Tree-Based Methods – part 1

Regression Trees

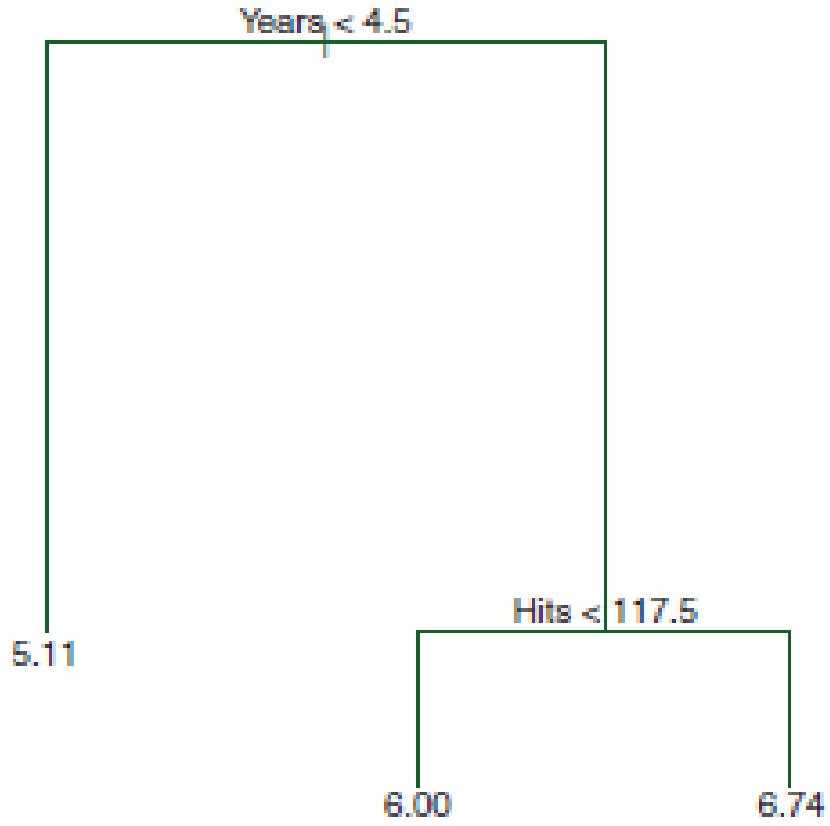
- These models take place somewhere between linear models and the nonparametric approaches (e.g. KNN)
- They are best described by the algorithm used to create them and we will take this approach too
- The result is a decision tree

For the `Hitters` data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year

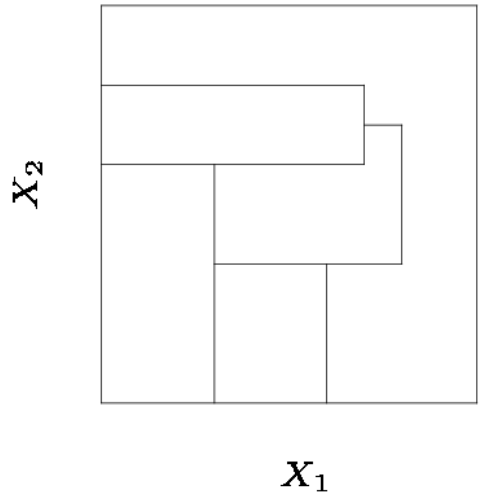
The number in each leaf is the mean of the response for the observations that fall there

A prediction for a player with 3 years of experience and 105 hits:

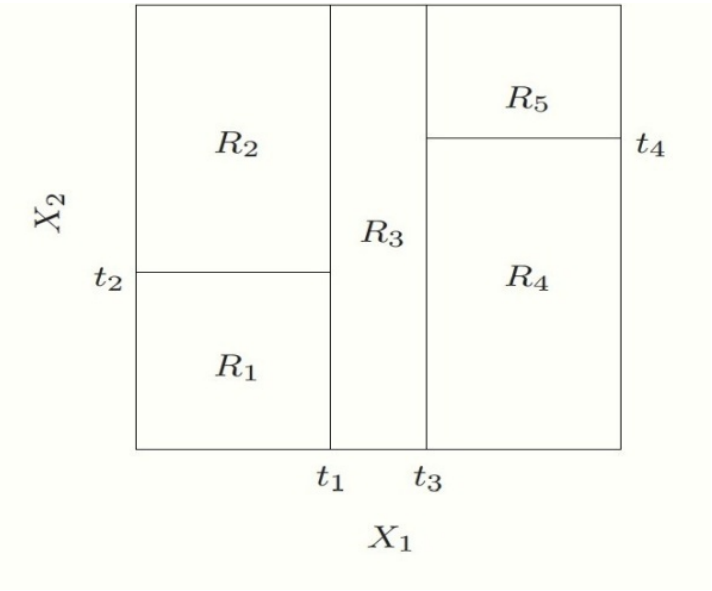
$$\$1{,}000 * e^{6.00} = \$403{,}429$$

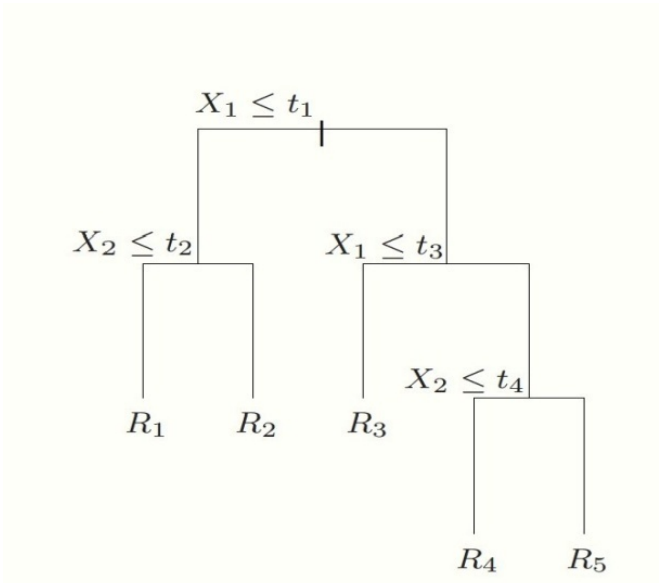


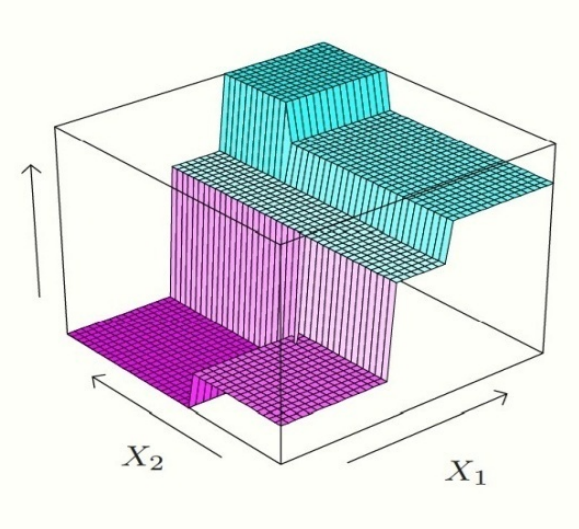
- Let's assume we have 2 variables X_1 and X_2
- One simple idea to model the outcome Y is to split the domain of the values of X_1 and X_2 in several areas and take the average of the Y's in each area
- One way to divide the domain of their values in 5 regions is like this



- In Regression Trees we use a special, recursive partitioning







Top left: Recursive Binary splitting

Top right: A tree corresponding to this partitioning

Bottom left: Prediction surface corresponding to this tree

Recursive partitioning regression algorithm

1. All partitions of the region of the predictors into 2 regions with the division parallel to one of the axes.

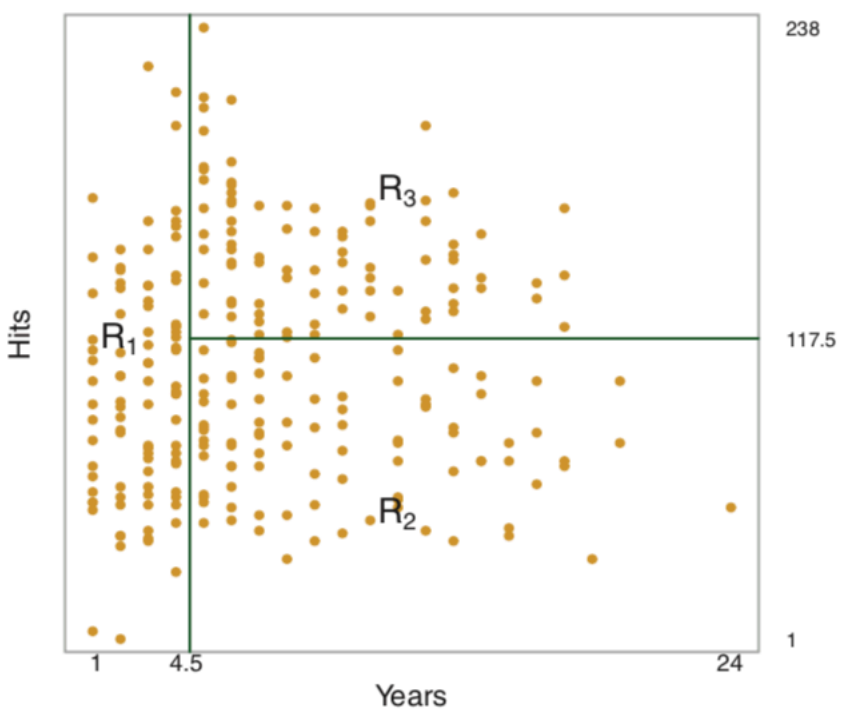
2. Take the mean of the response in each partition and compute

   $$\text{RSS(partition)} = \text{RSS(part}_1\text{)} + \text{RSS(part}_2\text{)}$$

   Take the partition minimizing the RSS.

3. (Sub)partition the partitions recursively, i.e. “within existing partitions”. Repeated splits on the same variable are allowed.

The tree and the regions for the Hitters data again:



Few remarks:

- For categorical predictors split on the levels of the factor (L levels)
  - For ordered factor: at most $L-1$ splits
  - For unordered factor: up to $2^{L-1}-1$ splits

    (We can split into a set of 1 level vs. $L-1$ levels in $L=\binom{L}{1}$ ways, in a set of 2 levels vs. $L-2$ levels in a $\binom{L}{2}$ ways etc. $L-1$ level vs. 1 level in $L=\binom{L}{L-1}$ ways. The total is
    $\left[\binom{L}{1}+\binom{L}{2}+\cdots+\binom{L}{L-1}\right]/2=[2^L-\binom{L}{0}-\binom{L}{L}]/2 = 2^{L-1}-1$)

- There is no point in monotone predictor transformations. But transforming the response will affect the tree selection (through the RSS)
- Missing values are "gracefully" handled either by indicators or surrogates
- Tree methods find easily (even too easily) interaction
- The resulting data model is easy to understand

Tree pruning

- When to stop "growing" the tree?
- The greedy strategy is to keep growing the tree until the reduction in overall cost (RSS for the regression tree) is not reduced by more than some small number ε. Of course it's not easy to determine a good ε.
- Another strategy is to grow a large tree T_0 (with only restriction to have at least 5 observations in the "leaves") and then prune it back. Pruning means that given a tree of size *n*, the best tree of size *n-1* (the one increasing the criterion by the least amount) is determine by combining adjacent nodes in all possible ways.
- This is similar to backward elimination in variable selection for linear regression. It is achieved by using a cost-complexity function defined on next slide as the criterion in the pruning process. This function is used to reduce the number of the considered trees.

- For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T| \qquad (8.4)$$

(the cost-complexity function) is as small as possible. Here

$|T|$ = # of terminal nodes (leaves) of T

R_m = subset corresponding to the m-th terminal node

$\hat{y}_{R_m}$ = predicted response, i.e. the mean of all training obs "in" R_m

α = tuning parameter controlling the trade-off between the subtree complexity and its fit (on the train data).

- When $\alpha = 0$, then the subtree T will simply equal the large tree T_0 (then (8.4) just measures the training error).

However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity (8.4) will tend to be minimized for a smaller subtree. The detailed description of the algorithm follows.

Algorithm 8.1(ISLR): Building a Regression Tree

1. Use recursive binary splitting to grow a large tree T_0 on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step2 that corresponds to the chosen value of α.

Classification Trees

Trees can be fit to different response data types

- Regression tree (continuous) – the mean (e.g. the null model) is computed in each partition
- Extend the approach to Poisson, survival etc. data by
  - Fitting the proper null model on each partition
  - Using the deviance instead of the RSS as a criterion
- Classification trees, e.g. for binomial data, need to use a criterion different than the RSS to split the nodes. This new criterion is not necessarily the deviance

- Data in nodes is divided so that observations in a split are as "pure" as possible. For binomial data (2 class type), it means one class type dominates in the split

- Let the target y be a classification outcome taking values $1,2,\ldots,K$
- In a node i, representing region R_i with number of observations N_i, let

$$p_{ik} = \frac{1}{N_i} \sum_{x_j \in R_i} I(y_j = k),$$

  i.e. the proportion of class k observations in the node

- We classify the observations in a (terminal) node i to class $k(i) = argmax_k\, p_{ik}$, i.e. the majority class in the node

- Then we measure the "impurity" over all terminal nodes

- Popular choices for an "impurity" measure at node i are

Misclassification error: $D_i = \frac{1}{N_i} \sum_{x_j \in R_i} I(y_j \neq k(i)) = 1 - p_{i,k(i)}$

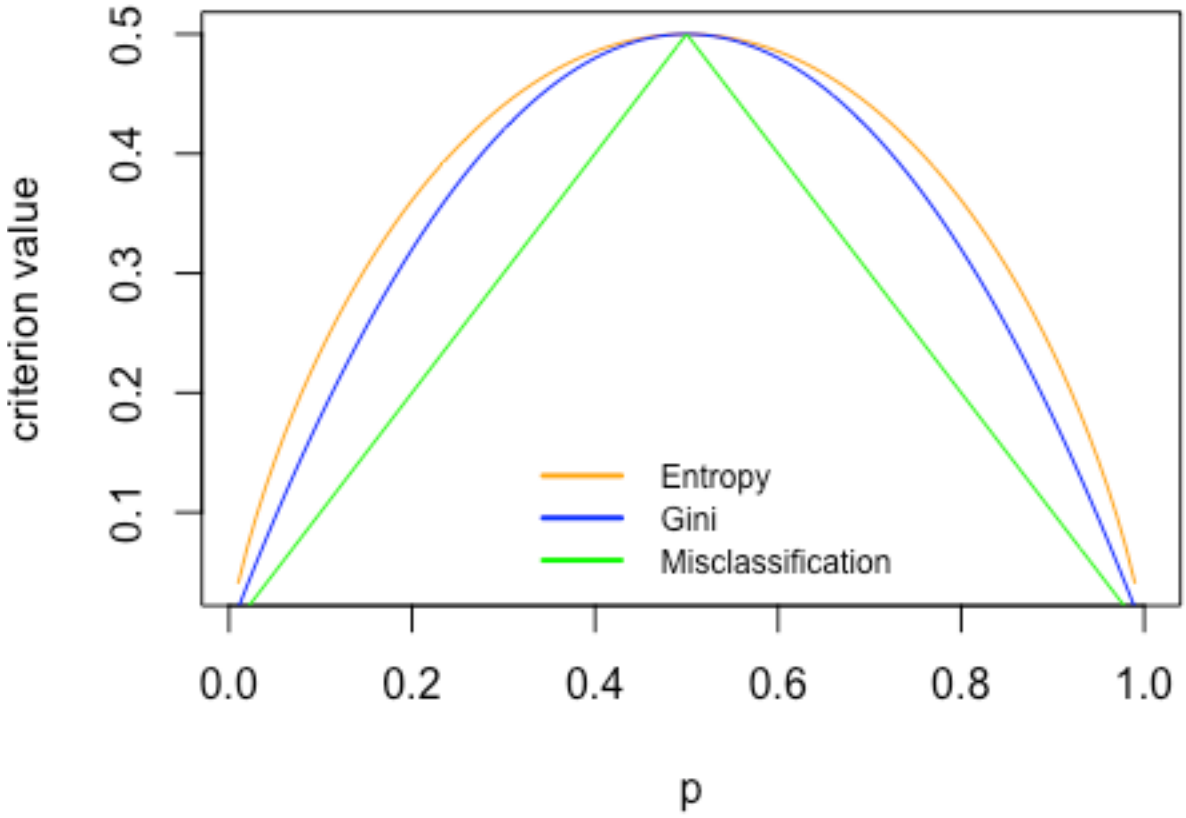
Entropy: $D_i = -\sum_{k=1}^{K} p_{ik} \log p_{ik}$

Gini index: $D_i = \sum_{k=1}^{K} p_{ik}(1 - p_{ik}) = 1 - \sum_{k=1}^{K} (p_{ik})^2$

Deviance: $D_i = -2 \sum_{k=1}^{K} n_{ik} \log p_{ik}$

(n_{ik} is the number of observations of class k in node i).

- For 2 classes, if p is the proportion in the second class, the first three measures are shown below (as functions of p). All three are similar, but entropy and the Gini index are differentiable, and hence more amenable to numerical optimization.



Example 2. Consider data `Carseats` from ISLR, Lab 8.3 (Decision trees). We are going to create a binary variable `High` from `Sales` (if Sales>8 then High=='Yes').
(In R: Use package rpart again to build a classification tree.)

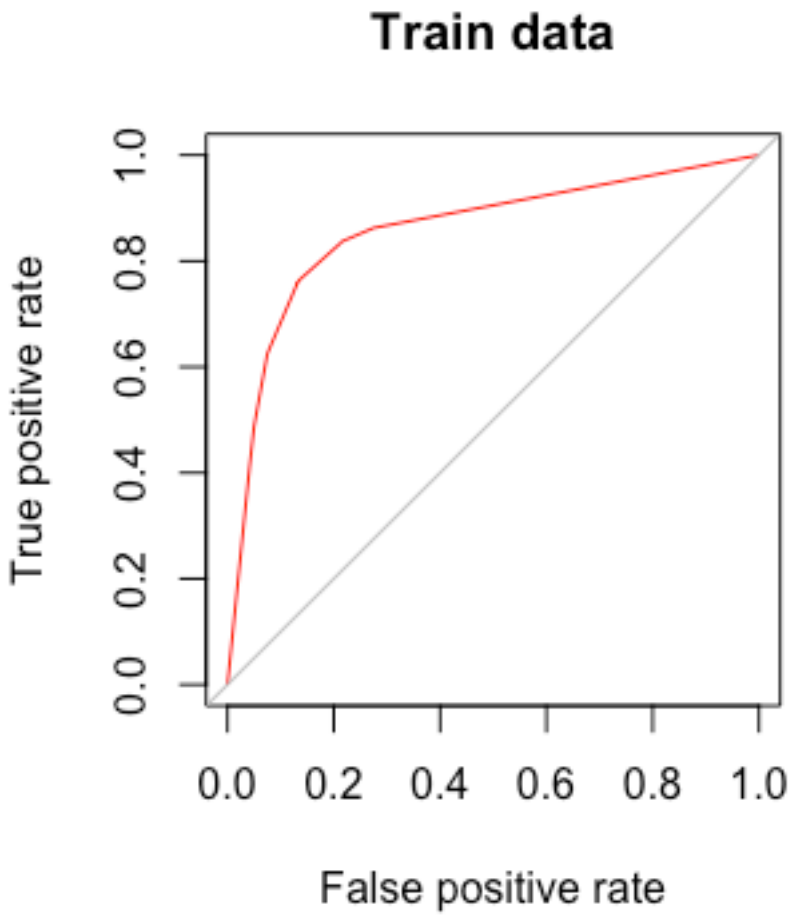
Since we have 400 observations, we split then randomly in half into train and test data. The model build on the train data is evaluated by

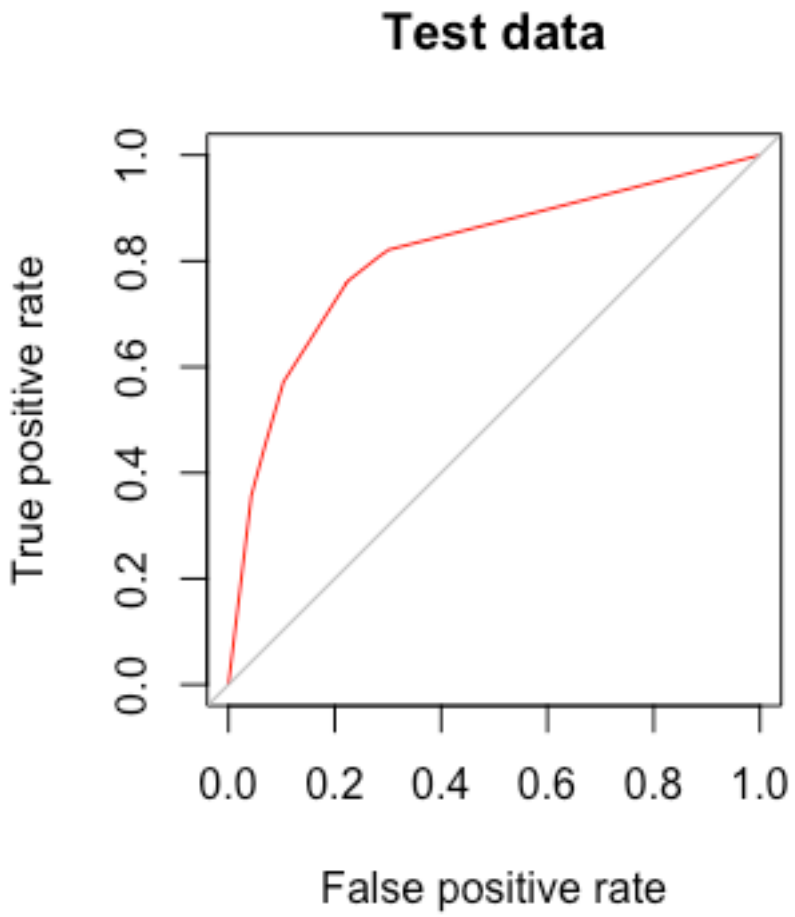
a) misclassification rate:

train: 17.5% test: 23%

b) AUC (the ROC curves are on next slide)

train: 0.8582 test: 0.8135


Train data
True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate


Test data
True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate

- 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 predict correctly approach leads to correct predictions for around 77% of the high sales in the test data set which with $AUC = 0.8135$ is a decent result

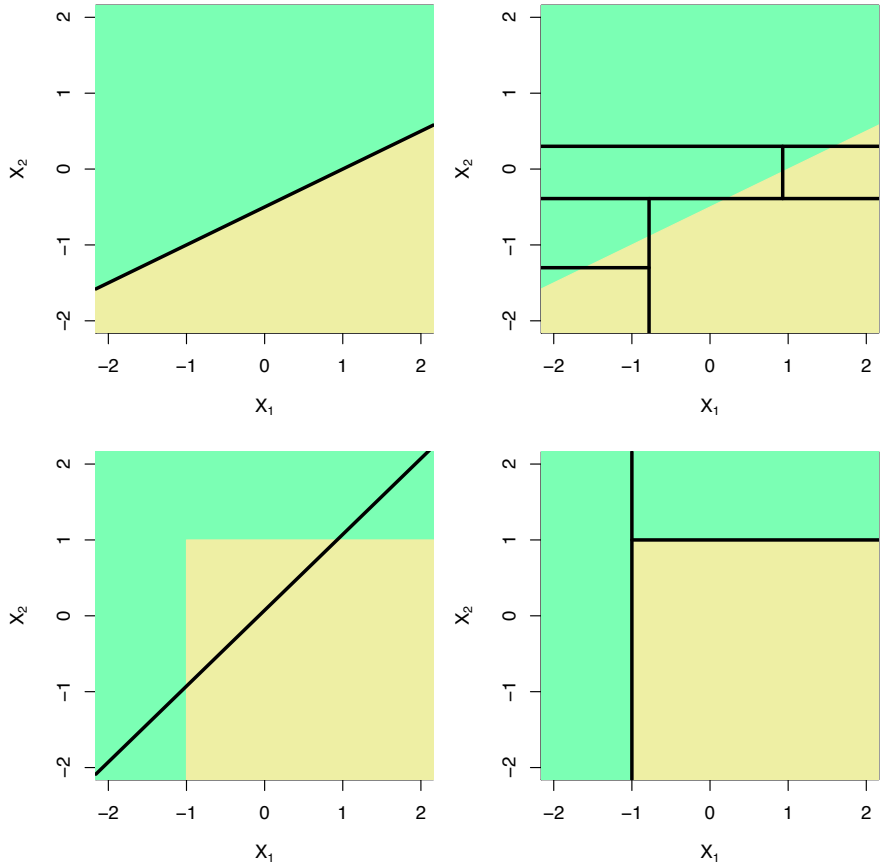
- Check out also the R code where trees are built with package `tree`
  - `Boston` – regression tree
  - `Carseats` – classification tree

Trees vs. Linear Models

- Which model is better?
    - If the relationship between the predictors and response is linear, then classical linear models such as linear regression would outperform regression trees
    - On the other hand, if the relationship between the predictors is non-linear, then decision trees would outperform classical approaches

Trees vs. Linear Model: Classification Example

- Top row: the true decision boundary is linear
  - Left: linear model (good)
  - Right: decision tree
- Bottom row: the true decision boundary is non-linear
  - Left: linear model
  - Right: decision tree (good)



To summarize: Decision trees for regression and classification have a number of advantages over the more classical approaches:

- Trees are very easy to explain (even easier than linear regression)
- Decision trees perhaps mirror well human decision-making
- Trees can be displayed graphically, and are easily interpreted even by a non-expert
- Trees can easily handle qualitative predictors without the need to create dummy variables. Missing values are easily handled too

Unfortunately, trees generally lack predictive accuracy at the level of some of the other regression and classification approaches can provide

Reading:
ISLR: 8.1, 8.3.1
ESLII: 9.2

APPENDIX: Individual tree fitting in R

Example 1 (data in package `faraway`). Study the relationship between ozone concentration and meteorology in LA. Start with some EDA

```
> data(ozone)
> summary(ozone)
> pairs(ozone,pch=".")
```

On the scatter plots produced by `pairs` non-linear relations are noticed (the graph is pretty busy; the 1st row of the graphs corresponds to `O3` which is the response variable).

Fit a tree using package `rpart` (other options `party`, `tree` etc.)

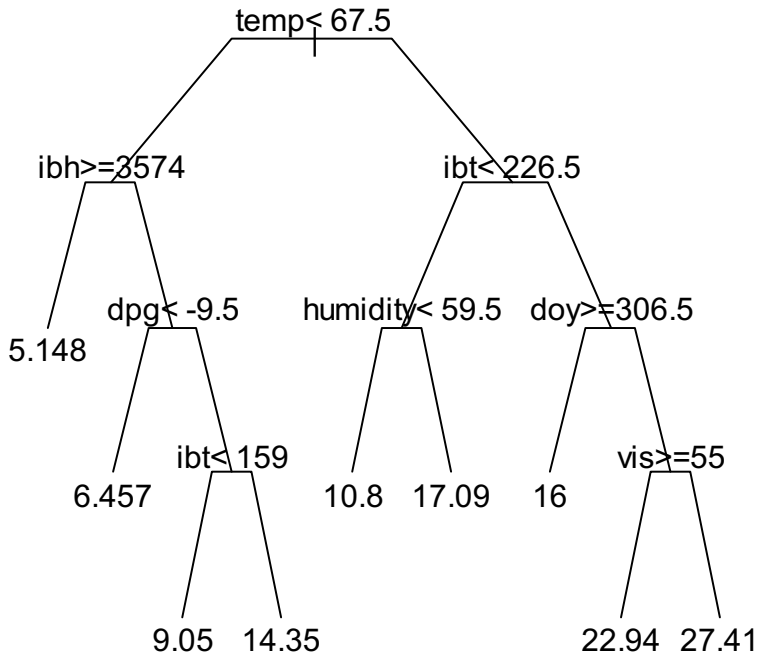
```
> library(rpart)
> roz <- rpart(O3 ~ . , ozone)
> print(roz,digits=2)
```

And then plot the regression tree

```
> plot(roz, compress=T, uniform=T, branch=0.4, margin=.10)
> text(roz)
```

On the plot below, "branch out left" if the condition at a node is true, otherwise follow the right branch.

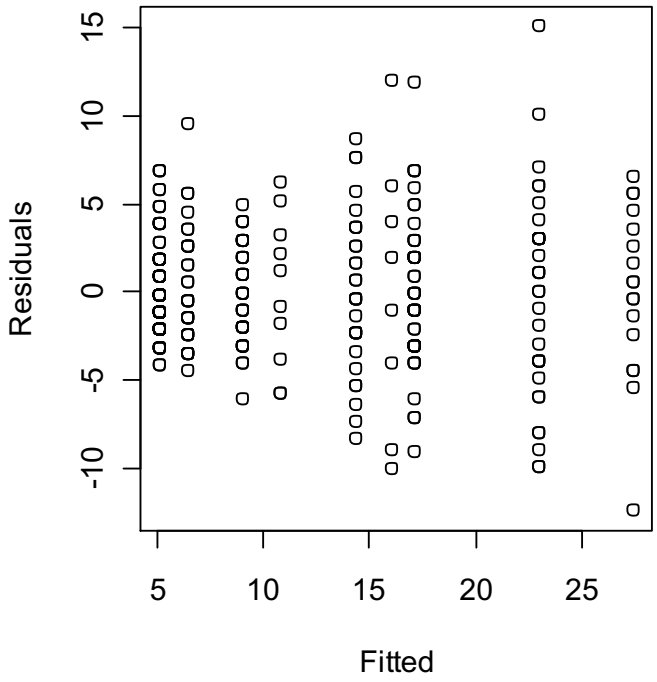
```
n= 330

node), split, n, deviance, yval
      * denotes terminal node

 1) root 330 21120.0 11.780
   2) temp< 67.5 214  4114.0  7.425
     4) ibh>=3574 108   689.6  5.148 *
     5) ibh< 3574 106  2294.0  9.745
      10) dpg< -9.5 35   362.7  6.457 *
      11) dpg>=-9.5 71  1366.0 11.370
        22) ibt< 159 40   287.9  9.050 *
        23) ibt>=159 31   587.1 14.350 *
   3) temp>=67.5 116  5478.0 19.800
     6) ibt< 226.5 55  1277.0 15.950
      12) humidity< 59.5 10   167.6 10.80 *
      13) humidity>=59.5 45   785.6 17.09 *
     7) ibt>=226.5 61  2646.0 23.280
      14) doy>=306.5 8   398.0 16.000 *
      15) doy< 306.5 53  1760.0 24.380
        30) vis>=55 36  1150.0 22.940 *
        31) vis< 55 17   380.1 27.410 *
```

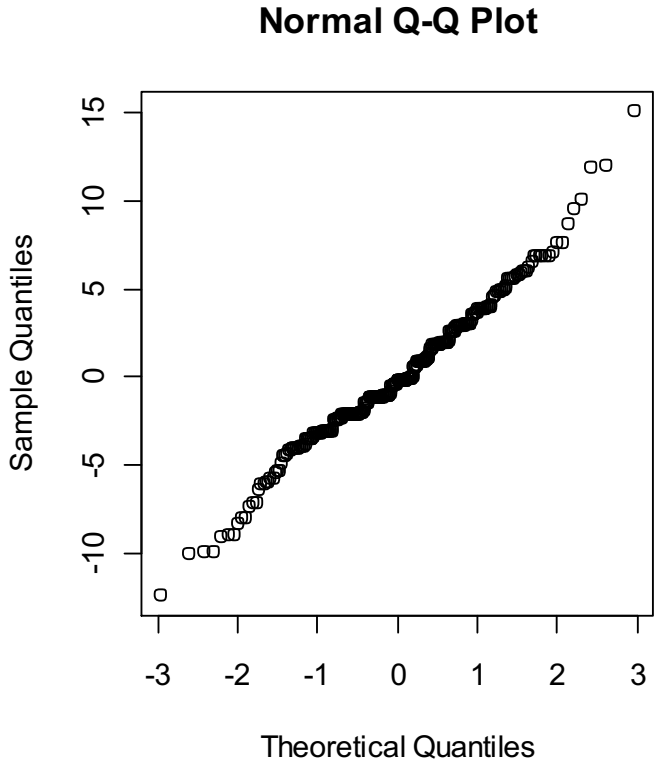


The 1st split on temperature produces a large reduction in RSS – from 21120 at the root to $4114 + 5478 = 9592$ at nodes $2 \& 3$. Some of the subsequent splits don't do so well.

We see that high levels of ozone are associated with high temperatures.
A regression tree is a regression model so fit diagnostics are possible:

```
> plot(predict(roz),residuals(roz),xlab="Fitted",ylab="Residuals")
> qqnorm(residuals(roz))
```





- No problems are observed in this case.
- But in general outliers may conceal themselves (and be influential on the fit) as with linear models.
- We can predict the response for a set of new values, e.g. the medians of the predictors:

```
> (x0 <- apply(ozone[,-1],2,median))
      vh     wind humidity     temp     ibh      dpg     ibt      vis      doy 
  5760.0      5.0     64.0     62.0   2112.5     24.0   167.5   120.0    205.5 

> predict(roz, data.frame(t(x0)))
       1 
14.35484 
```

- The same value can be found on the graph of the tree following the splits.

Tree pruning

- When to stop "growing" the tree?
- The greedy strategy is to keep growing the tree until the reduction in overall cost (RSS for the regression tree) is not reduced by more than some small number ε. Of course it's not easy to determine a good ε.
- Another strategy is to grow a large tree T_0 (with only restriction to have at least 5 observations in the "leaves") and then prune it back. Pruning means that given a tree of size *n*, the best tree of size *n-1* (the one increasing the criterion by the least amount) is determine by combining adjacent nodes in all possible ways.
- This is similar to backward elimination in variable selection for linear regression. It is achieved by using a cost-complexity function defined on next slide as the criterion in the pruning process. This function is used to reduce the number of the considered trees.

- For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T| \qquad (8.4)$$

(the cost-complexity function) is as small as possible. Here

$|T|$ = # of terminal nodes (leaves) of T

R_m = subset corresponding to the m-th terminal node

$\hat{y}_{R_m}$ = predicted response, i.e. the mean of all training obs "in" R_m

α = tuning parameter controlling the trade-off between the subtree complexity and its fit (on the train data).

- When $\alpha = 0$, then the subtree T will simply equal the large tree T_0 (then (8.4) just measures the training error).

- However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity (8.4) will tend to be minimized for a smaller subtree.

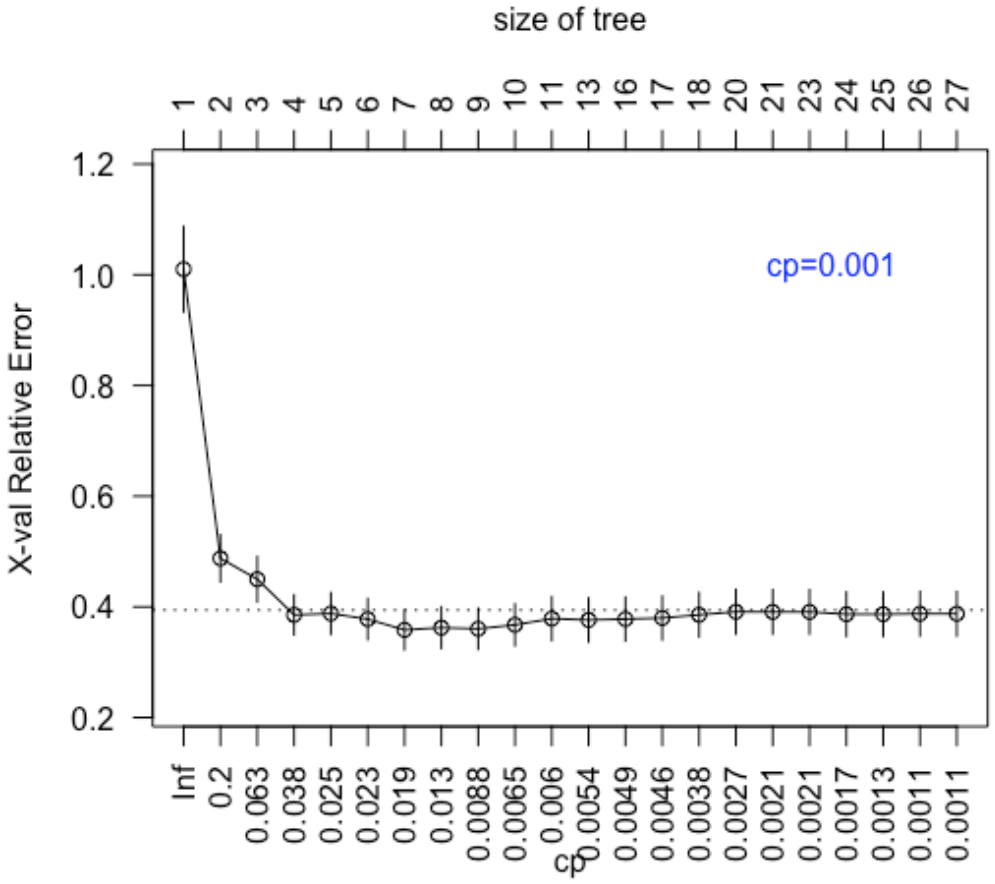
- CV is used to select from the sequence of trees (best for each size).
- The method is based on random sampling, so to exactly replicate the output below we need to set the seed:

```
> set.seed(12345)
> roze <- rpart(O3 ~.,ozone,cp=0.001)
> printcp(roze,digits=3)

          CP nsplit  rel.error  xerror  xstd
1  0.54570      0      1.000  1.010 0.0772
2  0.07366      1      0.454  0.488 0.0426
3  0.05354      2      0.381  0.450 0.0409
4  0.02676      3      0.327  0.386 0.0360
5  0.02328      4      0.300  0.388 0.0376
6  0.02310      5      0.277  0.378 0.0370
7  0.01532      6      0.254  0.358 0.0359
8  0.01091      7      0.239  0.362 0.0375
9  0.00707      8      0.228  0.360 0.0370
...
```

cp = *α/(RSS at the root tree)* - smoothing parameter;
rel error = *(RSS of the tree)/(RSS of the root tree)*;
xerror = *CV error* also scaled by the RSS of the root (null) tree.

- Further interpretation of the above amounts follows.

```
> plotcp(roze,las=2)  # cp=0.001
```



Smaller value of the complexity parameter (default is $cp = 0.01$) allows to grow a larger tree.

The 10-fold partitions needed for CV are random so xstd, the standard deviation of the CV error, is given in the output and useful too.

We can choose size of the tree in one of these ways:

1) Minimize the value of `xerror` and select the corresponding value of `CP`:

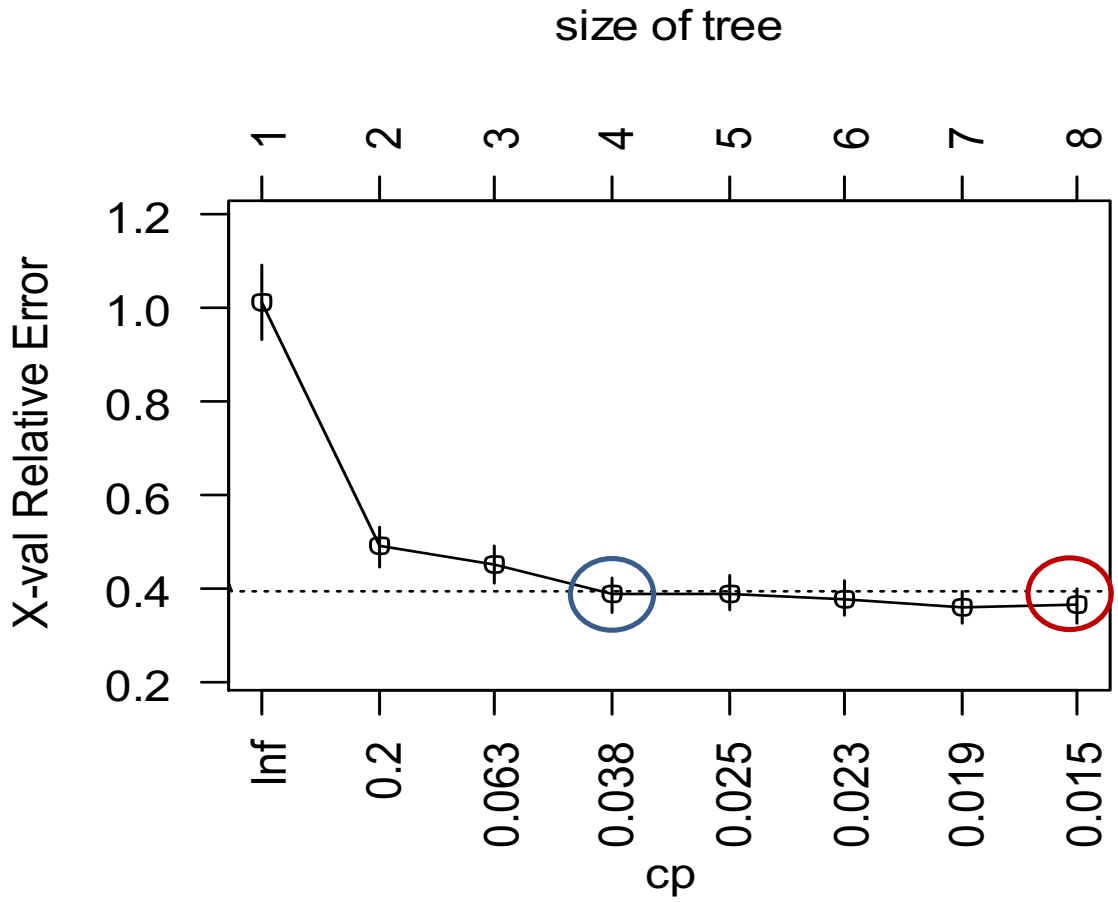
```
> rozr <- prune.rpart(roze, 0.01532) # nsplit=6
```

Instead of the complexity parameter (cp) it's easier to track the corresponding # of splits ($nsplit$). The number of all nodes (nn), number of terminal nodes (ntn) and $nsplit$ are related: $nn = 1 + 2 * nsplit,\ ntn = nsplit + 1.$

2) Another approach is to select the smallest tree with CV error within $1\ std$ error of the minimum, i.e. where $\text{xerror} \leq 0.358 + 0.036 = 0.394$. This would be the tree with 3 splits (`rozr2`). This approach is presented graphically on next slide.

Choice 1) is highlighted in yellow in table above, and choice 2) – in grey

```
> plotcp(rozr,las=2)  # cp=0.0153
```



Let's see what is the accuracy of this tree measured by R^2:

```
> rozr <- prune.rpart(roz,0.01532) #
> 1-sum(residuals(rozr)^2)/sum((ozone$O3-mean(ozone$O3))^2)
```

```
[1] 0.7614
```

Although the fit produced by the tree is piecewise constant over the regions defined by the partitions, in this example it outperforms the OLS linear regression that can also be built.