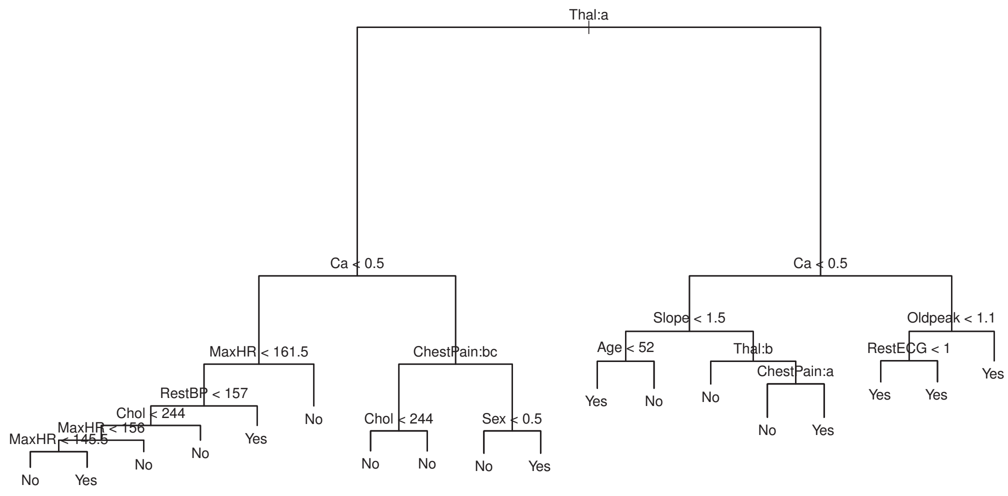
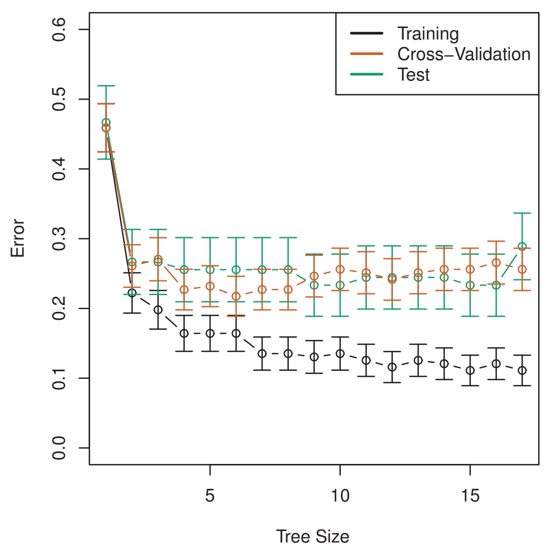
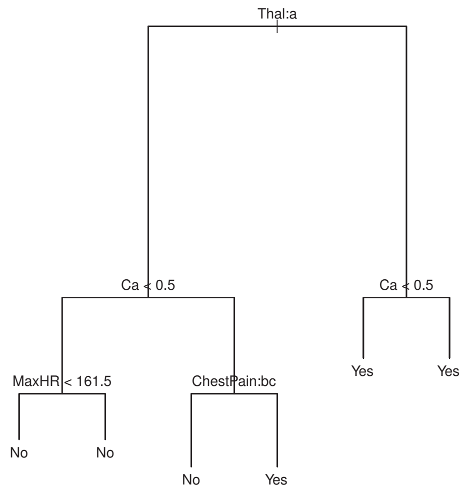






Heart data. These data contain a binary outcome HD for 303 patients who presented with chest pain. An outcome value of Yes indicates the presence of heart disease based on an angiographic test, while No means no heart disease. There are 13 predictors including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements. Top: The unpruned tree. Bottom Left: Cross-validation error, training, and test error, for different sizes of the pruned tree. Bottom Right: The pruned tree corresponding to the minimal cross-validation error.

Trees Versus Linear Models

Linear regression assumes a model of the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j$$

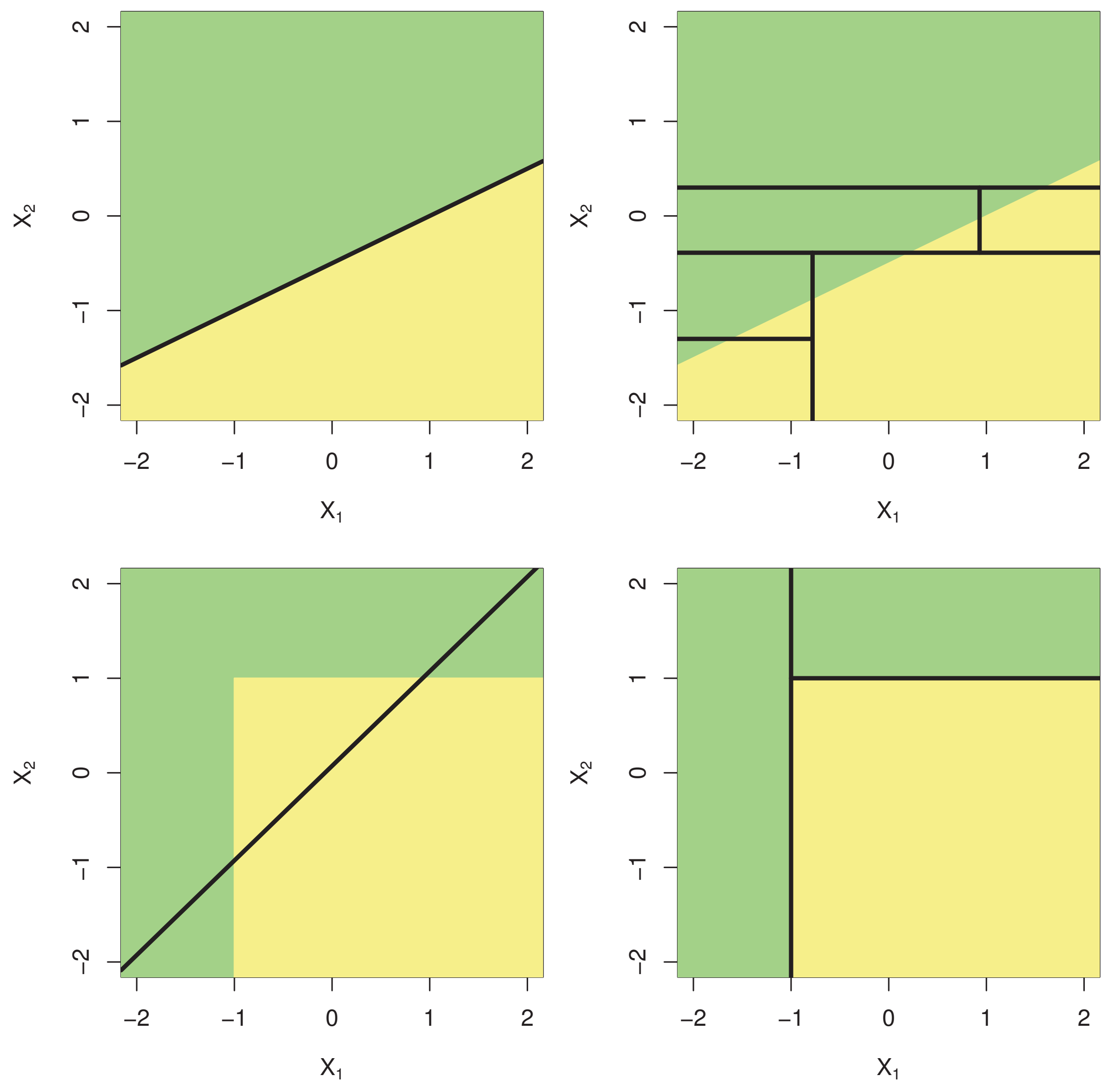
whereas regression trees assume a model of the form

$$f(X) = \beta_0 + \sum_{m=1}^{M} c_m \cdot 1_{(X \in R_m)}$$

where $R_1, \ldots, R_M$ represent a partition of feature space.

Which model is better?

It depends on the problem at hand.



Top Row: A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right). Bottom Row: Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).

Advantages and Disadvantages of Trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen so far.
- Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

Performing Decision Trees in R

The tree library is used to construct classification and regression trees.

We first use classification trees to analyze the Hitters data set. In these data, Salary is a continuous variable, and so we begin by recoding it as a binary variable. We use the ifelse() function to create a variable, called High, which takes on a value of Yes if the log Salary variable exceeds 6.5, and takes on a value of No otherwise.

```
> library(ISLR2)
> library(tree)
> attach(Hitters)
> dim(Hitters)
[1] 263  20
```

```
> names(Hitters)
 [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"
 [6] "Walks"     "Years"     "CAtBat"    "CHits"     "CHmRun"
[11] "CRuns"     "CRBI"      "CWalks"    "League"    "Division"
[16] "PutOuts"   "Assists"   "Errors"    "Salary"    "NewLeague"
> summary(Hitters)
     AtBat            Hits           HmRun            Runs
 Min.   : 19.0   Min.   :  1.0   Min.   : 0.00   Min.   :  0.00
 1st Qu.:282.5   1st Qu.: 71.5   1st Qu.: 5.00   1st Qu.: 33.50
 Median :413.0   Median :103.0   Median : 9.00   Median : 52.00
 Mean   :403.6   Mean   :107.8   Mean   :11.62   Mean   : 54.75
 3rd Qu.:526.0   3rd Qu.:141.5   3rd Qu.:18.00   3rd Qu.: 73.00
 Max.   :687.0   Max.   :238.0   Max.   :40.00   Max.   :130.00
      RBI             Walks             Years            CAtBat
 Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0
 1st Qu.: 30.00   1st Qu.: 23.00   1st Qu.: 4.000   1st Qu.:  842.5
 Median : 47.00   Median : 37.00   Median : 6.000   Median : 1931.0
 Mean   : 51.49   Mean   : 41.11   Mean   : 7.312   Mean   : 2657.5
 3rd Qu.: 71.00   3rd Qu.: 57.00   3rd Qu.:10.000   3rd Qu.: 3890.5
 Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0
     CHits            CHmRun           CRuns             CRBI
    Min.   :   4.0   Min.   :  0.00   Min.   :   2.0   Min.   :   3.0
 1st Qu.: 212.0   1st Qu.: 15.00   1st Qu.: 105.5   1st Qu.:  95.0
 Median : 516.0   Median : 40.00   Median : 250.0   Median : 230.0
 Mean   : 722.2   Mean   : 69.24   Mean   : 361.2   Mean   : 330.4
 3rd Qu.:1054.0   3rd Qu.: 92.50   3rd Qu.: 497.5   3rd Qu.: 424.5
 Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.0
     CWalks         League  Division    PutOuts          Assists
 Min.   :   1.0   A:139   E:129    Min.   :   0.0   Min.   :  0.0
 1st Qu.:  71.0   N:124   W:134    1st Qu.: 113.5   1st Qu.:  8.0
 Median : 174.0                    Median : 224.0   Median : 45.0
 Mean   : 260.3                    Mean   : 290.7   Mean   :118.8
 3rd Qu.: 328.5                    3rd Qu.: 322.5   3rd Qu.:192.0
 Max.   :1566.0                    Max.   :1377.0   Max.   :492.0
     Errors           Salary        NewLeague
 Min.   : 0.000   Min.   :  67.5   A:141
 1st Qu.: 3.000   1st Qu.: 190.0   N:122
 Median : 7.000   Median : 425.0
 Mean   : 8.593   Mean   : 535.9
 3rd Qu.:13.000   3rd Qu.: 750.0
 Max.   :32.000   Max.   :2460.0
```

```
> Log_Salary <- log(Salary)
> summary(Log_Salary)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.212   5.247   6.052   5.927   6.620   7.808

> High <- factor(ifelse(Log_Salary <= 6.5, "No", "Yes"))
```

Finally, we use the data.frame() function to merge High with the rest of the Hitters data.

```
> Hitters_new <- data.frame(Hitters, High)
> dim(Hitters_new)
[1] 263  21
> names(Hitters_new)
 [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"
 [6] "Walks"     "Years"     "CAtBat"    "CHits"     "CHmRun"
[11] "CRuns"     "CRBI"      "CWalks"    "League"    "Division"
[16] "PutOuts"   "Assists"   "Errors"    "Salary"    "NewLeague"
[21] "High"
```

We now use the tree() function to fit a classification tree in order to predict High using all variables but Salary. The syntax of the tree() function is quite similar to that of the lm() function.

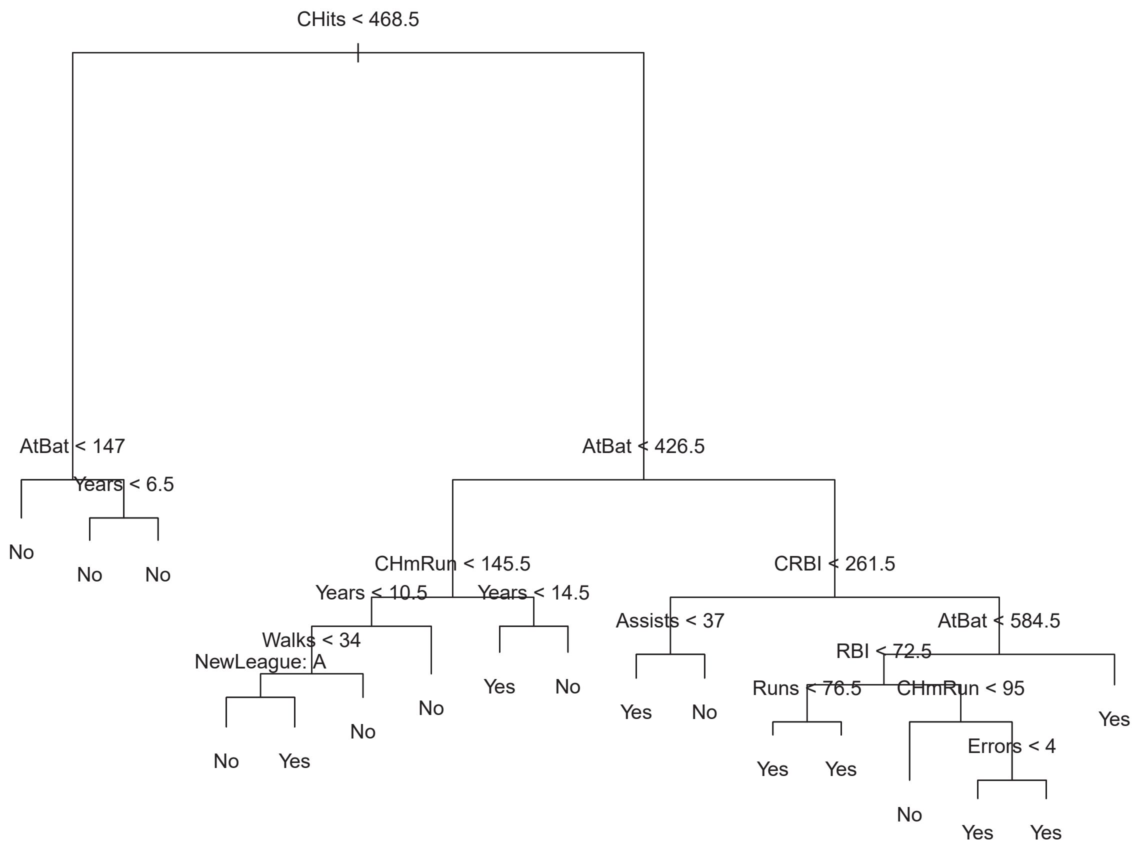
```
> tree_hitters <- tree(High ~ . - Salary, Hitters_new)
> summary(tree_hitters)

Classification tree:
tree(formula = High ~ . - Salary, data = Hitters_new)
Variables actually used in tree construction:
 [1] "CHits"     "AtBat"     "Years"     "CHmRun"    "Walks"
 [6] "NewLeague" "CRBI"      "Assists"   "RBI"       "Runs"
[11] "Errors"
Number of terminal nodes:  17
Residual mean deviance:  0.2897 = 71.27 / 246
Misclassification error rate: 0.06464 = 17 / 263
```

↗ training error rate

We use the plot() function to display the tree structure, and the text() function to display the node labels. The argument pretty = 0 instructs R to include the category names for any qualitative predictors, rather than simply displaying a letter for each category.

```
> plot(tree_hitters)
> text(tree_hitters, pretty = 0)
```



```
> tree_hitters
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

  1) root 263 331.000 No ( 0.676806 0.323194 )
    2) CHits < 468.5 122  28.160 No ( 0.975410 0.024590 )
      4) AtBat < 147 5   6.730 No ( 0.600000 0.400000 ) *
      5) AtBat > 147 117  11.520 No ( 0.991453 0.008547 )
       10) Years < 6.5 110   0.000 No ( 1.000000 0.000000 ) *
       11) Years > 6.5 7   5.742 No ( 0.857143 0.142857 ) *
    3) CHits > 468.5 141 191.700 Yes ( 0.418440 0.581560 )
      6) AtBat < 426.5 58  71.850 No ( 0.689655 0.310345 )
       12) CHmRun < 145.5 43  44.120 No ( 0.790698 0.209302 )
         24) Years < 10.5 24  31.760 No ( 0.625000 0.375000 )
           48) Walks < 34 14  19.120 Yes ( 0.428571 0.571429 )
             96) NewLeague: A 6   5.407 No ( 0.833333 0.166667 ) *
             97) NewLeague: N 8   6.028 Yes ( 0.125000 0.875000 ) *
           49) Walks > 34 10   6.502 No ( 0.900000 0.100000 ) *
         25) Years > 10.5 19   0.000 No ( 1.000000 0.000000 ) *
       13) CHmRun > 145.5 15  20.190 Yes ( 0.400000 0.600000 )
         26) Years < 14.5 5   0.000 Yes ( 0.000000 1.000000 ) *
         27) Years > 14.5 10  13.460 No ( 0.600000 0.400000 ) *
      7) AtBat > 426.5 83  89.300 Yes ( 0.228916 0.771084 )
       14) CRBI < 261.5 11  12.890 No ( 0.727273 0.272727 )
         28) Assists < 37 5   6.730 Yes ( 0.400000 0.600000 ) *
         29) Assists > 37 6   0.000 No ( 1.000000 0.000000 ) *
       15) CRBI > 261.5 72  61.560 Yes ( 0.152778 0.847222 )
         30) AtBat < 584.5 52  53.660 Yes ( 0.211538 0.788462 )
           60) RBI < 72.5 25   8.397 Yes ( 0.040000 0.960000 )
            120) Runs < 76.5 20   0.000 Yes ( 0.000000 1.000000 ) *
            121) Runs > 76.5 5   5.004 Yes ( 0.200000 0.800000 ) *
           61) RBI > 72.5 27  35.590 Yes ( 0.370370 0.629630 )
            122) CHmRun < 95 6   0.000 No ( 1.000000 0.000000 ) *
            123) CHmRun > 95 21  20.450 Yes ( 0.190476 0.809524 )
              246) Errors < 4 6   8.318 Yes ( 0.500000 0.500000 ) *
              247) Errors > 4 15   7.348 Yes ( 0.066667 0.933333 ) *
         31) AtBat > 584.5 20   0.000 Yes ( 0.000000 1.000000 ) *
```

In order to properly evaluate the performance of a classification tree on these data, we must estimate the test error rather than simply computing the training error. We split the observations into a training set and a test set, build the tree using the training set, and evaluate its performance on the test data. The predict() function can be used for this purpose. In the case of a classification tree, the argument type = ”class” instructs R to return the actual class prediction.

```
> set.seed(1)
> train <- sample(1:nrow(Hitters_new), 132)
> Hitters_test <- Hitters_new[-train, ]
> High_test <- High[-train]
> tree_hitters <- tree(High ~ . - Salary, Hitters_new, subset = train)
> tree_pred <- predict(tree_hitters, Hitters_test, type = "class")
> table(tree_pred, High_test)
         High_test
tree_pred No Yes
      No  78  12
      Yes 10  31
> mean(tree_pred!=High_test)
[1] 0.1679389
```

Next, we consider whether pruning the tree might lead to improved results.

The function cv.tree() performs cross-validation in order to determine the optimal level of tree complexity; cost complexity pruning is used in order to select a sequence of trees for consideration. The cv.tree() function reports the number of terminal nodes of each tree considered (size) as well as the corresponding error rate and the value of the cost-complexity parameter used.