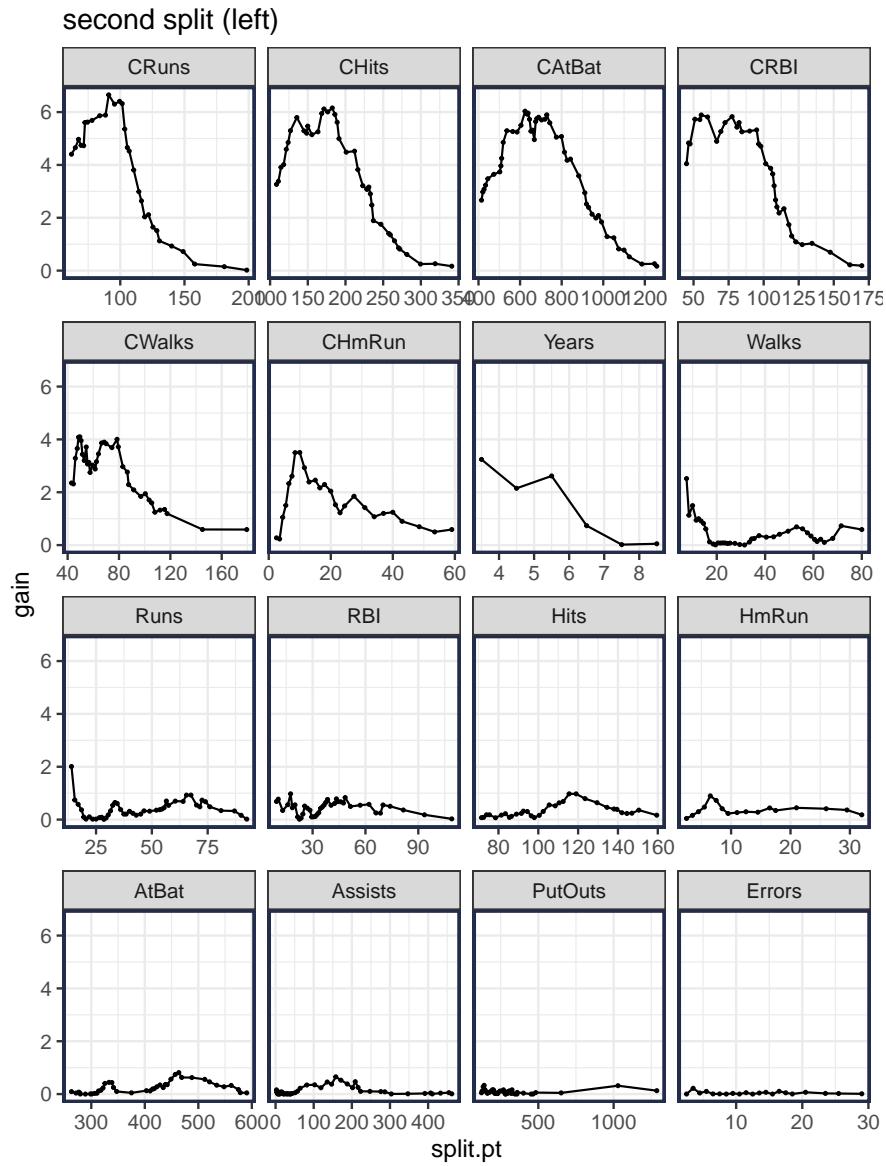
2.4.1 First Split

Under the hood, trees will search all possibly split points for all predictor variables. It will use the variable and associated split point that has the maximum gain (or improvement in evaluation metric).

var	split.pt	n.L	n.R	est.L	est.R	SSE.L	SSE.R	SSE	gain
CAtBat	1296.5	70	130	4.942	6.433	20.06	48.35	68.42	101.109
CHits	331.0	71	129	4.964	6.432	21.93	48.93	70.86	98.661
CRuns	153.0	68	132	4.946	6.408	21.02	52.56	73.58	95.950
CRBI	140.0	73	127	5.013	6.427	26.54	50.24	76.79	92.738
CWalks	111.0	74	126	5.026	6.431	27.59	49.88	77.47	92.053
CHmRun	25.5	75	125	5.107	6.393	40.31	51.74	92.06	77.470
Years	4.5	69	131	5.062	6.358	32.33	61.31	93.64	75.881
Hits	117.5	114	86	5.497	6.459	73.89	50.25	124.14	45.389
RBI	43.5	83	117	5.366	6.297	55.13	72.29	127.42	42.109
Runs	58.5	113	87	5.534	6.401	78.66	53.93	132.59	36.932
AtBat	472.5	123	77	5.571	6.454	85.27	47.37	132.63	36.893
Walks	36.5	95	105	5.479	6.302	62.42	73.28	135.71	33.820
HmRun	8.5	94	106	5.495	6.279	69.21	69.70	138.91	30.611
PutOuts	508.0	174	26	5.826	6.482	141.56	18.22	159.79	9.737
Assists	60.5	111	89	5.799	6.050	89.94	76.47	166.40	3.121
Errors	4.5	71	129	5.763	5.992	47.98	119.15	167.13	2.396

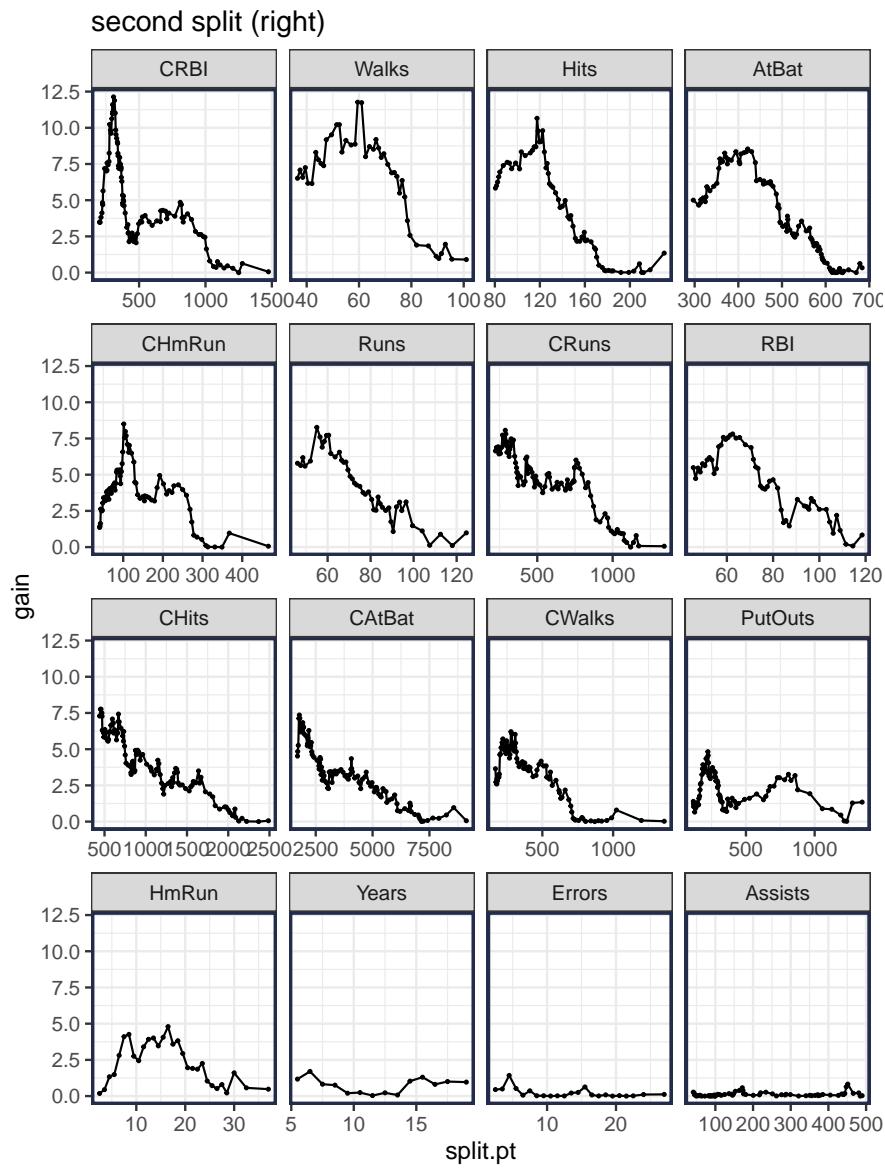


2.4.2 Second Split (left): CRuns < 91



var	split.pt	n.L	n.R	est.L	est.R	SSE.L	SSE.R	SSE	gain
CRuns	91.0	43	27	4.698	5.331	12.136	1.274	13.41	6.651
CHits	182.5	45	25	4.721	5.340	12.713	1.193	13.91	6.155
CAtBat	624.0	35	35	4.648	5.236	11.302	2.725	14.03	6.034
CRBI	55.5	35	35	4.652	5.232	11.434	2.737	14.17	5.890
CWalks	49.5	33	37	4.686	5.170	11.784	4.179	15.96	4.099
CHmRun	10.0	35	35	4.718	5.166	12.749	3.807	16.56	3.505
Years	3.5	46	24	4.787	5.240	13.689	3.131	16.82	3.241
Walks	7.5	3	67	5.838	4.902	5.198	12.345	17.54	2.518
Runs	14.0	3	67	5.742	4.906	5.963	12.090	18.05	2.008
RBI	17.5	7	63	5.296	4.903	8.443	10.643	19.09	0.975
Hits	115.5	57	13	4.886	5.189	18.843	0.243	19.09	0.974
HmRun	6.5	40	30	4.844	5.073	16.212	2.953	19.16	0.896
AtBat	463.5	58	12	4.893	5.179	19.021	0.227	19.25	0.813
Assists	158.0	51	19	4.883	5.100	10.136	9.272	19.41	0.653
PutOuts	140.5	25	45	5.034	4.891	13.077	6.653	19.73	0.331
Errors	3.5	22	48	4.859	4.980	4.290	15.553	19.84	0.218

2.4.3 Second Split (right): CRBI < 307.5



var	split.pt	n.L	n.R	est.L	est.R	SSE.L	SSE.R	SSE	gain
CRBI	307.5	48	82	6.033	6.666	10.419	25.802	36.22	12.134
Walks	59.5	92	38	6.239	6.901	27.106	9.482	36.59	11.767
Hits	117.5	57	73	6.109	6.686	12.411	25.290	37.70	10.655
AtBat	421.5	51	79	6.114	6.639	9.444	30.373	39.82	8.538
CHmRun	101.5	86	44	6.250	6.790	26.250	13.605	39.85	8.500
Runs	55.0	52	78	6.124	6.639	12.518	27.569	40.09	8.268
CRuns	288.0	39	91	6.052	6.596	6.912	33.377	40.29	8.066
RBI	62.5	78	52	6.232	6.733	19.475	21.068	40.54	7.812
CHits	457.5	23	107	5.906	6.546	3.459	37.133	40.59	7.763
CAtBat	1779.5	25	105	5.945	6.549	3.913	37.086	41.00	7.356
CWalks	278.5	68	62	6.224	6.662	20.745	21.402	42.15	6.209
PutOuts	223.0	57	73	6.215	6.603	16.219	27.311	43.53	4.825
HmRun	16.5	83	47	6.288	6.688	24.000	19.559	43.56	4.797
Years	6.5	47	83	6.281	6.519	18.510	28.144	46.65	1.702
Errors	4.5	44	86	6.286	6.508	10.663	36.267	46.93	1.425
Assists	451.5	125	5	6.417	6.831	46.372	1.158	47.53	0.825

3 Bagging Trees

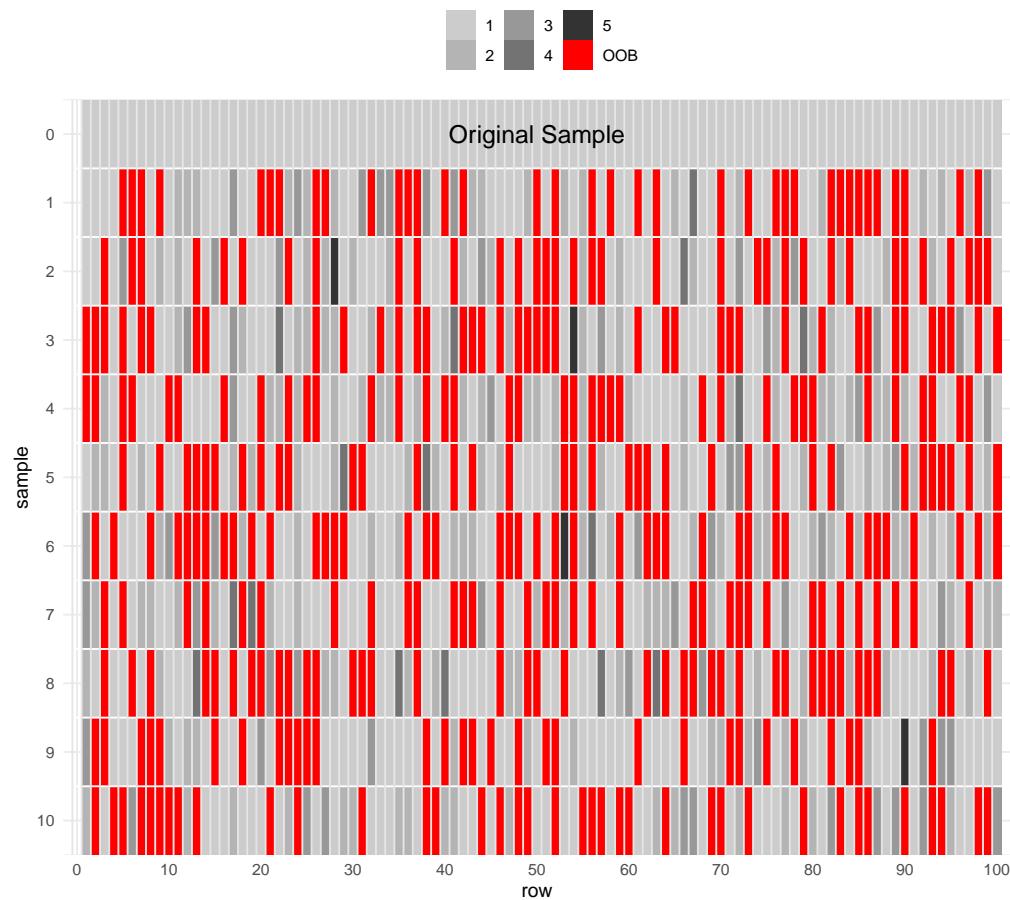
3.1 Better Trees

- 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 $\hat{\theta}_b = \hat{\theta}(\mathcal{D}_b)$ are all the estimated parameters for fitting the tree (split variables, cutpoints, and terminal (leaf) node values) from the bootstrap sample \mathcal{D}_b .

- Details and discussion can be found in Breiman's article "Bagging Predictors" (1996, *Machine Learning*).
 - Bagging = Bootstrap Aggregating
 - 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:
 - For regression: use the average predictions
 - 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)



3.1.1 Variance Reduction with Bagging

Note

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

Properties of Variance/Covariance

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

Variance Reduction

- Let θ be something we want to estimate (e.g., $\theta = f(x)$) and $\hat{\theta}$ an estimate.
- Suppose we have M models to estimate θ which produces the estimates $\{\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_M\}$
- 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:

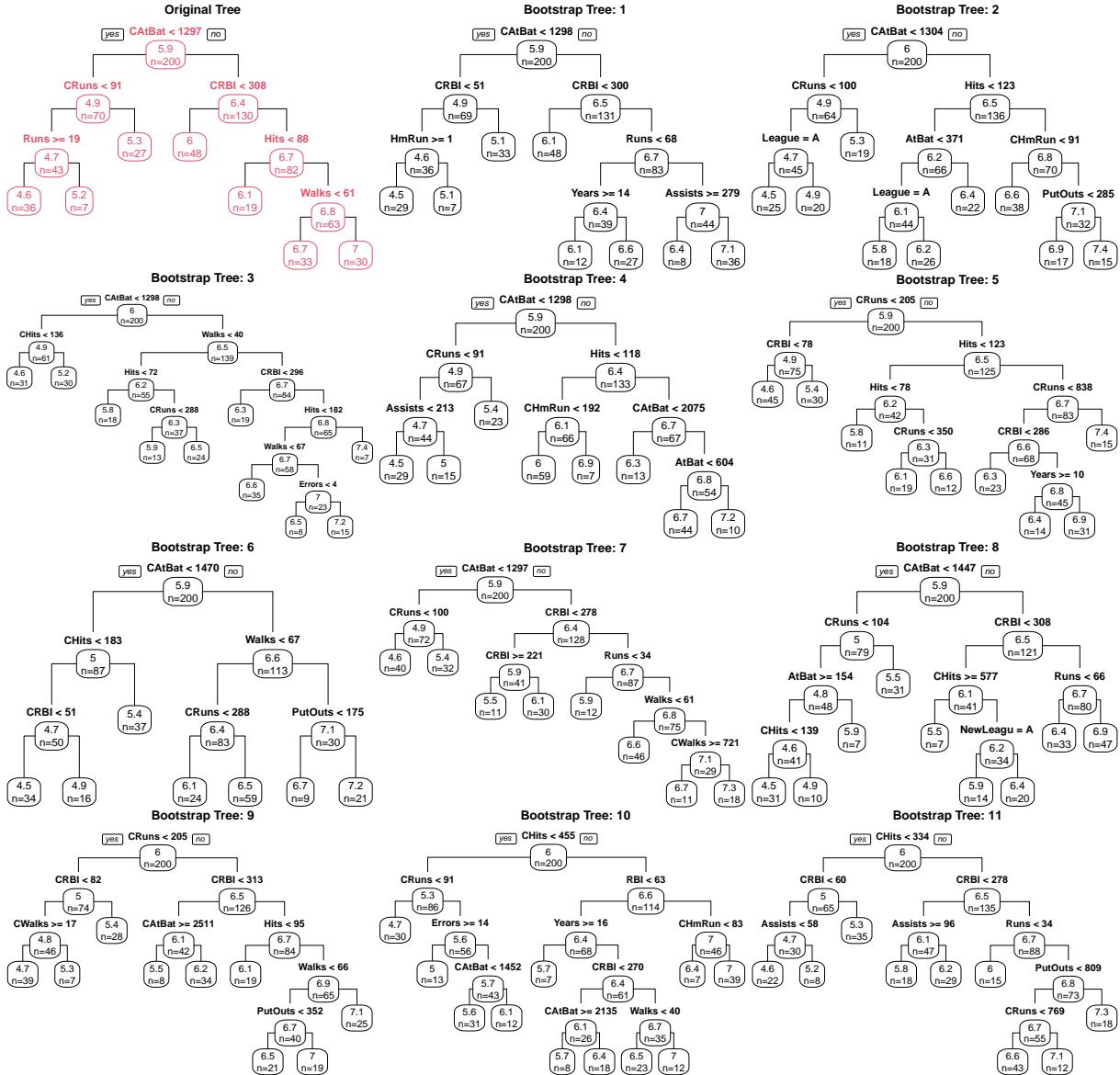
$$E(\bar{\theta}) = \frac{1}{M} \sum_{i=1}^M E(\hat{\theta}_i)$$

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

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

- Thus to reduce the variance, we want to **use models that have low correlation**.
 - If $\text{Cor}(\hat{\theta}_i, \hat{\theta}_j) = 0 \quad \forall i, j$, then variance is minimized (for example, when the models are *independent*)
 - If $\text{Cor}(\hat{\theta}_i, \hat{\theta}_j) = 1 \quad \forall i, j$, 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.

3.2 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

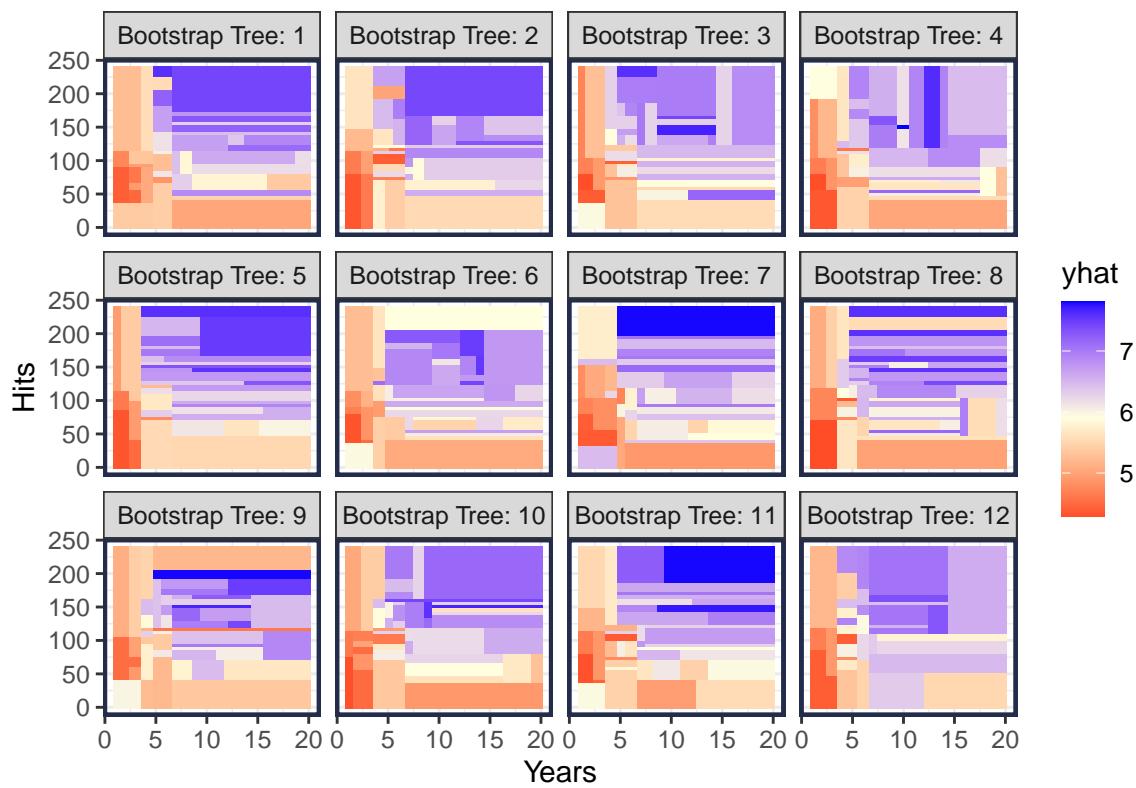


Figure 1: Bagging in 2 dimensions (Years and Hits)

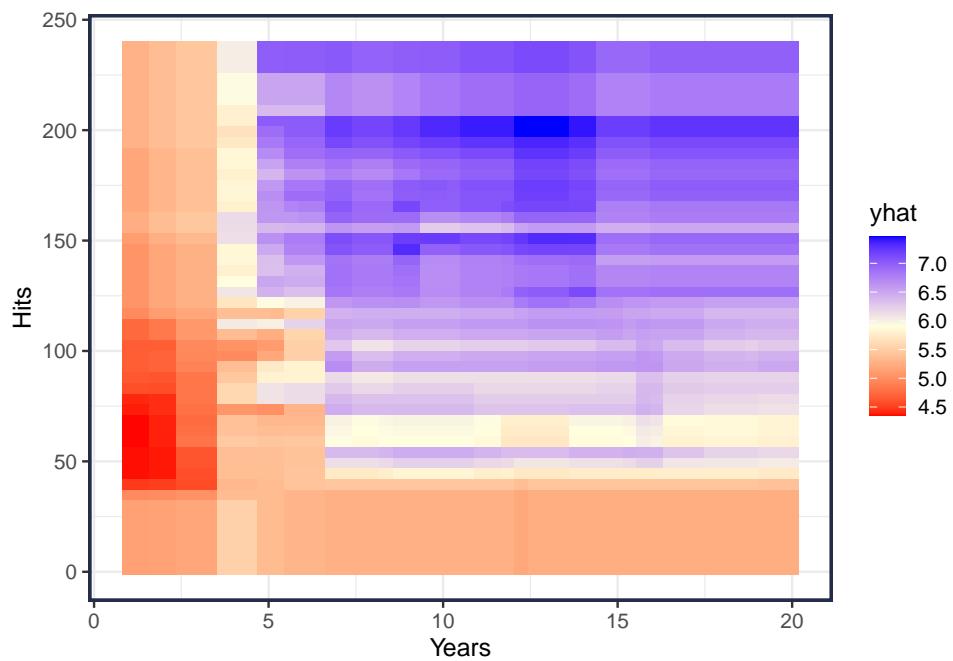


Figure 2: Average of bootstrap predictions

4 Random Forest

4.1 Random Forest

Random Forest is a modification of bagging that attempts to build *de-correlated trees* by considering a restricted set of features for splitting.

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 :

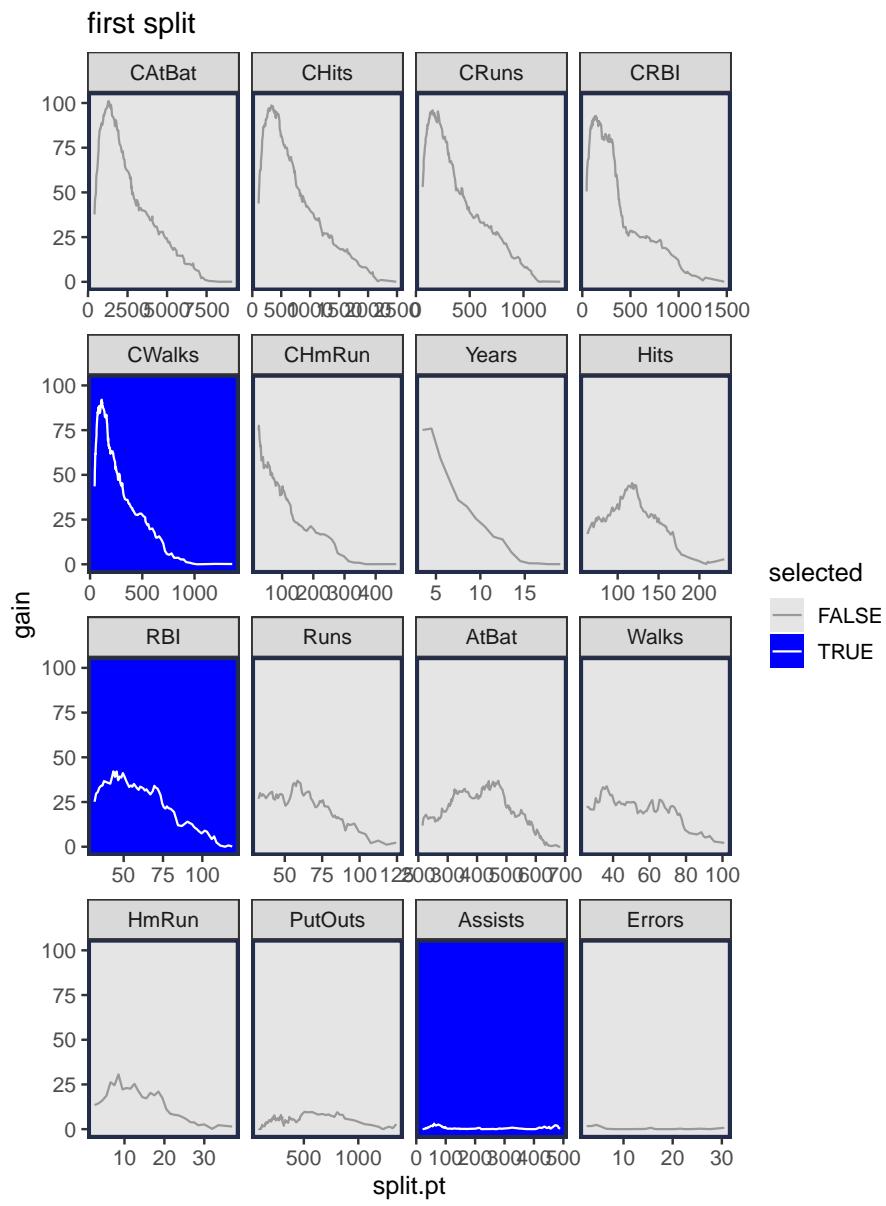
$$\text{Regression: } \hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x).$$

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

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

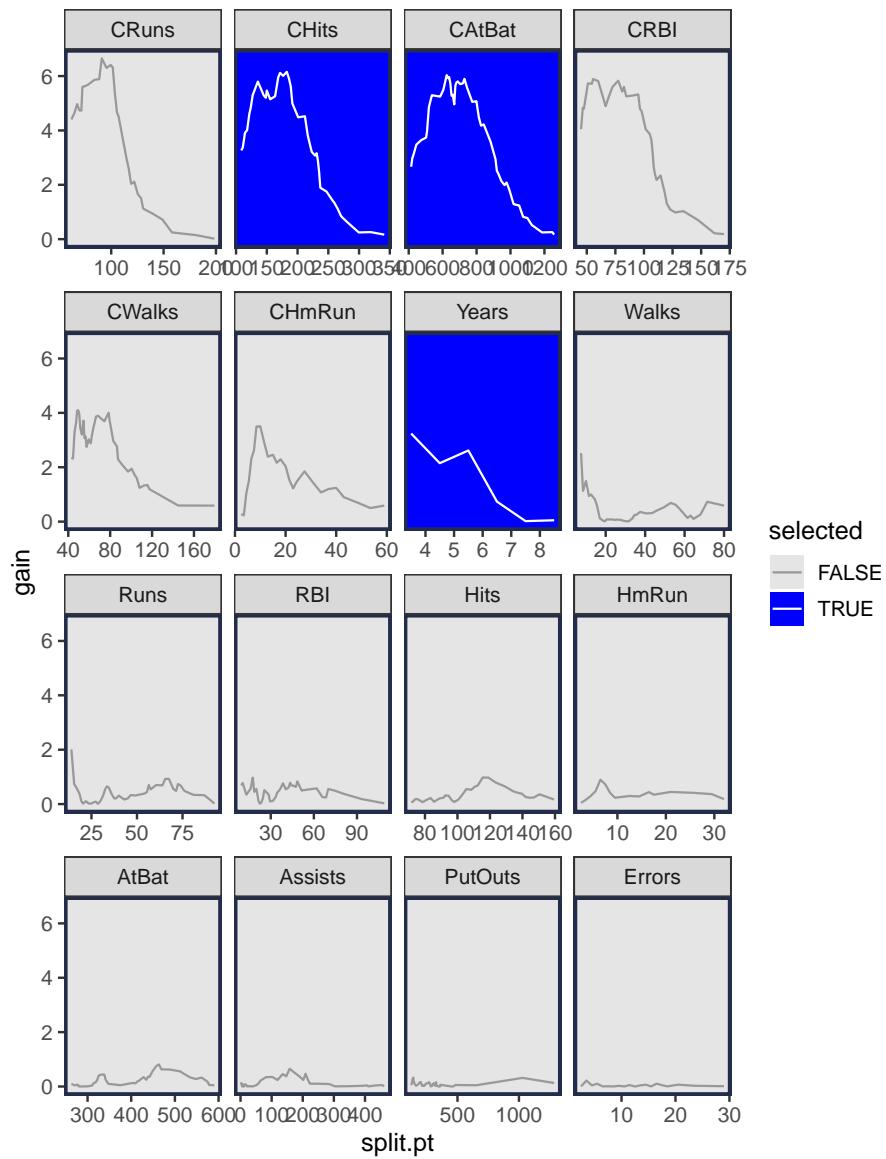
4.1.1 Illustration of Restricted Set of Features for Splitting

var	split.pt	n.L	n.R	est.L	est.R	SSE.L	SSE.R	SSE	gain
CAtBat	1296.5	70	130	4.942	6.433	20.06	48.35	68.42	101.109
CHits	331.0	71	129	4.964	6.432	21.93	48.93	70.86	98.661
CRuns	153.0	68	132	4.946	6.408	21.02	52.56	73.58	95.950
CRBI	140.0	73	127	5.013	6.427	26.54	50.24	76.79	92.738
CWalks	111.0	74	126	5.026	6.431	27.59	49.88	77.47	92.053
CHmRun	25.5	75	125	5.107	6.393	40.31	51.74	92.06	77.470
Years	4.5	69	131	5.062	6.358	32.33	61.31	93.64	75.881
Hits	117.5	114	86	5.497	6.459	73.89	50.25	124.14	45.389
RBI	43.5	83	117	5.366	6.297	55.13	72.29	127.42	42.109
Runs	58.5	113	87	5.534	6.401	78.66	53.93	132.59	36.932
AtBat	472.5	123	77	5.571	6.454	85.27	47.37	132.63	36.893
Walks	36.5	95	105	5.479	6.302	62.42	73.28	135.71	33.820
HmRun	8.5	94	106	5.495	6.279	69.21	69.70	138.91	30.611
PutOuts	508.0	174	26	5.826	6.482	141.56	18.22	159.79	9.737
Assists	60.5	111	89	5.799	6.050	89.94	76.47	166.40	3.121
Errors	4.5	71	129	5.763	5.992	47.98	119.15	167.13	2.396



var	split.pt	n.L	n.R	est.L	est.R	SSE.L	SSE.R	SSE	gain
CRuns	91.0	43	27	4.698	5.331	12.136	1.274	13.41	6.651
CHits	182.5	45	25	4.721	5.340	12.713	1.193	13.91	6.155
CAtBat	624.0	35	35	4.648	5.236	11.302	2.725	14.03	6.034
CRBI	55.5	35	35	4.652	5.232	11.434	2.737	14.17	5.890
CWalks	49.5	33	37	4.686	5.170	11.784	4.179	15.96	4.099
CHmRun	10.0	35	35	4.718	5.166	12.749	3.807	16.56	3.505
Years	3.5	46	24	4.787	5.240	13.689	3.131	16.82	3.241
Walks	7.5	3	67	5.838	4.902	5.198	12.345	17.54	2.518
Runs	14.0	3	67	5.742	4.906	5.963	12.090	18.05	2.008
RBI	17.5	7	63	5.296	4.903	8.443	10.643	19.09	0.975
Hits	115.5	57	13	4.886	5.189	18.843	0.243	19.09	0.974
HmRun	6.5	40	30	4.844	5.073	16.212	2.953	19.16	0.896
AtBat	463.5	58	12	4.893	5.179	19.021	0.227	19.25	0.813
Assists	158.0	51	19	4.883	5.100	10.136	9.272	19.41	0.653
PutOuts	140.5	25	45	5.034	4.891	13.077	6.653	19.73	0.331
Errors	3.5	22	48	4.859	4.980	4.290	15.553	19.84	0.218

second split: left



4.2 Random Forest Tuning

There are two primary tuning parameters for Random Forest:

1. Variety: m controls the number of predictors that are evaluated for each split (this is named `mtry` argument in `randomForest` package)
2. Complexity: The depth/size of the trees are controlled by setting the *minimum number of observations in the leaf nodes* (`min.obs`) or the *depth* of the tree or the number of leaf nodes

Your Turn #2

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

- The tuning parameters can be determined from cross-validation or OOB error
- In `randomForest` package:
 - For classification, the default value is $mtry = \lfloor \sqrt{p} \rfloor$ and $min.obs = 1$.
 - For regression, the default value is $mtry = \lfloor p/3 \rfloor$ and $min.obs = 5$.
- The *number of trees* is another tuning parameter, but want this to be as large as possible (subject to computational and memory constraints)
 - See [This stats.stackexchange answer](#) for further explanation.

4.3 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.

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

where $N_B(i)$ is the number of trees with observation i out-of-bag.

- Recall that there is a 37% chance that any observation is out-of-bag in any bootstrap sample.
- Thus, $N_B(i) \approx 0.37B$ (the number of trees used to estimate the OOB error is about 37% of the total number of trees in the forest).
 - More encouragement to use *many* trees in the forest

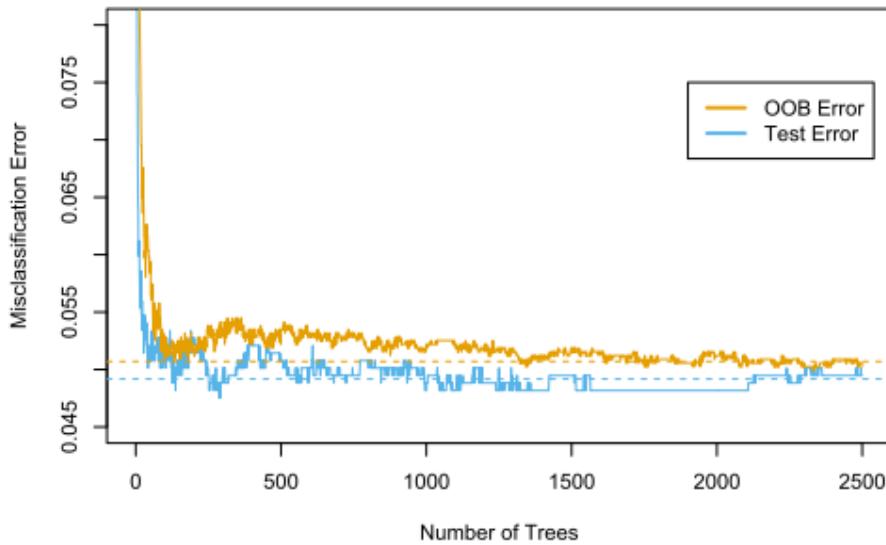


Figure 15.4 in ESL

4.4 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 :

$$\mathcal{I}_j(T) = \sum_t \text{gain}(t) \cdot \mathbb{1}(\text{split } t \text{ uses feature } j)$$

That is, the importance of feature j in tree T is the total *gain* from all splits involving feature j . In the equation, the sum is over all splits t in tree T .

The importance of predictor j in a forest is the average importance from all trees in the forest:

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

- Note: a final normalizing step may transform importance scores to sum to 1
- There are other ways to measure feature importance, like permutation.

4.5 Random Forest and k-NN

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

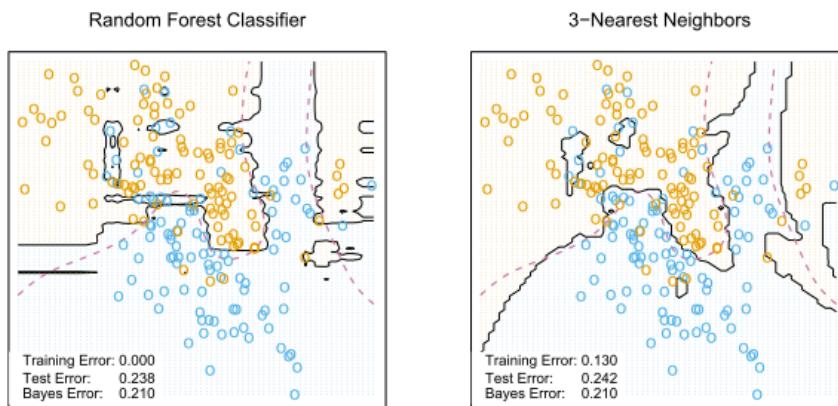


FIGURE 15.11. Random forests versus 3-NN on the mixture data. The axis-oriented nature of the individual trees in a random forest lead to decision regions with an axis-oriented flavor.

4.6 Random Forests in R

- `randomForest`
- `ranger`
- `randomForestSRC`
- `Rborist`
- `party`
- `tidymodels` https://parsnip.tidymodels.org/reference/rand_forest.html