Prediction Trees

CART, Bagging, and Random Forest

SYS 6018 | Spring 2022

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 . . . 12
- 1.10 Trees in R . . . 12

2 Trees Demo **13**
- 2.1 Required R Packages . . . 13
- 2.2 Baseball Salary Data . . . 13
- 2.3 Regression Tree . . . 13
- 2.4 Details of Splitting (for Regression Trees) . . . 21

3 Bagging Trees **27**
- 3.1 Better Trees . . . 27
- 3.2 Bagging Trees . . . 29

4 Random Forest **31**
- 4.1 Random Forest . . . 31
- 4.2 Random Forest Tuning . . . 32
- 4.3 OOB error . . . 32
- 4.4 Variable Importance . . . 33
- 4.5 Random Forest and k-NN . . . 33

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:

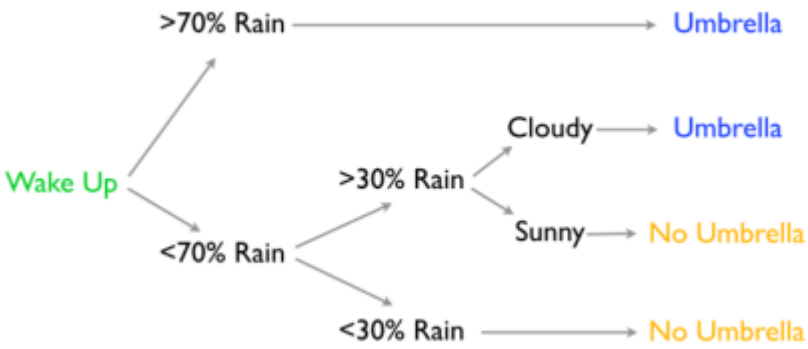
1. Partition the feature space into a set of (hyper) rectangles
2. Fit a simple model (e.g., constant) in each one.

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

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



1.2 Building Trees

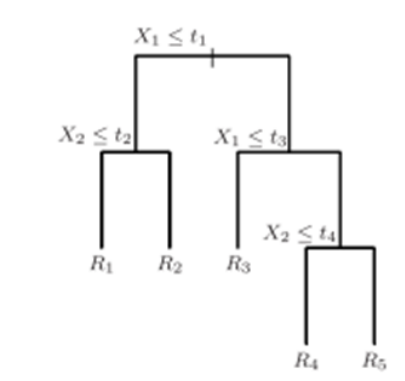
As usual, we want find the tree that minimizes some loss.

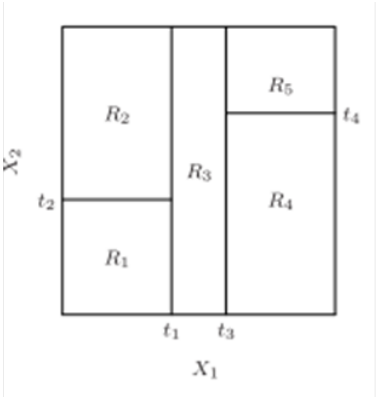
- Classification trees have class probabilities at the leaves (e.g., the probability I'll be in heavy rain is 0.9).
  - E.g., Binomial likelihood.
- Regression trees have a mean response at the leaves. (e.g., the expected amount of rain is 2in).
  - E.g., squared error.

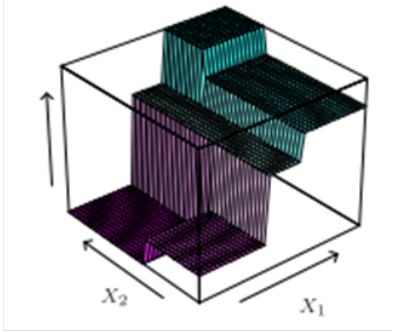
1.3 Recursive Binary Partition (CART)

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

1.3.1 Model and Model Parameters

Model the response 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*) $R_1, \ldots, R_M$
  - The coefficients/scores for the regions $c_1, \ldots, c_M$
- 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; sum to one).

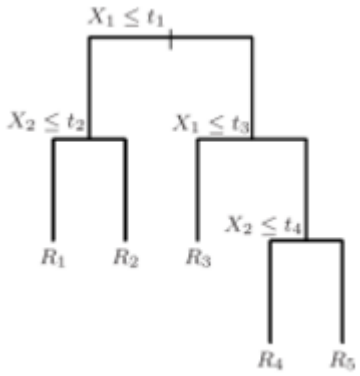
$$\begin{aligned}\hat{c}_{mk} &= \text{Proportion of class } k \text{ in region } R_m \\ &= \frac{1}{N_m} \sum_{i:\mathbf{x}_i \in R_m} \mathbb{1}(y_i = k)\end{aligned}$$

- 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



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

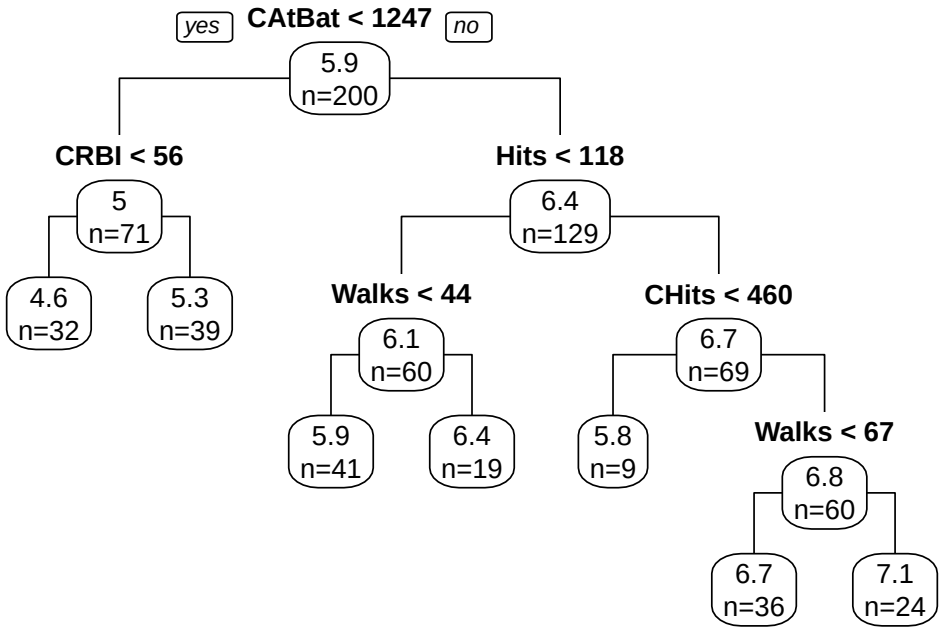
$$\begin{aligned}b_1(x_1, x_2) &= \mathbb{1}(x_1 \le t_1)\,\mathbb{1}(x_2 \le t_2) \\ b_2(x_1, x_2) &= \mathbb{1}(x_1 \le t_1)\,\mathbb{1}(x_2 > t_2) \\ b_3(x_1, x_2) &= \mathbb{1}(x_1 > t_1)\,\mathbb{1}(x_1 \le t_3) \\ b_4(x_1, x_2) &= \mathbb{1}(x_1 > t_1)\,\mathbb{1}(x_1 > t_3)\,\mathbb{1}(x_2 \le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

1.3.3 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 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:

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

    or

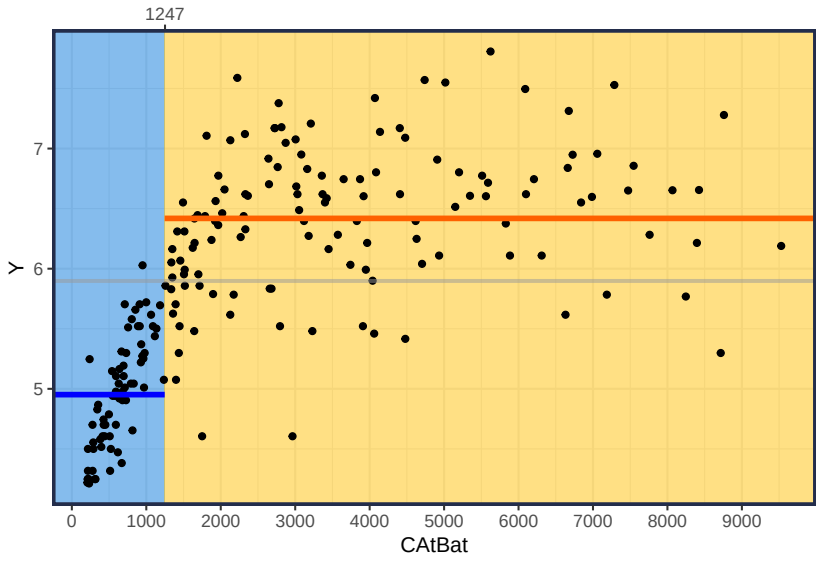
    $R_1(j,t) = \{x : x_j \in A_j\}$ and $R_2(j,t) = \{x : x_j \notin A_j\}$ *Nominal/Categorical Feature*
- Repeat this step for each child region
- Continue until stopping criteria met, e.g.
  - Minimum number of observations in region
  - Loss function has minimal improvement
- The final regions are called leaf nodes

1.5 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 (`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 (`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

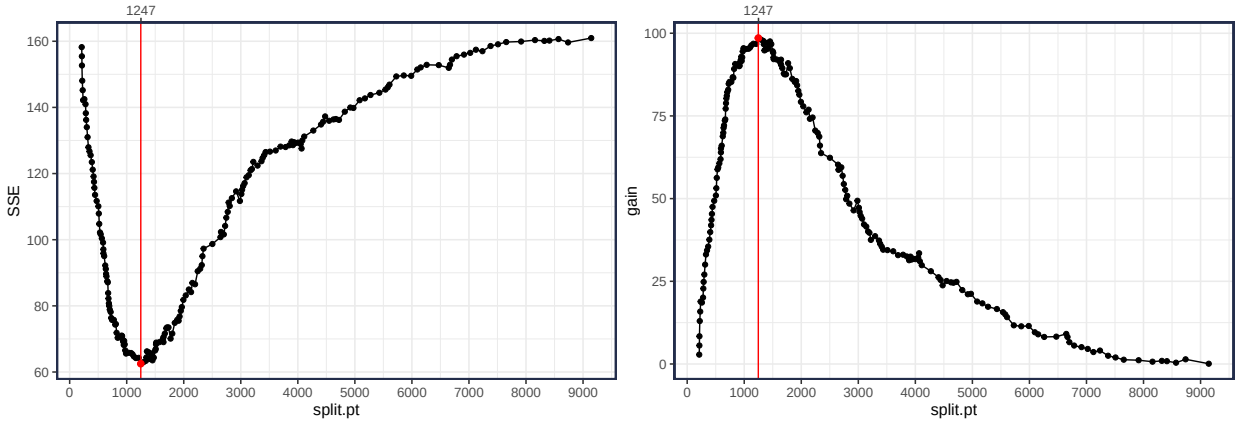
$$R_1(s) = \{x : x_j < s\}$$
$$\bar{y}_1(s) = \frac{1}{n_1} \sum_{\{i:x_i \in R_1(s)\}} y_i$$
$$Q_1(s) = \sum_{\{i:x_i \in R_1(s)\}} (y_i - \bar{y}_1(s))^2$$

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:

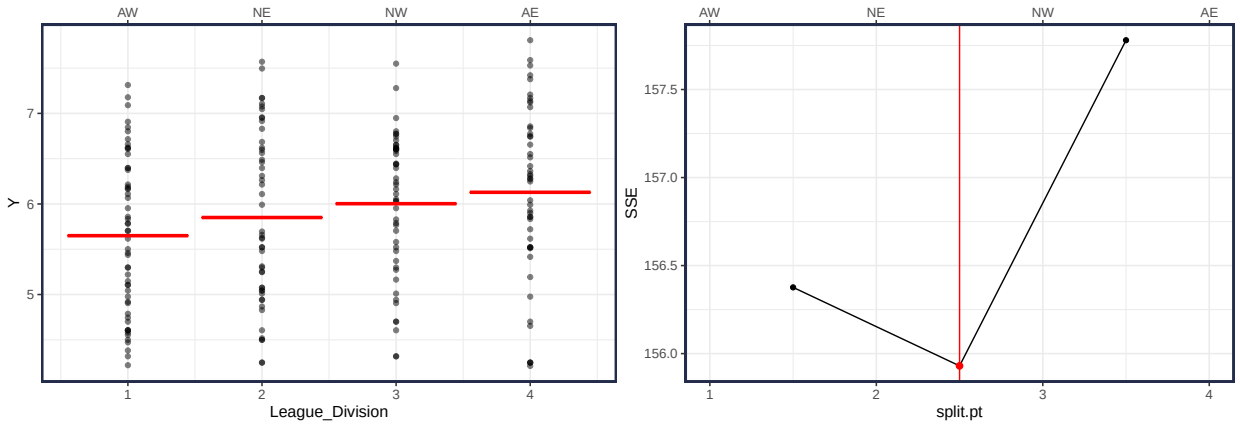


1.5.2 Regression Trees and Categorical (Nominal) Features

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 $y \in \mathcal{G} = (1, 2, \ldots, K)$.

- 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: $\bar{p}_m = [\bar{p}_m(1), \ldots, \bar{p}_m(K)]$
  - $\sum_{k=1}^{K} \bar{p}_m(k) = 1$

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

1.5.4 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 minimizes the loss (or maximizes the gain):

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

- 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.,

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

- *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.

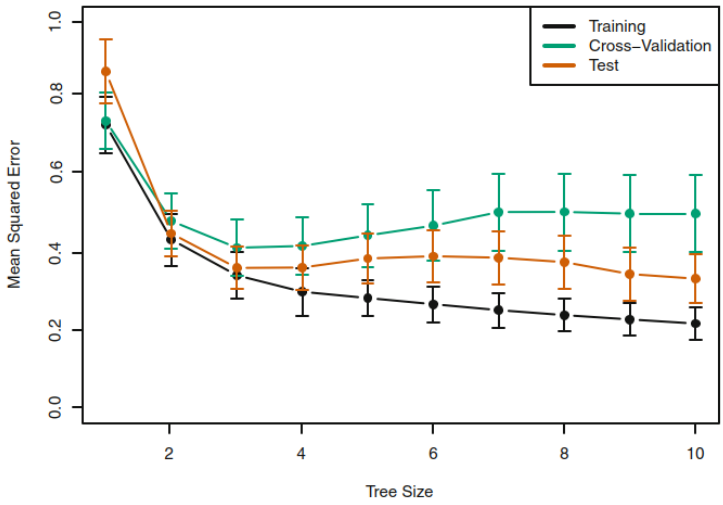


FIGURE 8.5. *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

1.7 Special Considerations

1.7.1 Missing Predictor Values

- 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 i) omit observations with missing values or (ii) impute
  - The surrogate approach is similar to imputation, but is based on the "nearest neighbors"; the observations in the same branch of the tree.

1.7.2 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

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

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 (splitting on sample quantiles)
- Easy to interpret

1.9 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

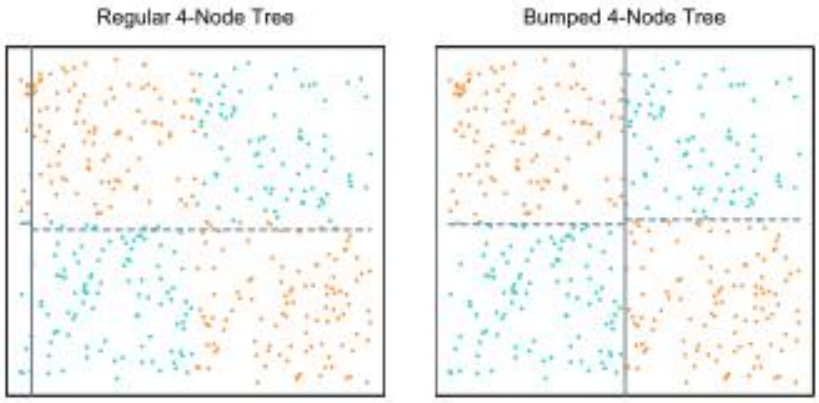


FIGURE 8.13. *Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two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

1.10 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 `prp()` 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 = sample(c(rep(TRUE, 200), rep(FALSE, nrow(Hitters)-200)))
bball = Hitters[train, ]

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

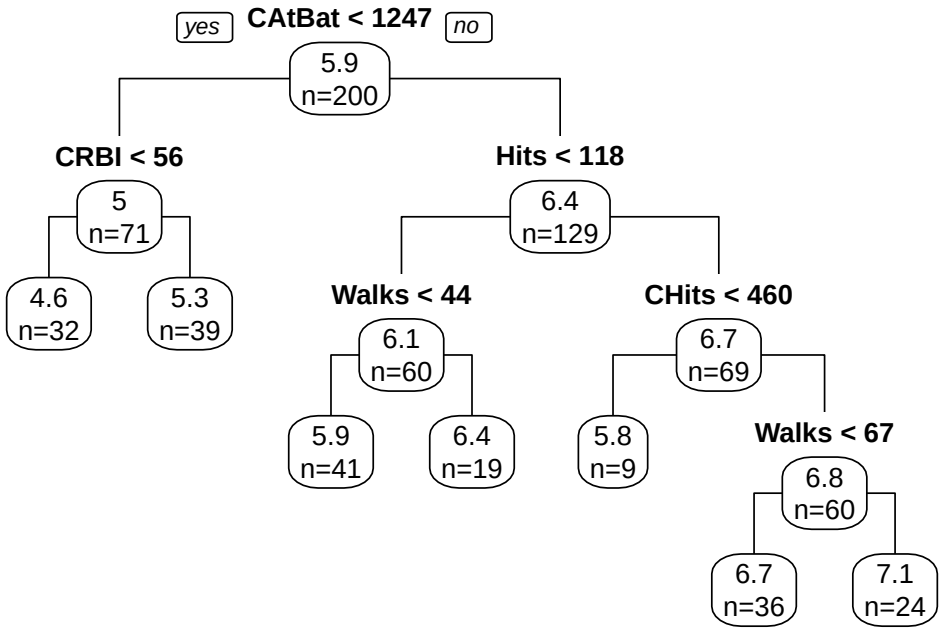
```
head(bball)
#> # A tibble: 6 x 20
#>   AtBat  Hits HmRun  Runs   RBI Walks Years CAtBat CHits CHmRun CRuns  CRBI
#>   <int> <int> <int> <int> <int> <int> <int>  <int> <int>  <int> <int> <int>
#> 1   315    81     7    24    38    39    14   3449   835     69   321   414
#> 2   479   130    18    66    72    76     3   1624   457     63   224   266
#> 3   321    87    10    39    42    30     2    396   101     12    48    46
#> 4   594   169     4    74    51    35    11   4408  1133     19   501   336
#> 5   185    37     1    23     8    21     2    214    42      1    30     9
#> 6   298    73     0    24    24     7     3    509   108      0    41    37
#> # ... with 8 more variables: CWalks <int>, League <fct>, Division <fct>,
#> #   PutOuts <int>, Assists <int>, Errors <int>, Y <dbl>, NewLeague <fct>
```

2.3 Regression Tree

2.3.1 Build 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 0.61187      0    1.0000 1.0034 0.07223
#> 2 0.08077      1    0.3881 0.4361 0.04334
#> 3 0.05616      2    0.3074 0.3754 0.04487
#> 4 0.05037      3    0.2512 0.3534 0.04483
#> 5 0.01840      4    0.2008 0.2500 0.02677
#> 6 0.01381      5    0.1824 0.2553 0.02637
#> 7 0.01000      6    0.1686 0.2433 0.02594
#>
#> Variable importance
#> CAtBat  CHits  CRuns   CRBI CWalks CHmRun   Hits  AtBat   Runs  Walks    RBI
#>     17     17     16     15     14     11      2      2      2      2      1
#>  HmRun
#>      1
#>
#> Node number 1: 200 observations
#>   mean=5.898, MSE=0.8053
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)
```



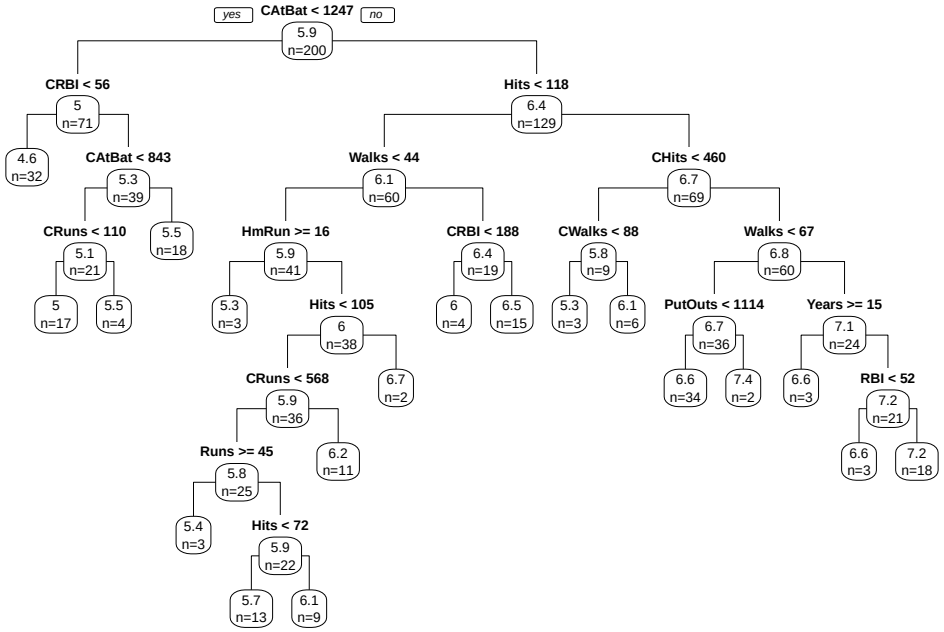
2.3.2 Evaluate Tree

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


mse(predict(tree), bball$Y)            # training error
#> [1] 0.1358
mse(predict(tree, X.test), Y.test)     # testing error
#> [1] 0.4931
```

Build a more complex tree

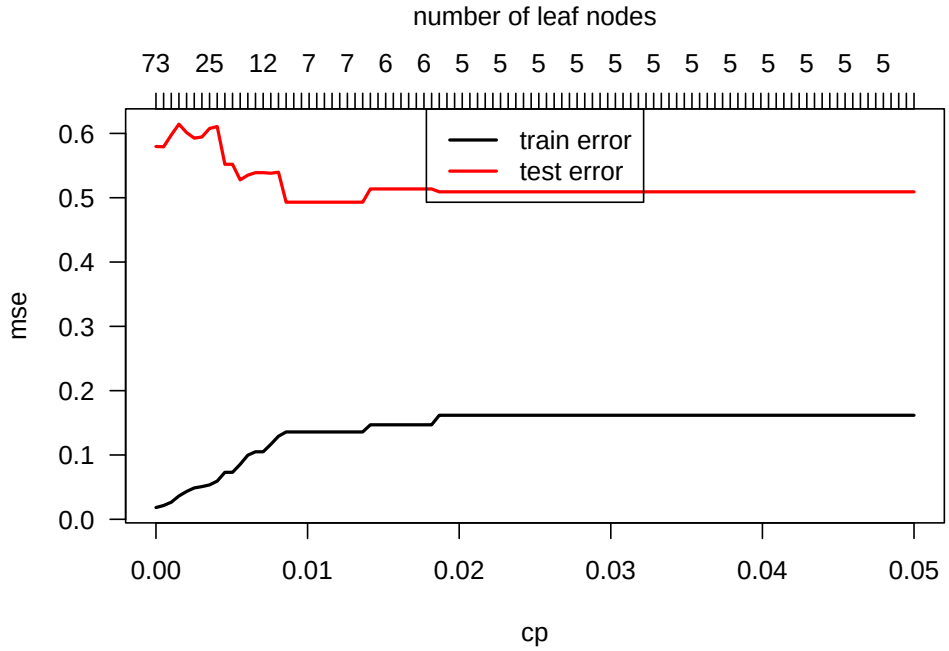
```
#-- More complex tree
# see ?rpart.control() for details
# xval: number of cross-validations
# minsplit: min obs to still allow a split
# cp: complexity parameter

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  0.611866      0   1.00000
#> 2  0.080767      1   0.38813
#> 3  0.056162      2   0.30737
#> 4  0.050368      3   0.25121
#> 5  0.018404      4   0.20084
#> 6  0.013809      5   0.18243
#> 7  0.008264      6   0.16862
#> 8  0.007883      7   0.16036
#> 9  0.007740      8   0.15248
#> 10 0.007267      9   0.14474
#> 11 0.007129     10   0.13747
#> 12 0.006491     11   0.13034
#> 13 0.005916     12   0.12385
#> 14 0.005816     14   0.11202
#> 15 0.005332     15   0.10620
#> 16 0.005078     16   0.10087
#> 17 0.005000     18   0.09071
#>
#> Variable importance
#> CAtBat  CHits  CRuns   CRBI CWalks CHmRun   Hits  AtBat   Runs    RBI  Walks
#>     17     17     16     15     13     10      2      2      2      2      2
#>  HmRun  Years
#>      1      1
#>
#> Node number 1: 200 observations
#>   mean=5.898, MSE=0.8053
length(unique(tree2$where))
#> [1] 19
```

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

mse(predict(tree2), bball$Y)             # training error
#> [1] 0.07305
mse(predict(tree2, X.test), Y.test)      # testing error
#> [1] 0.552
```



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

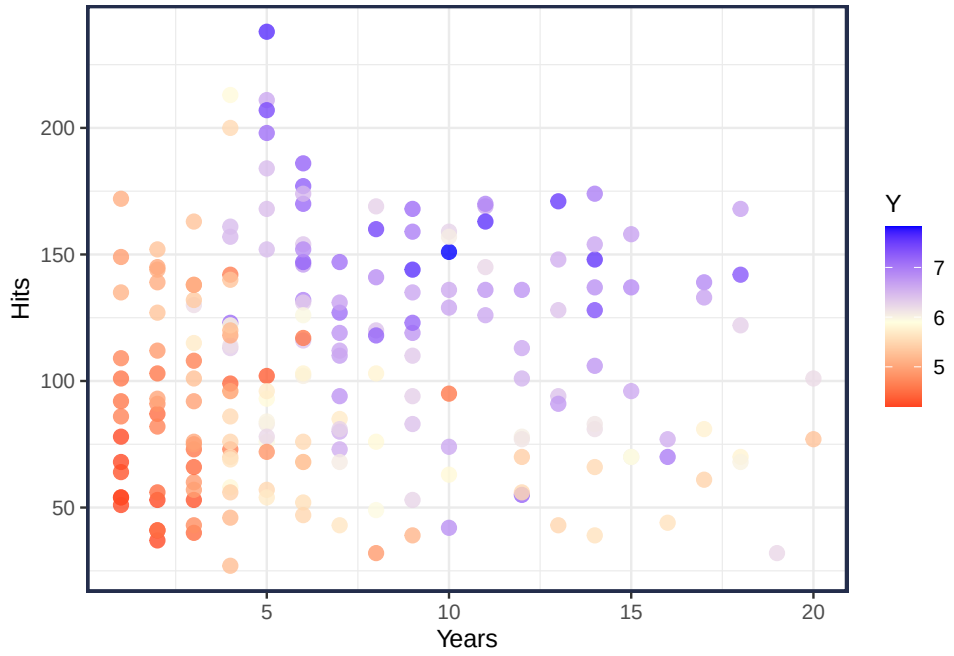
```
cp = seq(.05,0,length=100)  # cp is like a penalty on the tree size
for(i in 1:length(cp)){
  if(i == 1){train.error = test.error = nleafs = numeric(length(cp))}
  tree.fit = rpart(Y~.,data=bball, xval=0, minsplit=5, cp=cp[i])
  train.error[i] = mse(predict(tree.fit),bball$Y)            # training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

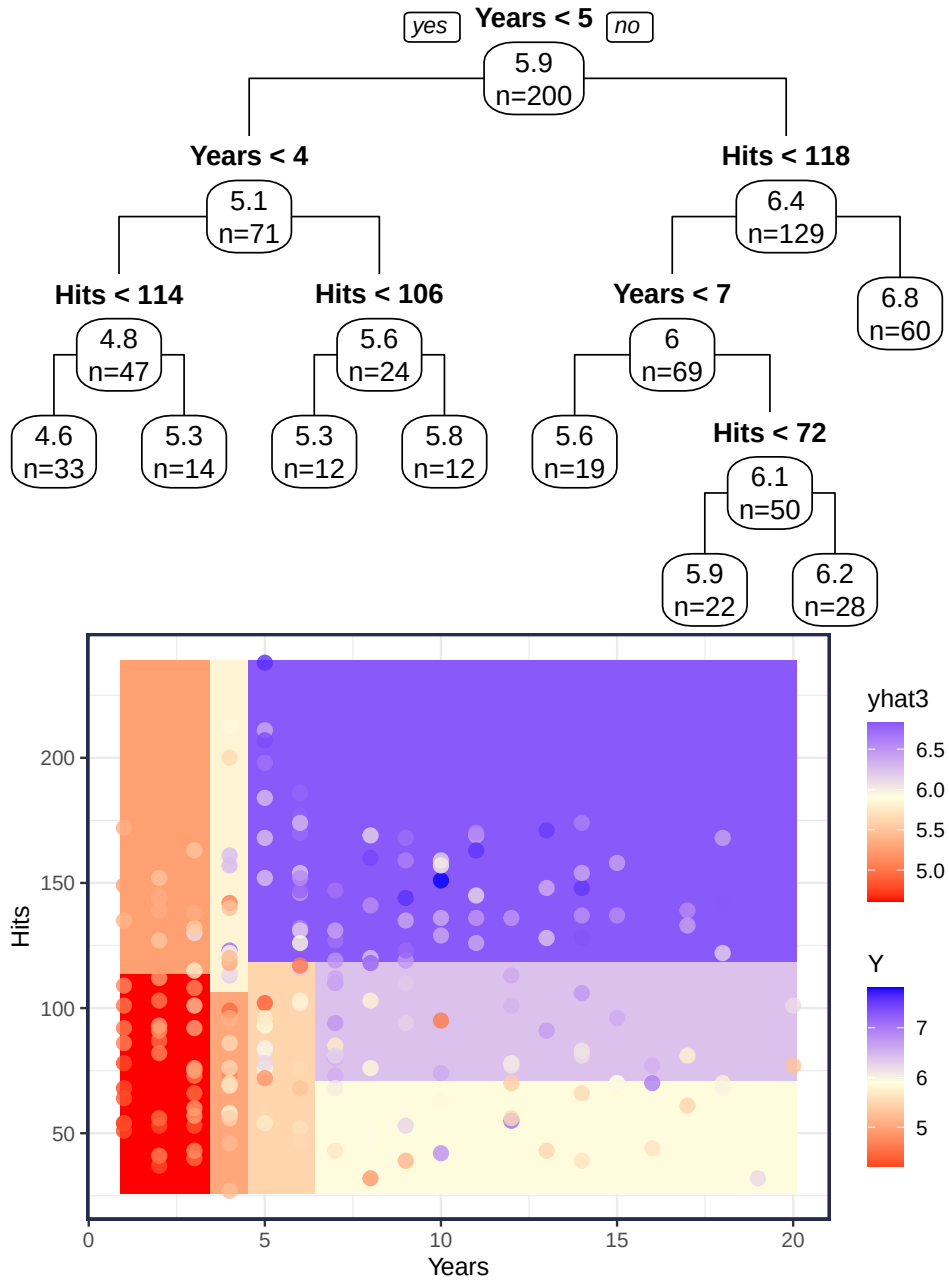
```
#####################################################################
#-- Regression Tree Examples for 2D
#####################################################################
library(ggplot2)

#-- 2D plot (using only Years and Hits)
p2D = ggplot(bball) + #scale_size_area(max_size=5) +
            scale_color_gradient2(midpoint=mean(bball$Y),mid="lightyellow",low="red",high="blue") +
  scale_fill_gradient2(midpoint=mean(bball$Y),mid="lightyellow",low="red",high="blue")

p2D +
  geom_point(aes(x=Years, y=Hits, color=Y), alpha=.8, size=3)
```



Let's fit a tree with the two predictors

```
#-- Fit tree to only Years and Hits
tree3 = rpart(Y~Years+Hits, data=bball)
summary(tree3,cp=1)
#> Call:
#> rpart(formula = Y ~ Years + Hits, data = bball)
#>   n= 200
#>
#>         CP nsplit rel error xerror    xstd
#> 1 0.47952      0    1.0000 1.0110 0.07293
#> 2 0.15162      1    0.5205 0.5722 0.06131
#> 3 0.05761      2    0.3689 0.4242 0.05369
#> 4 0.02587      3    0.3113 0.3831 0.04738
#> 5 0.01892      4    0.2854 0.3593 0.04643
#> 6 0.01019      5    0.2665 0.3427 0.04869
#> 7 0.01011      6    0.2563 0.3539 0.05074
#> 8 0.01000      7    0.2462 0.3539 0.05074
#>
#> Variable importance
#> Years  Hits
#>    73    27
#>
#> Node number 1: 200 observations
#>   mean=5.898, MSE=0.8053
length(unique(tree3$where))              # number of leaf nodes
#> [1] 8
prp(tree3, type=1, extra=1, branch=1)
mse(predict(tree3), bball$Y)             # training error
#> [1] 0.1982
mse(predict(tree3,X.test),Y.test)        # testing error
#> [1] 0.5179


#-- Plot Results
grid = expand.grid(Years = seq(min(bball$Years),max(bball$Years),length=90),
```

```
            Hits = seq(min(bball$Hits),max(bball$Hits),length=90))
grid$yhat3 = predict(tree3,newdata = grid)

p2D +
  geom_tile(data=grid,aes(x=Years, y=Hits, fill=yhat3)) +
  #geom_point(data=grid,aes(x=Years, y=Hits, color=yhat3),alpha=.9)
  geom_point(aes(x=Years, y=Hits, color=Y), alpha=.8, size=3)
```

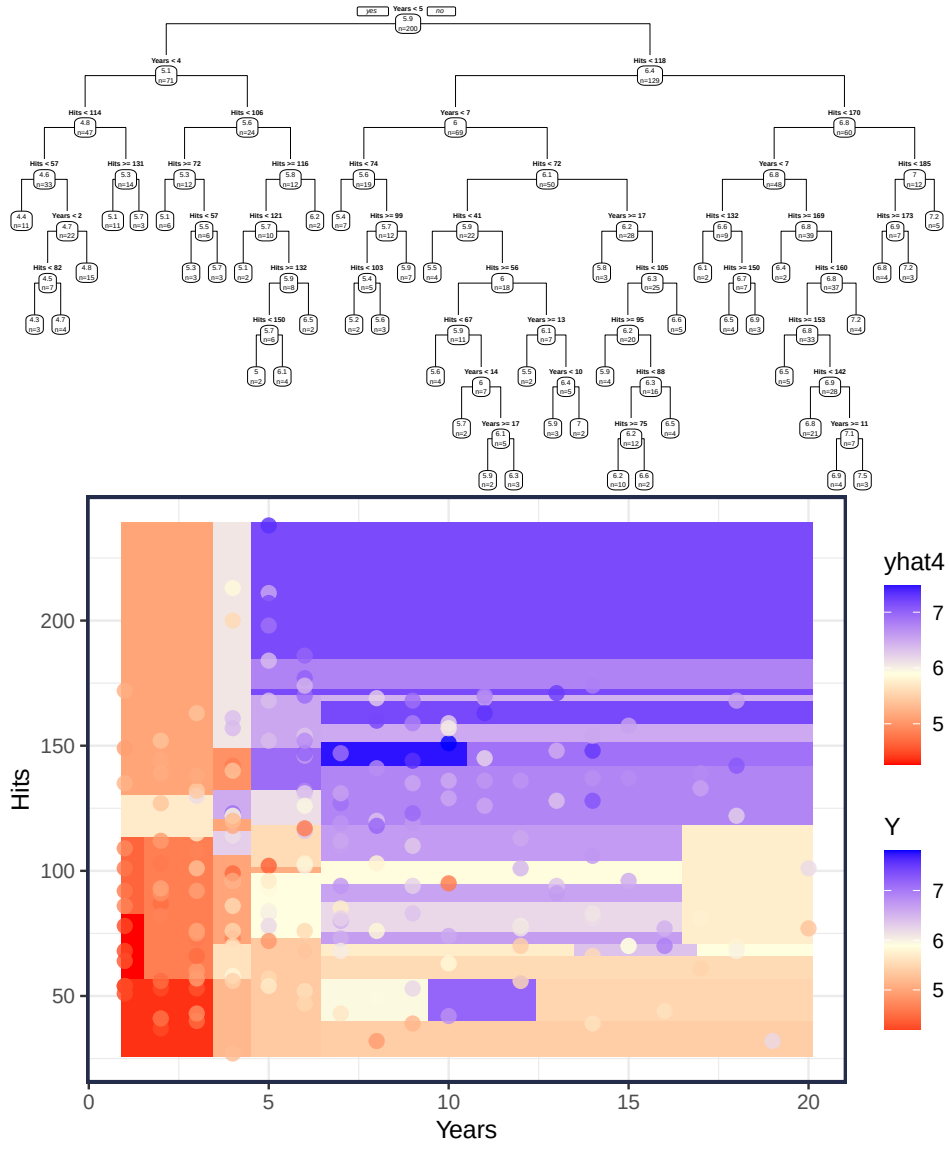


This shows the leaf regions (in 2D).

And we can also use more complex trees:

```
#-- Fit more complex tree to only Years and Hits
tree4 = rpart(Y~Years+Hits,data=bball,xval=0,minsplit=5,cp=0.001)
length(unique(tree4$where))              # number of leaf nodes
```

```
#> [1] 44
prp(tree4, type=1, extra=1, branch=1)
mse(predict(tree4), bball$Y)              # training error
#> [1] 0.09876
mse(predict(tree4,X.test), Y.test)        # testing error
#> [1] 0.6959


#-- Plot Results
grid$yhat4 = predict(tree4,newdata = grid)

# p2D + geom_point(data=grid,aes(x=Years,y=Hits,color=yhat4),alpha=.9) +
#    geom_point(aes(x=Years,y=Hits,color=Y),alpha=.8, size=3)

p2D +
  geom_tile(data=grid,aes(x=Years, y=Hits, fill=yhat4)) +
  #geom_point(data=grid,aes(x=Years, y=Hits, color=yhat3),alpha=.9)
  geom_point(aes(x=Years, y=Hits, color=Y), alpha=.8, size=3)
```



2.4 Details of Splitting (for Regression Trees)

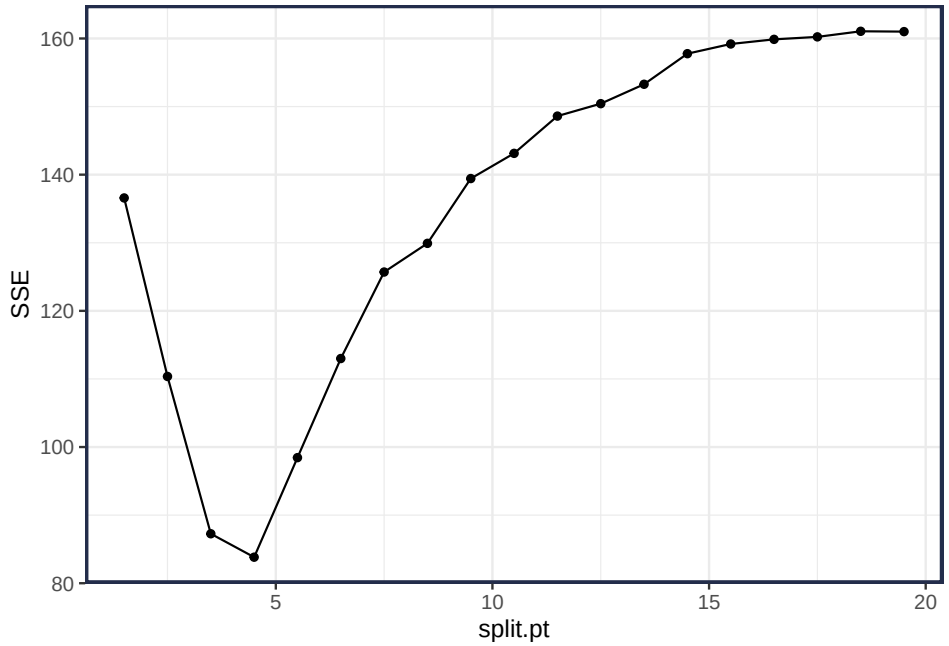
Consider only two dimensions, `hits` and `years` on which to make first split.

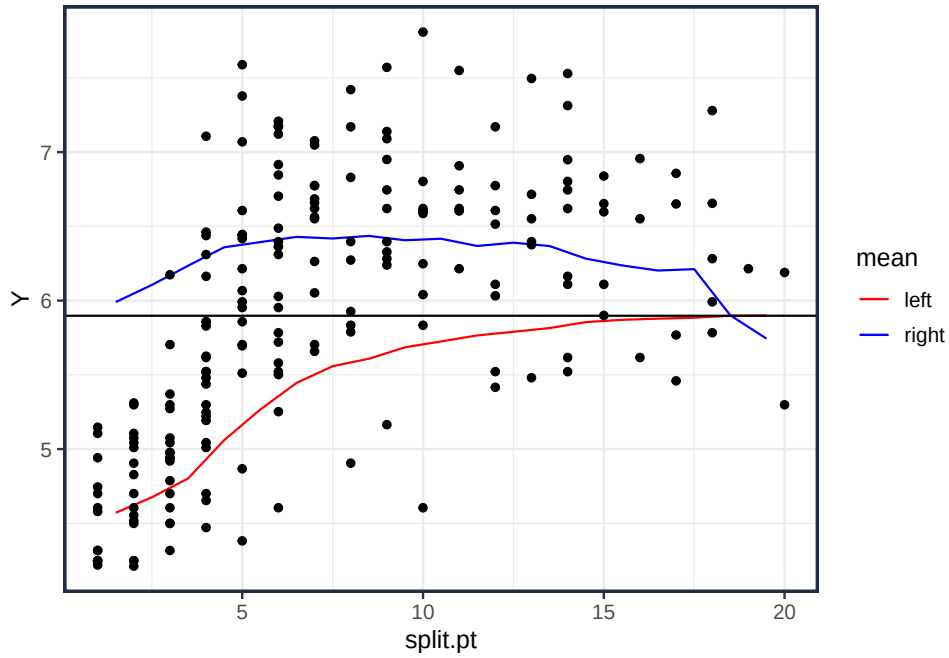
See `trees.R` for the `split_info()` and `split_metrics()` functions

2.4.1 First Split

```
## Split by Years
years = split_info(x=bball$Years, y=bball$Y)
head(years)
#> # A tibble: 6 x 9
#>   split.pt   n.L   n.R est.L est.R SSE.L SSE.R   SSE  gain
#>      <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1      1.5    13   187  4.57  5.99  1.37 135.  137.   24.5
#> 2      2.5    29   171  4.68  6.11  3.63 107.  110.   50.7
#> 3      3.5    47   153  4.80  6.23  8.29  79.0  87.3  73.8
#> 4      4.5    71   129  5.06  6.36 26.5   57.3  83.8  77.2
#> 5      5.5    88   112  5.27  6.39 52.7   45.7  98.4  62.6
#> 6      6.5   108    92  5.45  6.43 78.3   34.7 113.   48.1

ggplot(years,aes(x=split.pt,y=SSE)) + geom_line() + geom_point()

filter(years, min_rank(SSE) == 1)  # optimal split point for Years
#> # A tibble: 1 x 9
#>   split.pt   n.L   n.R est.L est.R SSE.L SSE.R   SSE  gain
#>      <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1      4.5    71   129  5.06  6.36  26.5  57.3  83.8  77.2

ggplot(years,aes(x=split.pt)) +
  geom_line(aes(y=est.L,color="left")) +             # mean left of split pt
  geom_line(aes(y=est.R,color="right")) +            # mean right of split pt
  geom_hline(yintercept=mean(bball$Y))+              # overall mean
  scale_color_manual("mean",values=c('left'='red','right'='blue')) +
  geom_point(data=bball,aes(x=Years,y=Y))  +         # add points
  labs(y="Y")
```

2.4.1.1 Split on Years

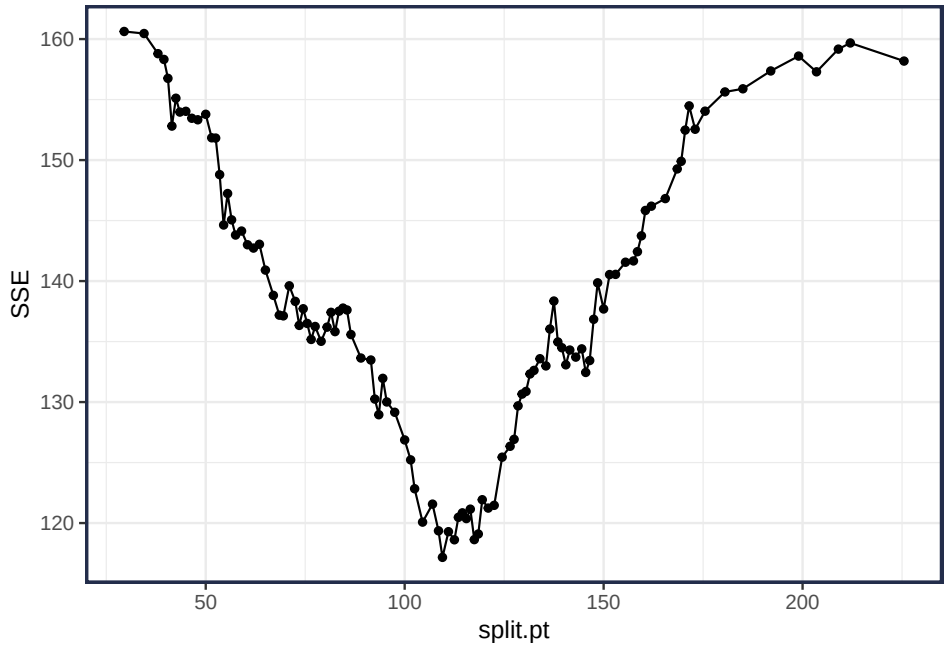


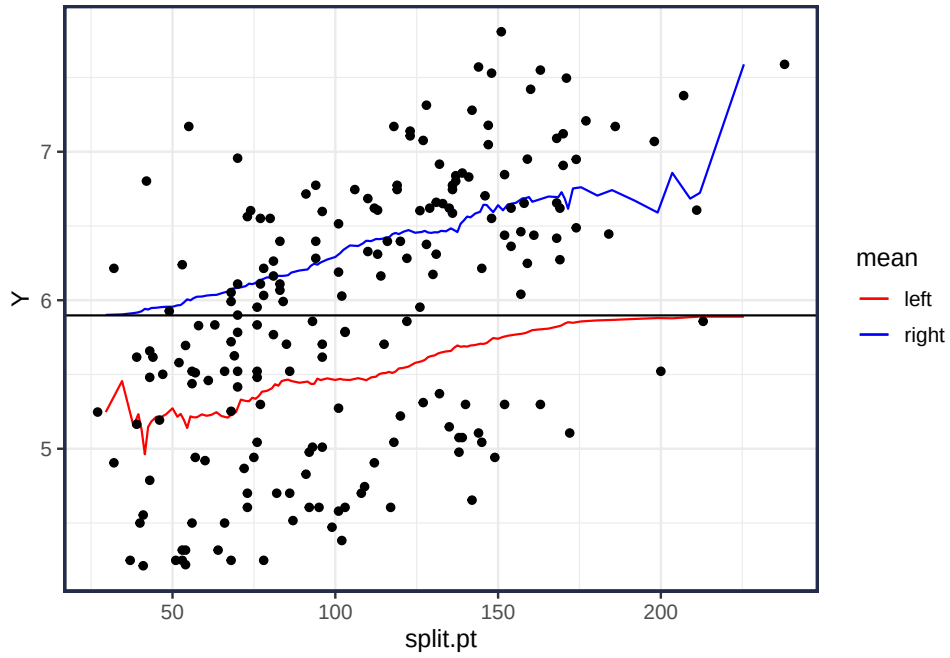



```
## Split by Hits
hits = split_info(x=bball$Hits,y=bball$Y)
head(hits)
#> # A tibble: 6 x 9
#>   split.pt   n.L   n.R est.L est.R SSE.L SSE.R   SSE  gain
#>      <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1     29.5     1   199  5.25  5.90 0      161.  161. 0.426
#> 2     34.5     3   197  5.46  5.90 0.922  160.  160. 0.596
#> 3     38       4   196  5.15  5.91 2.02   157.  159. 2.26
#> 4     39.5     6   194  5.23  5.92 2.19   156.  158. 2.74
#> 5     40.5     7   193  5.13  5.93 2.65   154.  157. 4.30
#> 6     41.5     9   191  4.96  5.94 3.57   149.  153. 8.25
```

```
ggplot(hits,aes(x=split.pt,y=SSE)) + geom_line() + geom_point()

filter(hits, min_rank(SSE)==1)     # optimal split point for Hits
#> # A tibble: 1 x 9
#>   split.pt   n.L   n.R est.L est.R SSE.L SSE.R   SSE  gain
#>      <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1     110.   107    93  5.46  6.40  61.3  55.9  117.  43.9

ggplot(hits,aes(x=split.pt)) +
  geom_line(aes(y=est.L,color="left")) +              # mean left of split pt
  geom_line(aes(y=est.R,color="right")) +             # mean right of split pt
  geom_hline(yintercept=mean(bball$Y))+          # overall mean
  scale_color_manual("mean",values=c('left'='red','right'='blue')) +
  geom_point(data=bball,aes(x=Hits,y=Y)) +              # add points
  labs(y = "Y")
```

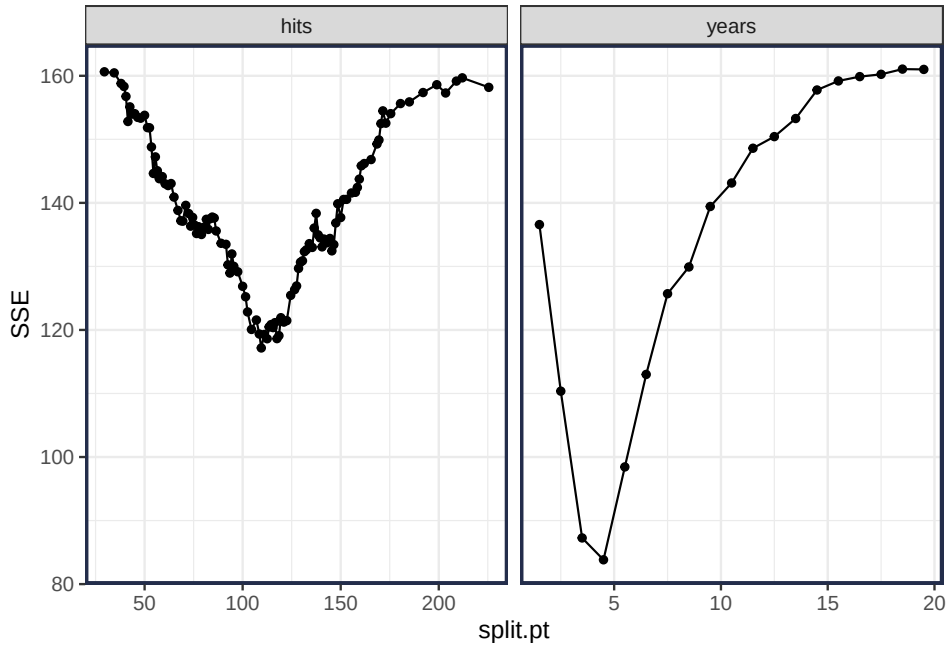
2.4.1.2 Split on Hits






```
## No splits
sum((bball$Y-mean(bball$Y))^2)    # SSE if no splits are made
#> [1] 161.1
# (nrow(bball)-1)*var(bball$Y)



## Results (see function split_metrics at top of file)
#  splitting on Years gives the best reduction in SSE, so we would split on
#  Years (at a value of 4.5).
sum((bball$Y-mean(bball$Y))^2)        # no split
#> [1] 161.1
filter(years, min_rank(SSE)==1)       # split on years
#> # A tibble: 1 x 9
#>   split.pt   n.L   n.R est.L est.R SSE.L SSE.R   SSE  gain
#>      <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1      4.5    71   129  5.06  6.36  26.5  57.3  83.8  77.2
filter(hits, min_rank(SSE)==1)        # split on hits
#> # A tibble: 1 x 9
#>   split.pt   n.L   n.R est.L est.R SSE.L SSE.R   SSE  gain
#>      <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1     110.   107    93  5.46  6.40  61.3  55.9  117.  43.9
split_metrics(bball$Years,bball$Y, 4.5)
#> # A tibble: 2 x 3
#>   region   SSE     n
#>   <chr>  <dbl> <int>
#> 1 LEFT    26.5    71
#> 2 RIGHT   57.3   129

## Comparison of splitting on both variables
bind_rows(hits=hits, years=years, .id="split.var") %>%
  ggplot(aes(x=split.pt, y=SSE)) + geom_line() + geom_point() +
  facet_wrap(~split.var, scales="free_x")
```

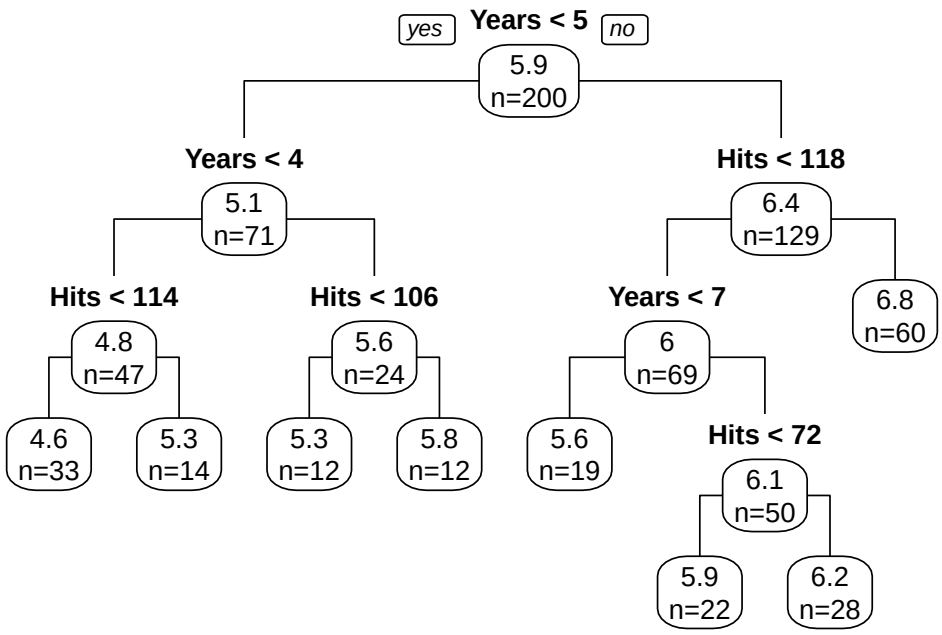
2.4.1.3 Find best variable to split on



2.4.2 Second Split

```
#-- 2nd Split
# now we have to compare 4 possibilities. We can split on Years or Hits, but
#  use data that has Years < 4.5 or Years > 4.5

left = (bball$Years<=4.5)                                  # split point from previous step
years2.L = split_info(x=bball$Years[left],y=bball$Y[left])
years2.R = split_info(x=bball$Years[!left],y=bball$Y[!left])
hits2.L = split_info(x=bball$Hits[left],y=bball$Y[left])
hits2.R = split_info(x=bball$Hits[!left],y=bball$Y[!left])

#-- Find best region to split on
max(years2.L$gain,na.rm=TRUE)
#> [1] 9.278
max(years2.R$gain,na.rm=TRUE)
#> [1] 1.576
max(hits2.L$gain,na.rm=TRUE)
#> [1] 8.726
max(hits2.R$gain,na.rm=TRUE)
#> [1] 24.42

hits2.R[which.max(hits2.R$gain),]
#> # A tibble: 1 x 9
#>   split.pt   n.L   n.R est.L est.R SSE.L SSE.R   SSE  gain
#>      <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1     118.    69    60  5.95  6.83  22.6  10.3  32.9  24.4

# 2nd split on Hits <= 117.5 in region 2.



#-- Summary of Splits
```

```
# Rule 1: Years < 4.5
# Rule 2: Years >= 4.5 & Hits < 117.5
# ...
prp(tree3, type=1, extra=1, branch=1)
```



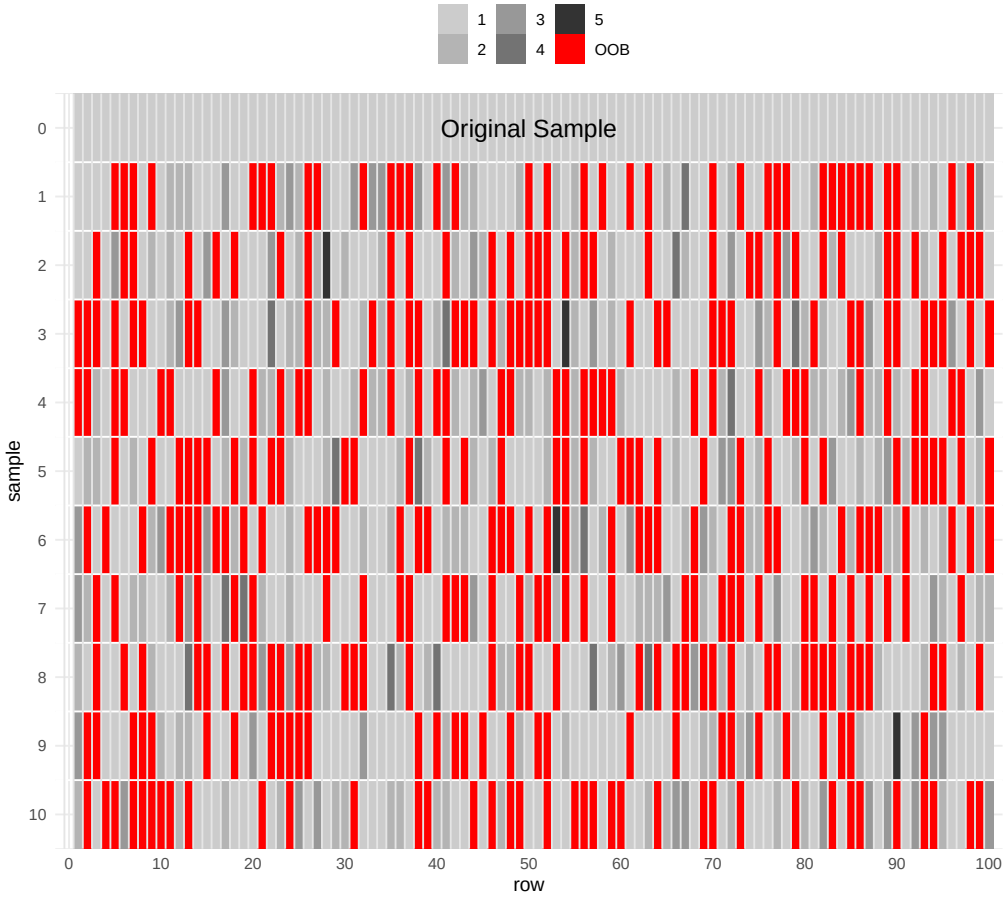
3 Bagging Trees

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 $\mathcal{D}_b$.
- 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)



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

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

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 $\{\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_M\}$
- 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:

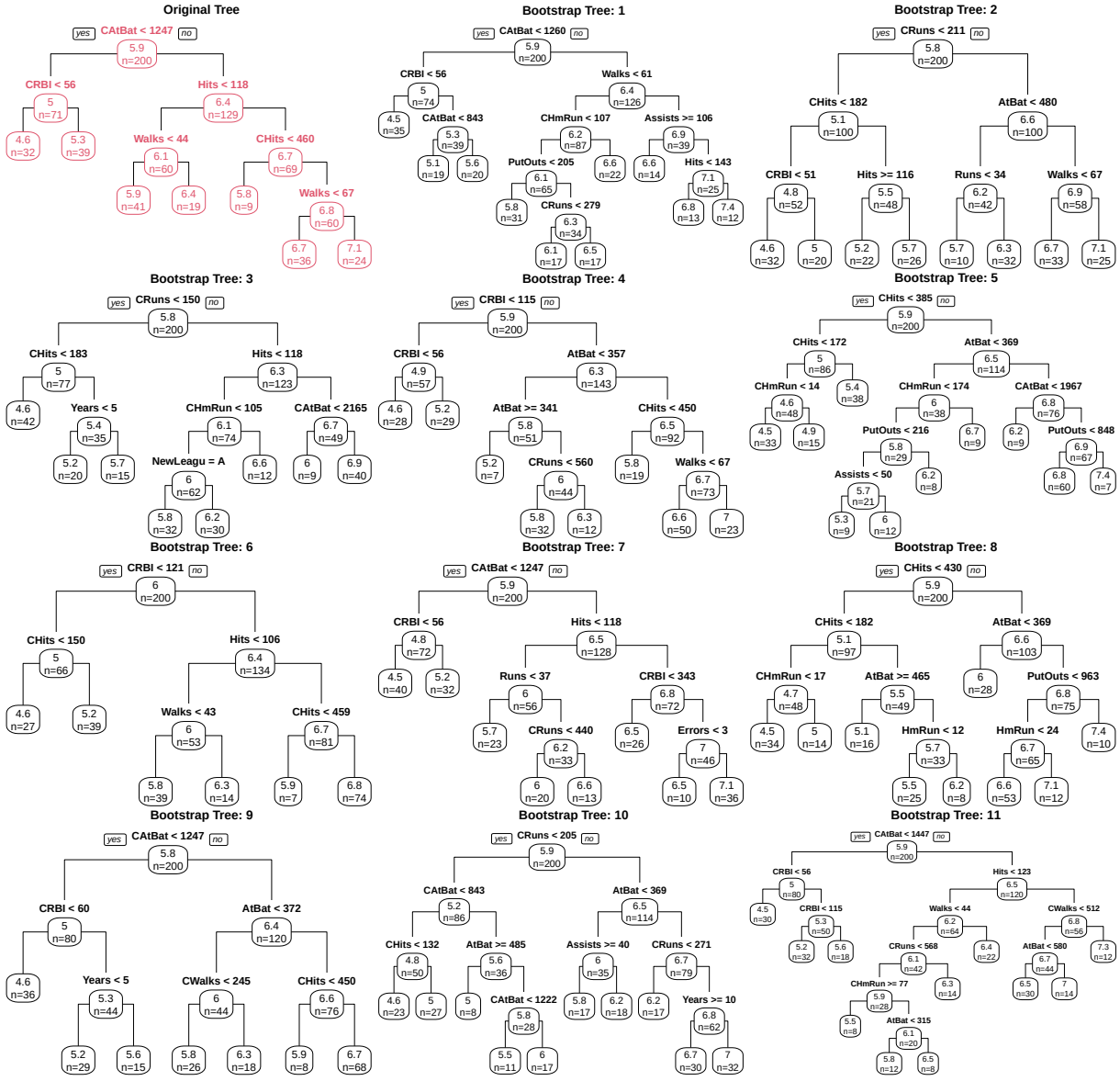
$$\mathrm{E}(\bar{\theta}) = \frac{1}{M} \sum_{i=1}^{M} \mathrm{E}(\hat{\theta}_i)$$

- 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.

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

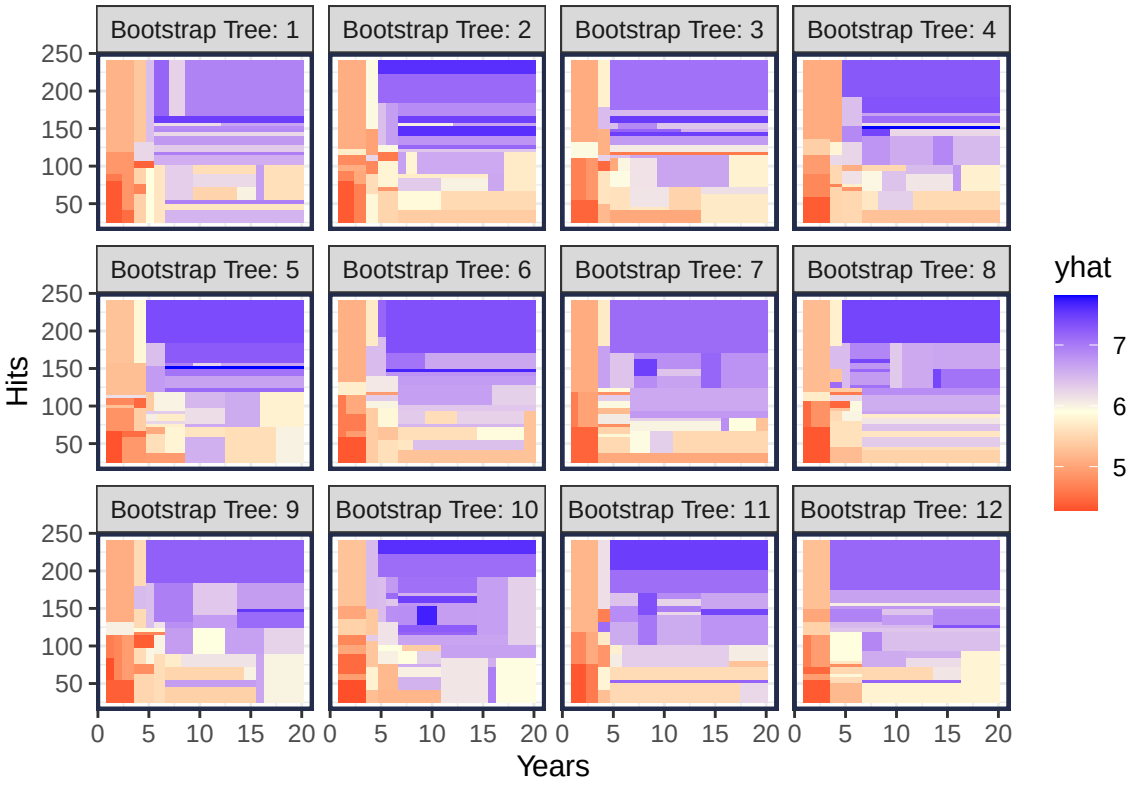


Figure 1: Bagging in 2 dimensions (Years and Hits)

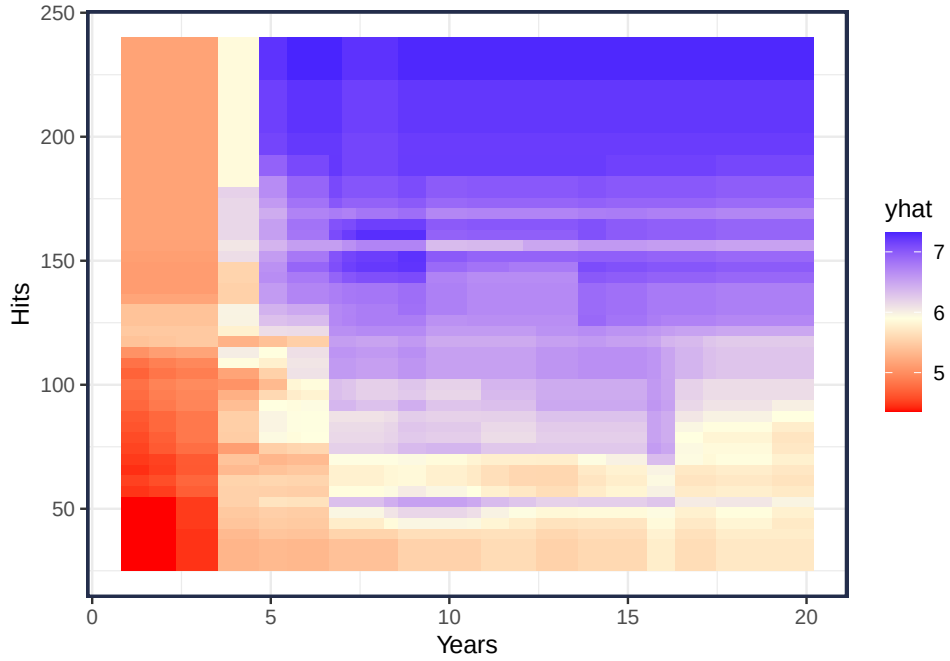


Figure 2: Average of bootstrap predictions

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:

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 bth 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.

Illustration

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.

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

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

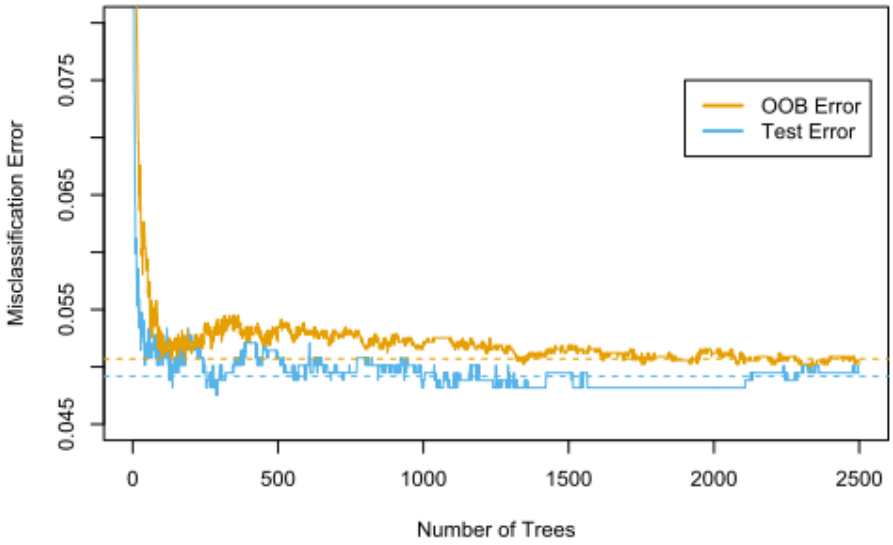


Figure 15.4 in ESL

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.

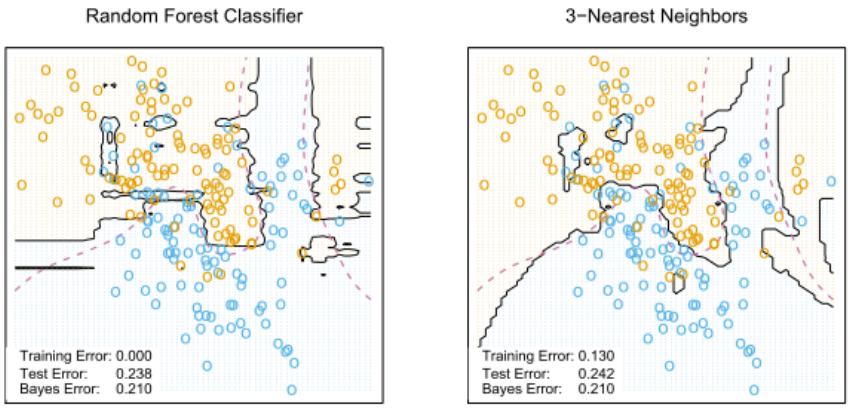


FIGURE 15.11. *Random forests versus 3-NN on the mixture data. The axis-oriented nature of the individual trees in a random forest lead to decision regions with an axis-oriented flavor.*