Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D
Professor
Scheller College of Business

Introduction to Tree Based Methods

Lessons in this Module

A. Introduction to Tree Based Methods
B. Regression Trees
C. Regression Tree Examples
D. Classification Trees
E. Carseats Example for Classification Trees
F. Bagging
G. Random Forests
H. Boosting
I. Bagging, Random Forests, and Boosting Examples

* The materials in this module are from Chapter 8 of the ISLR textbook

Decision Trees: The Basics

- The main idea is that we segment the predictor space into simple regions.
- To make a prediction for an observation, find the region that the observation belongs to and then (typically) use the mean of the training data in that region as the predicted value.
- The set of splitting rules that are used to segment the predictor space can be summarized (and visualized) by a tree. Therefore, such approaches are also called **decision tree methods**.
- Decision tree methods are simple to implement and useful for interpretation.
- They can be used for **classification** or **regression** problems, hence they are now also known as **Classification and Regression Trees (CART)** methods.

Decision Trees: The Basics

- We have a set of predictors $X_1, X_2, \dots , X_p$ and we want to predict a response Y (quantitative or categorical value). We do this by growing a binary tree (i.e., each **internal node** has two branches).
- At each internal node of a tree, we apply a test to one of the predictors, say X_i
- Let's assume the test to be is $X_i < t$?
    - If this test is true (i.e., X_i *'s* value is $< t$) then we go to the left-hand branch.
    - If this test is false (i.e., X_i *'s* value is $>= t$), then we go to the right-hand branch.
- Eventually, when we come to a **terminal** or **leaf node**, we make a **prediction**.
- This prediction will aggregate or average all the training data which reach that leaf node.

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D
Professor
Scheller College of Business

Regression Trees

Regression Trees

- We use a simple example that uses the ***Hitters*** dataset in the ISLR package
- The objective is to predict a baseball player's Salary (measured in thousands of dollars) based on
  - Years (#of Years playing in the major leagues) and
  - Hits (# of Hits that he made the previous year)
- We clean up the data by dropping observations with missing Salary values.
- We also log-transform Salary so that its distribution is more normally-shaped.

Regression Trees: *Hitters* Example

Figure 8.1 shows a regression tree for predicting the log salary of a baseball player.

- The split at the top node of the tree has two branches.
- The left-hand branch has observations for players with Years < 4.5, for which the mean log salary is 5.11. Therefore, we make a prediction in thousands of dollars, i.e., $165,174
- How would you interpret the right branch of the tree?

Figure 8.1

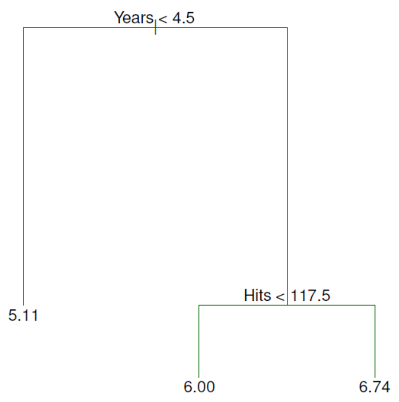


Figure 8.2: The Three-Region Partition for the Hitters Data from Figure 8.1

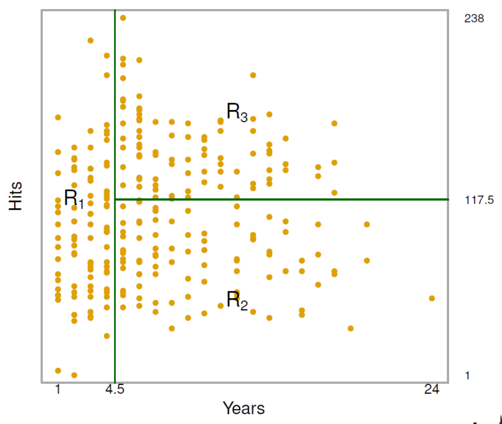
The three regions can be written as

1. $R_1 = \{X \mid \text{Years} < 4.5\}$
2. $R_2 = \{X \mid \text{Years} >= 4.5, \text{Hits} < 117.5\}$
3. $R_3 = \{X \mid \text{Years} >= 4.5, \text{Hits} >= 117.5\}$

Their respective predicted salaries are:

1. $\$1{,}000 * e^{5.107} = \$165{,}174$ for R_1
2. $\$1{,}000 * e^{5.999} = \$402{,}834$ for R_2
3. $\$1{,}000 * e^{6.740} = \$845{,}346$ for R_3

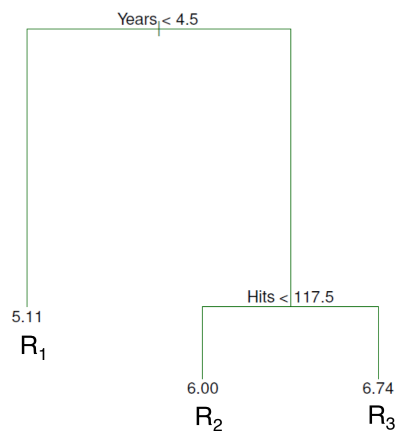
Figure 8.2



Terminology

- The regions R_1, R_2, and R_3 are known as **terminal nodes** or **leaves** of the tree.
- Decision trees are usually drawn upside down since the leaves are at the bottom of the tree.
- **Internal nodes** refer to the points in the tree where the predictor space is split.
- In Figure 8.1, the two internal nodes are Years < 4.5 and Hits < 117.5
- The segments of the trees that connect nodes are called **branches**.

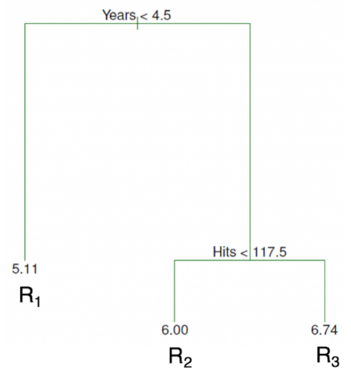
Figure 8.1



How to Interpret Figure 8.1

- Years is the most important predictor in determining Salary. Players with less experience earn lower salaries than players with more experience.
- For a less experienced player, the number of hits made in the previous year seems to play little role in his salary.
- For players with equal to or more than 4.5 years of experience, the number of hits in the previous year does influence their salary, with players who made more hits tending to have higher salaries.
- The regression tree in Figure 8.1 is likely an over-simplification of the true relationship between Years, Hits, and Salary but is easier to interpret and has a helpful graphical representation.

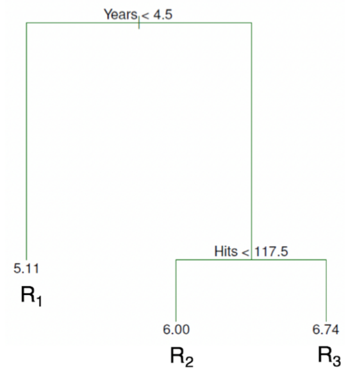
Figure 8.1



How is a Regression Tree Built?

- How a regression tree is built:
  1. We divide the predictor space (i.e., the set of possible values for the predictor variables) into *J* distinct and non-overlapping regions, $R_1, \dots, R_J$
  2. For every observation that falls into a region R_j, we make the identical prediction, which is the mean of the response values for the training observations in that region.
- Using the example of Figure 8.1, for a given observation in region R_1, we will predict a salary of $165,174.
- The question remains – how are these regions – $R_1, \dots, R_J$ – actually constructed?

Figure 8.1



GTx

Constructing the Regions

- In theory, the regions could have any shape.
- However, for simplicity and ease of exposition of the resulting predictive model, we opt to divide the predictor space into high-dimensional rectangles or boxes.
- The goal is to find boxes $R_1, \dots, R_J$ that minimize the SSE (Sum of Squared Errors)

$$\sum_{j=1}^{J} \sum\nolimits_{i \in R_j} (y_i - \hat{y}_{R_j})^2 ,$$

  where $\hat{y}_{R_j}$ is the mean response for the training observations within the *j*th box
- Note that the ISLR book uses the term RSS (Residual Sum of Squares) instead of SSE. They are identical.

GTx

Top-Down Greedy Approach

- Computationally, it is infeasible to consider every possible partition of the feature space into J boxes.
- Hence, we take a top-down, greedy approach called **recursive binary splitting**
    - Called top-down because it begins at the top of the tree with all observations in a single region and then successively splits the predictor space.
    - Every split has two new branches further down the tree.
    - It is greedy because, at each step of the process of tree-building, the best split is made at that particular step rather than looking ahead and picking a split that will lead to a better tree at a later step.

Recursive Binary Splitting

- Select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X_j \,/\, X_j < s\}$ and $\{X_j \,/\, X_j >= s\}$ leads to the greatest possible reduction in SSE.
- We have to consider all predictors $X_1, X_2, \ldots, X_p$ and all possible values of s for each of the predictors, and then choose the predictor and its cutpoint such that the resulting tree has the lowest SSE. This means that for any j and s, we define the pair of half-planes $R_1(j,s) = \{X_j \,/\, X_j < s\}$ and $R_2(j,s)$ $\{X_j \,/\, X_j >= s\}$, and we seek the value of j and s that minimize the equation

$$\sum_{i;x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i;x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

  where $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j,s)$ and

  where $\hat{y}_{R_2}$ is the mean response for the training observations in $R_2(j,s)$
- Finding the values of j and s that minimize the equation can be done quickly, especially when p is not too large.

Recursive Binary Splitting

- We repeat the process looking for the best predictor and cutpoint to split the data further so as to minimize the SSE within each of the split regions.
- Note that we are now splitting one of the two previously identified regions so we now have three regions.
- Again, we look to split one of these three regions further, so as to minimize SSE.
- The process continues until a stopping criterion is reached; for example, we could continue until no region has more than five observations.
- Once the regions $R_1, \ldots, R_J$ have been created, we predict the response for a given test observation using the means of the training observation in the region in which that test observation belongs.

Tree Pruning

- Recursive binary splitting may produce good predictions on the training set but is likely to overfit the data, leading to poor performance on the test set. This is because the tree might be too complex.
- A smaller tree with fewer splits (i.e., fewer regions $R_1, \ldots, R_J$) might lead to
  - lower variance
  - better interpretation at the cost of a little bias
- One alternative is to build a tree as long as the decrease in SSE due to each split exceeds a (high) threshold.
- This will result in smaller trees but could be short-sighted since an initial seemingly-worthless split might be followed by a very good split later on.

Pruning to Build a Better Tree

- A better strategy is to grow a vary large tree (call it T_0) and then prune in back to obtain a subtree.
- How does one find the best subtree?
  - Our aim is to find a subtree that leads to the lowest test error rate.
  - Given a subtree, we can estimate the test error rate using the validation set approach or using cross-validation (CV).
  - But estimating the CV for every possible subtree would take a long time since there are so many subtrees.
  - So we need a way to select a small set of subtrees to consider.

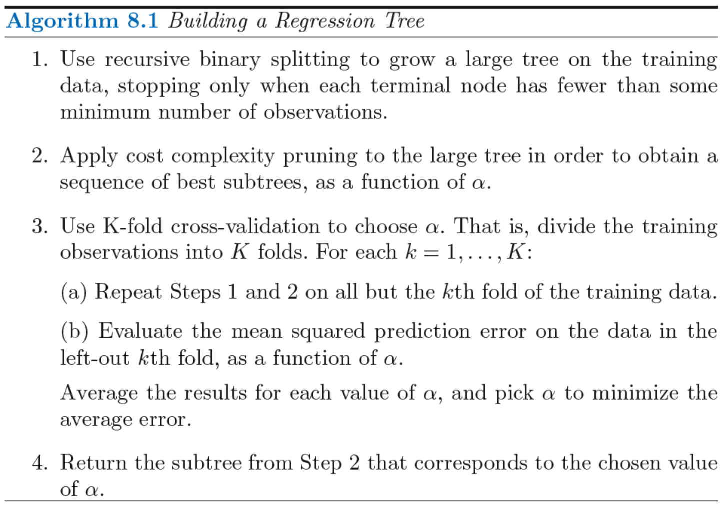
Cost Complexity Pruning

- **Cost complexity pruning** or **weakest link pruning** is a way to select a small set of subtrees to consider.
- Instead of examining every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α
- ITI indicates the number of terminal nodes of a tree T
- R_m is the rectangle or box corresponding to the *m*th terminal node
- $\hat{y}_{R_m}$ is the mean response for the training observations within the *m*th box
- α controls a trade-off between the subtree's complexity and its fit to the training data and for each value of α there is a subtree $T \in T_0$, such that

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

is as small as possible.

Cost Complexity Pruning



Algorithm 8.1 *Building a Regression Tree*

```
1. Use recursive binary splitting to grow a large tree 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 $\alpha$.

3. Use K-fold cross-validation to choose $\alpha$. 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 $k$th fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the
   left-out $k$th fold, as a function of $\alpha$.

   Average the results for each value of $\alpha$, and pick $\alpha$ to minimize the
   average error.

4. Return the subtree from Step 2 that corresponds to the chosen value
   of $\alpha$.
```

* Chapter 8 of the ISLR textbook

Notes on the Algorithm

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

- When $\alpha = 0$, then the subtree T will simply be equal to T_0, because Equation 8.4 measures the training error.
- As α increases, there is a price to pay for having a tree with many terminal nodes and so Equation 8.4 will be minimized when we have a smaller tree.
- Note that Equation 8.4 is similar to the Lasso which was used to control the complexity of the linear model.
- As we increase α from 0 in Equation 8.4, branches get pruned from the tree in a nested and predictable manner, which means obtaining the whole sequence of subtrees as a function of α is easy.

Quiz

A regression tree can approximate only linear relationships, as each split is linear.

A. True
B. False (overall boundary after second split is non-linear)
C. Depends on how we prune
D. Depends on which variable is chosen first

Correct Answer: B

Which of these is a FALSE statement about Decision Trees?

A. At each internal node of a tree, we apply a test to one of the predictors, say X_i
B. When we come to a leaf node, we make a prediction
C. Decision trees are usually drawn with the leaves at the top of the tree
D. Recursive binary splitting may produce good predictions on the training set but is likely to overfit the data, leading to poor performance on the test set
E. Every split has two new branches further down the tree

Correct Answer: C

GTx

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D
Professor
Scheller College of Business

Regression Tree Examples

Regression Tree for the *Hitters* Data Set

- There are several R packages for regression trees
  - tree() and rpart() are two packages
- Remove observations with missing data (Hitters or Salary)
- Log-transform Salary so that its distribution is closer to normal
- Then build a tree

```
library(ISLR)
library(tree)
attach(Hitters)
# remove NA values
Hitters <- na.omit(Hitters)
Salary <- na.omit(Salary)
# put salary on log scale and fit
reg. tree
treefit <- tree(log(Salary) ~ Years
+ Hits, data=Hitters)
```

Summary of Fitted Tree

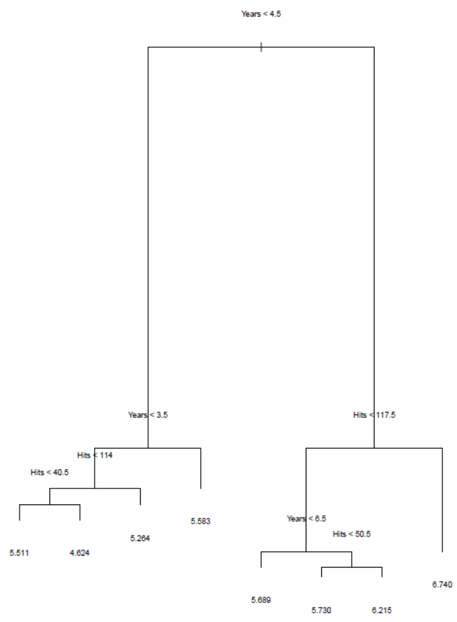
```
summary(treefit)
##
## Regression tree:
## tree(formula = log(Salary) ~ Years + Hits,
data = Hitters)
## Number of terminal nodes: 8
## Residual mean deviance: 0.2708 = 69.06 / 255
## Distribution of residuals:
##    Min.  1st Qu. Median Mean   3rd Qu. Max.
## -2.2400 -0.2980 -0.0365 0.0000 0.3233 2.1520
```

Do a summary of treefit – the fitted regression tree.

There are 8 terminal nodes or leaves of the tree. Here “deviance” is just the sum of squared errors; this gives us a MSE = 69.056/(263-8) = 69.06/255 = 0.2708

Plot the Tree

```
# plot the regression tree.
plot(treefit)
text(treefit,cex=0.75)
# note leaf labels are log salary!
```



GTx

Another Regression Tree Example

```
### Regression Tree using the Boston data set in MASS library

set.seed (1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston=tree(medv~.,Boston, subset=train)
summary (tree.boston)
```

GTx

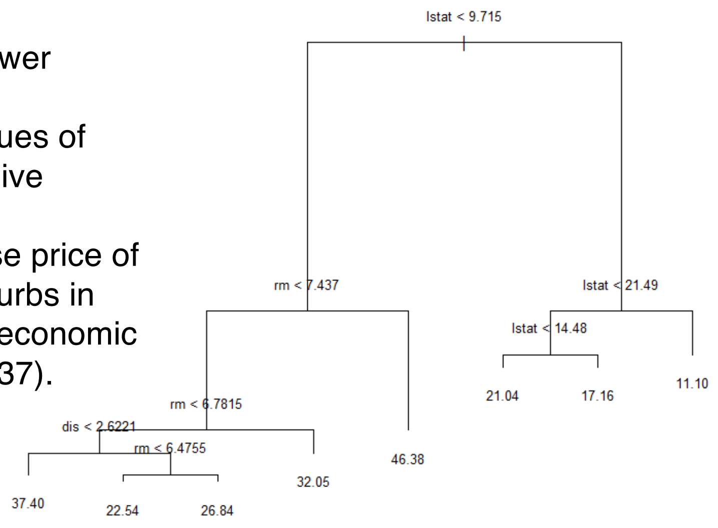
Another Regression Tree Example…

```
> summary (tree.boston)

Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "lstat" "rm"    "dis"
Number of terminal nodes:  8
Residual mean deviance:  12.65 = 3099 / 245
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-14.10000  -2.04200  -0.05357   0.00000   1.96000  12.60000
```

- Note that only three of the variables (`"lstat" "rm" "dis"`) have been used in constructing the tree
- As in the previous example, in the context of a regression tree, the deviance is simply the sum of squared errors for the tree

Plot(tree.boston)

- The variable **lstat** measures the percentage of individuals with lower socioeconomic status.
- The tree indicates that lower values of lstat correspond to more expensive houses.
- The tree predicts a median house price of \$46,380 for larger homes in suburbs in which residents have high socioeconomic status (lstat<9.715 and rm>=7.437).

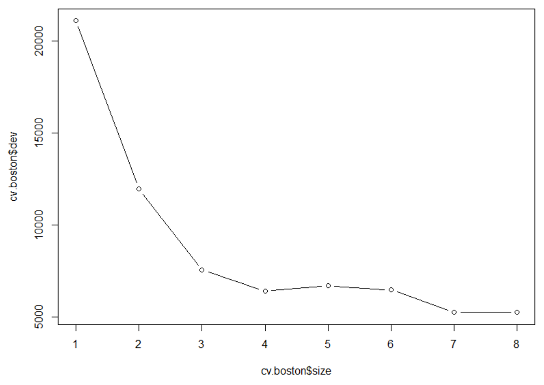


Cross Validation and Pruning

- The tree package contains functions **prune.tree()** and **cv.tree()** for pruning trees by cross-validation.
- **prune.tree()** uses as input a tree that you have fitted by tree() and determines a nested sequence of subtrees of the supplied tree by recursively "snipping" off the least important splits.
- Evaluation can be done on new data or on the training data.
- The default is method="deviance", which fits by minimizing the mean squared error (for continuous responses) or the negative log likelihood (for discrete responses).
- **cv.tree()** runs a K-fold cross-validation experiment to find the deviance or number of misclassifications as a function of the cost-complexity parameter.

Cross Validation and Pruning

```
# use the cv.tree() function to see
if pruning will improve performance

cv.boston =cv.tree(tree.boston)
plot(cv.boston$size,cv.boston$dev,
type='b’)
```

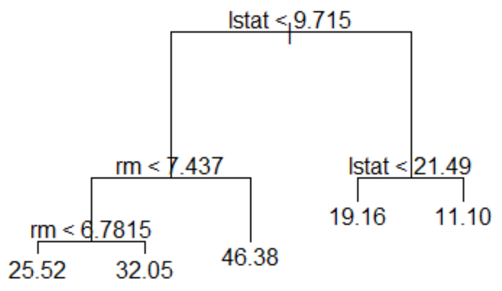


- The most complex tree is selected in this case by CV

Cross Validation and Pruning

- But, if we wish to prune the tree, we could do so as follows, using the **prune.tree()** function:

```
prune.boston
=prune.tree(tree.boston,best=5)
plot(prune.boston)
text(prune.boston,pretty =0)
```



Cross Validation and Pruning

```
# In keeping with the cross-validation results, we use the
# unpruned tree to make predictions on the test set:

 yhat=predict(tree.boston, newdata=Boston[-train,])
 boston.test=Boston[-train ,"medv"]
 plot(yhat,boston.test)
 abline(0,1)
 mean((yhat - boston.test)^2)
# [1] 15.26222
```

- In other words, the test set MSE associated with the regression tree is 25.05. The square root of the MSE is therefore around 5.005, indicating that this model leads to test predictions that are within around $5,005 of the true median home value for the suburb.

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D
Professor
Scheller College of Business

Classification Trees

Classification Trees

- Classification trees are very similar to regression trees.
- The difference is that the response (Y variable) is qualitative rather than quantitative.
- For a classification tree, the predicted response for an observation is to put it in the most commonly occurring class of training observations in the region to which that observation belongs.
- We want to know how to interpret the results of a classification tree

Growing a Classification Tree

- The process of growing a classification tree is very similar to that of growing a regression tree.
- Recursive binary split is used to grow the tree.
- However, we can't use the SSE as a criterion for making splits.
- **Classification error rate** is used as the criterion for making splits.
- The classification error rate is defined as the fraction of the training observations in a region that do not belong to that region's most common class:
  $$E = 1 - \max_k \hat{p}_{mk}$$
  where $\hat{p}_{mk}$ represents the proportion of training observations in region *m* that belong to class *k*
- However, the classification error is not sufficiently sensitive to the process of growing a tree. Two other measures will be discussed next.

The Gini Index

- The **Gini index** is measure of the total variance across all the *K* classes (in the *m*th region) and is defined as:
  $$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$
- The Gini index has a very small value when all of the $\hat{p}_{mk}$'s are close to 0 or 1.
- It is also referred to as a measure of **node purity**. A small value of *G* indicates that a node mostly contains observations from a single class.

Entropy

- An alternative measure to the Gini index is **entropy**, which is defined as:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log \hat{p}_{mk}$$

- Since $0 \leq \hat{p}_{mk} \leq 1,$ we get $0 \leq -\hat{p}_{mk} log \hat{p}_{mk}$
- Entropy will have a very small value if all of the $\hat{p}_{mk}$'s are close to 0 or 1.
- This means that entropy will also be close to 0 if the *m*th node is pure.
- As it happens, the Gini index and entropy are very similar numerically.

Using These Measures

- Either the Gini index or Entropy is used to evaluate the quality of a particular split when building a classification tree since they are very sensitive to purity compared to the classification error rate.
- They could also be used for pruning a tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is desired.

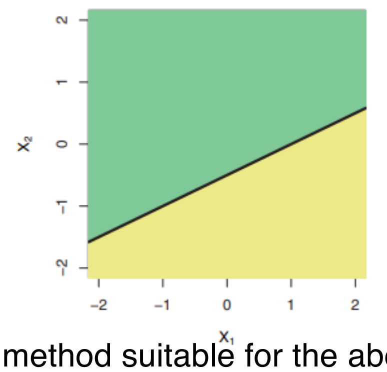
Deviance

- When one uses the summary() function on a classification tree, **deviance** is reported which is defined as:

$$-2\sum_m\sum_k n_{mk}\log\hat{p}_{mk}$$

  where n_{mk} is the number of observations.
- A small deviance is a sign that the tree provided good fit to the (training) data.
- The residual mean deviance reported is equal to the deviance divided by $n - |T_0|$

Quiz



Is a tree based method suitable for the above two-dimensional classification problem where each region is shaded a different color (see figure above)?

a) Yes
b) No, a linear model is more suitable
c) Depends on the tree that forms

Correct Answer: B

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D

Professor

Scheller College of Business

Carseats Example for Classification Trees

Carseats Dataset

- In the Carseats dataset in the ISLR package, Sales is a continuous variable and so we create a new binary variable (**High**) to build a classification tree

```
library(ISLR)
library(tree)
attach(Carseats)
# creating a binary variable
High <- ifelse(Sales <= 8, "No", "Yes")
# merge High with the rest of the Carseats
Carseats <- data.frame(Carseats, High)
```

- Note that 236 out of 400 records have Sales <=8 (so High is set to No for them) and the remaining 164 have Sales > 8 (so High is set to Yes for them)

Fit a Classification Tree

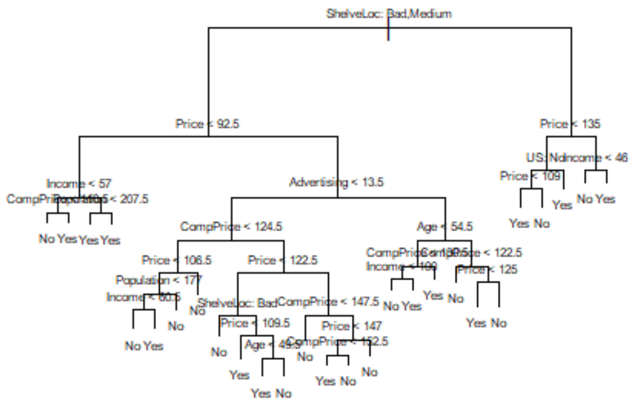
- Use the tree() function to fit a classification tree to predict **High** using all variables but Sales

```
a <-tree(formula = High ~ . - Sales, data = Carseats)
summary(a)

Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
"Advertising"
[7] "Age"         "US"
Number of terminal nodes:  27 # = |T_θ|)
Residual mean deviance:  0.4575 = 170.7 / 373 # (= deviance /(n -|T_θ|)
Misclassification error rate: 0.09 = 36 / 400 # (= training error rate)
```

Plot the Tree

```
plot(a)
text(a, cex=0.60,
pretty=0)
```

- The most important indicator of Sales seems to be Shelf location since the 1st branch separates **good** locations (on the right) from **bad** or **medium** ones (on the left).



Type the Name of the Tree

- If we type the name of the tree object,
  - R prints the output corresponding to each branch of the tree,
  - It displays the split criterion,
  - # of observations in that branch,
  - the deviance,
  - overall prediction for that branch (Yes or No),
  - and the fraction of observations in that that take the value of Yes or No.
  - Branches that lead to terminal nodes are marked with an asterisk *

The Classification Tree a First Few Lines of Output

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

  1) root 400 541.500 No ( 0.59000 0.41000 )
    2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
      4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
        8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
         16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
         17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
        9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
         18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
         19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
```

# at the root node 236/400 = 0.59 of the records have sales <= 8 (High = 0), hence the predicted value of Y = 0 (i.e., High = No)

# nodes 16, 17, 18, 19 are terminal nodes

Estimating the Test Error to Evaluate Tree Performance

- Split the observations into a training set and a test data.
- Build the tree using the training set.
- Evaluate the tree's performance on the test set using the predict() function.
- For a classification tree, the argument type="class" tells R to return the actual class prediction.

Prediction Using Test Data

```
# prediction performance on validation set
set.seed(2)
train=sample(1:nrow(Carseats), 200)
Carseats.test=Carseats[-train,]
High.test=High[-train]  # actual High values on the observations in the test set
tree.carseats=tree(High ~ . - Sales, data = Carseats, subset=train)
tree.pred=predict(tree.carseats, Carseats.test, type="class")
table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  86  27
      Yes 30  57
 (86+57)/200
[1] 0.715
```

The prediction accuracy is 71.5% for observations in the test data set

Pruning the Tree to Try to Improve Prediction

- Let's prune the tree to see if we get better prediction results.
- The function cv.tree() performs cross-validation to determine the optimal level of tree complexity; cost complexity pruning is how the sequence of trees are considered.
- The argument FUN=prune.misclass indicates that we want the classification tree error rate to guide the CV and pruning process rather that the default for the cv.tree() function which is the deviance.
- The cv.tree() function reports the # of terminal nodes of each tree considered (size), as well as the corresponding error rate and the value of the cost complexity parameter used (k, which corresponds to α in equation 8.4).

Cross Validation Using cv.tree

```
# cross validation
set.seed(3)
cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)
names(cv.carseats)
cv.carseats
```

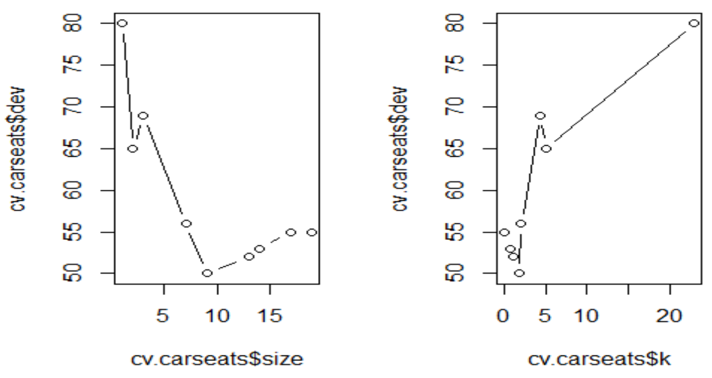
R Output of cv

dev corresponds to the cross-validation error rate in this example
# size corresponds to # of terminal nodes
# tree with 9 terminal nodes has lowest cv error rate of 50

```
> cv.carseats
$`size`
[1] 19 17 14 13  9  7  3  2  1
$dev
[1] 55 55 53 52 50 56 69 65 80
$k
[1]  -Inf  0.0000000  0.6666667  1.0000000  1.7500000  2.0000000
4.2500000  5.0000000
[9] 23.0000000
$method
[1] "misclass"
attr(,"class")
[1] "prune"         "tree.sequence"
```

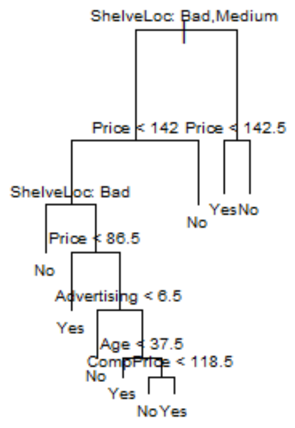
Plot Error Rate as a Function of Size and k

```
par(mfrow=c(1,2))
plot(cv.carseats$size,
cv.carseats$dev,type="b")
plot(cv.carseats$k,
cv.carseats$dev,type="b")
```



9-Node Tree Obtained from Pruning

```
# 9-node tree from pruning
prune.carseats=prune.misclass
(tree.carseats,best=9)
plot(prune.carseats)
text(prune.carseats,pretty=0,
cex=0.6)
```



Prediction Accuracy of the 9-Node Tree

```
tree.pred=predict(tree.carseats, Carseats.test,type="class")
table(tree.pred,High.test)

> table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  87  26
      Yes 29  58
> (87+58)/200   # this is the accuracy.
[1] 0.725
```

Pruning has produced a more interpretable tree with improved prediction accuracy

Prediction Accuracy of the 15-Node Tree

- If we increase the value of best to 15, we get a larger pruned tree with lower prediction accuracy.
- Verify this on your own!
- # 15-node tree from pruning

```
prune.carseats=prune.misclass(tree.carseats,best=15)
plot(prune.carseats)
text(prune.carseats,pretty=0,cex=0.6)
# prediction performance of this 15 node tree
tree.pred=predict(tree.carseats, Carseats.test,type="class")
table(tree.pred,High.test)
```

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D
Professor
Scheller College of Business

Bagging

Bagging

- When we use decision trees for prediction, they could suffer from **high variance**. If we split the training data into two parts at random, and fit a decision tree to both halves, the results we get could be quite different.
- On the other hand, a procedure with low variance will give similar results if applied repeatedly to distinct data sets; linear regression tends to have low variance if *n* is moderately larger than *p*
- **Bootstrap aggregation** or **bagging** is general-purpose technique for reducing the variance of a statistical learning method; it is particularly useful in the context of decision trees.

Bagging

- If we are given a set of independent observations $Z_1, \ldots, Z_n$ each with variance σ^2, the variance of the mean $\bar{Z}$ of these observations is given by σ^2 / n
- So, averaging a set of observations reduces variance.
- So, a natural way to reduce the variance and thus increase the prediction accuracy of a statistical learning method, is to
  - take many training sets from the population,
  - build a separate model using each training set,
  - and average the resulting predictions.

Bagging

- We calculate B predicted values $f^1(x), f^2(x), \ldots, f^B(x)$ using B separate training sets.
- We average them to get a single low-variance statistical learning model

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

- However, doing this process isn't practical because we mostly don't have multiple training sets.

Bagging

- But bootstrap gives us a workaround.
- We can take repeated samples from a single training data set.
- We then generate B different bootstrapped training sets.
- The learning method is trained on the *b*th bootstrapped training set to obtain the prediction $\hat{f}^{*b}(x)$
- We average all B predictions to get

$$\hat{f}_{bag} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

- This technique is called **bagging**.

Bagging and Regression Trees

- For regression trees, the bagging procedure works by building B regression trees using B bootstrapped training sets.
- The predictions of these B regression trees are then averaged.
- These trees are grown deep and not pruned.
- Each such tree has high variance but has low bias.
- Averaging the predictions of these B trees reduces variance.
- Bagging has been shown to give impressive accuracy improvements by combining together hundreds or thousands of trees into one procedure.

GTx

Bagging and Classification Trees

- The simplest way to apply bagging to classification trees is, for each test observation,
    - record the class predicted by each of the B trees for that observation, and
    - then take a majority vote
- In this context, a majority vote is the overall prediction of the most commonly occurring class among all B predictions for that test observation.

GTx

Out-of-Bag Estimation

- To estimate the test error of a bagged model, there is a straightforward way without using cross-validation or validation set approach.
- In bagging, as we have seen, trees are repeatedly fit to bootstrapped subsets of the observations.
- One can show that, on average, each bagged tree makes use of around 2/3 of the observations.
- The remaining 1/3 of the observations that are not used to fit a given bagged tree are called **out-of-bag** (OOB) observations.
- We can predict the response for the *i*th observation using each of the trees in which that observation was OOB.
- This will yield around B/3 predictions for that *i*th observation.

Out-of-Bag Estimation

- To obtain a single prediction for the *i*th observation, we can average these predicted responses (for regression trees) or take a majority vote (for classification trees).
- This gives a single OOB prediction for the *i*th observation
- Do this for all *n* observations and then compute the overall OOB MSE (for regression tree) or classification error (for classification tree).
- This resulting OOB error is a valid estimate of the test error for the bagged model because the response for each observation is predicted using only those trees that were fit not using that observation.
- It can be shown that with B sufficiently large, OOB error is virtually similar to LOOCV.
- This OOB approach is very convenient for bagging on large data sets where cross-validation would be computationally burdensome.

Variable Importance Measures

- We saw that bagging typically yields improved accuracy over prediction that only uses a single tree but it is difficult to interpret the model that results from bagging.
- It not possible to represent the model using a single tree because it is no longer clear which variables are most important to the procedure.
- Thus the improved prediction accuracy comes at the cost of interpretability but one can obtain an overall summary of the importance of each predictor using SSE (for bagging regression trees) or the Gini index (for bagging classification trees)
- When bagging regression trees, we can record the total amount the SSE is reduced due to splits over a given predictor, averaged over all B trees. A large average value indicates an important predictor
- Similarly, when bagging classification trees, we can add the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.

GTx

Quiz

When bagging regression trees, we can record the total amount the SSE is reduced due to splits over a given predictor, averaged over all B trees. A large average value indicates an important predictor

A. TRUE
B. FALSE
C. MAYBE

Correct Answer: A

GTx

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D
Professor
Scheller College of Business

Random Forests

Random Forests

- Random Forests are an improvement over bagged trees by way of a small tweak that **decorrelates** the trees.
- Similar to bagging, we build a number of trees on bootstrapped training samples.
- But in Random Forests, when building the decision trees, each time a split is considered, **a random sample of *m* predictors** is chosen from the full set of *p* predictors
- The split is allowed to use only of these *m* predictors.
- A fresh sample of *m* predictors is used at each split, and typically $m \approx \sqrt{p}$
- Note that in building a random forest, the algorithm is not even allowed to consider a majority of the *p* predictors.
- This has a good rationale.

Random Forests

- Assume that there is one very strong predictor in the data set along with a number of other moderately strong predictors.
- Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split.
- Therefore, all of the bagged trees will look similar to each other and thus their predictions will be highly correlated.
- Note that averaging many highly correlated quantities does not lead to as large a reduction in variance compared to averaging many uncorrelated quantities.
- This means that bagging will not lead to a substantial reduction in variance over a single tree in this scenario.

Random Forests

- Random Forests overcome the correlation problem by forcing each split to only consider a subset (typically $m \approx \sqrt{p}$) of the *p* predictors.
- Therefore, on average *(p – m)/p* of the splits will not even consider the strong predictor, thereby letting the other predictors have more of a chance
- This process can be considered to be **decorrelating** the trees, thus making the average of the resulting trees less variable and hence more reliable.
- Note that if *m = p*, then bagging is a special case of random forests.

Quiz

1. In Random forest, each tree is built on
   - A. All features, all data
   - B. Subset of features, all data
   - C. Subset of features, subset of data
   - D. All features, subset of data

Correct Answer: C

2. Random forest model is interpretable
   - A. Yes
   - B. No (it's an average of predictions from a number of trees)

Correct Answer: B

GTx

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D
Professor
Scheller College of Business

Boosting

Boosting

- Boosting is another approach to improve prediction results of a decision tree.
- It can also be used for statistical learning methods like regression or classification trees.
- We know how bagging works by making use of bootstrap to create multiple bootstrap data sets and then fitting a separate tree to each bootstrap training set. Then we combine the results of the decision trees to create a single prediction model.
- Boosting works similarly except trees are grown sequentially from previously grown trees but focus their attention on the misclassifications from the prior trees.
- Boosting also involves combining a large number of decision trees, $\hat{f}^1, \ldots, \hat{f}^B$

Boosting for Regression Trees

- The boosting approach learns slowly.
- The construction of each tree depends on the trees that have already been grown.
- Given the current model, we fit a decision tree to the residuals from that model.
- We fit a tree using the current residuals *r,* rather than the outcome *Y*, as the response.
- This new decision tree is added into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the (interaction depth) parameter *d* in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well.
- The shrinkage parameter λ slows the process even further, allowing more and different shaped trees to attack the residuals.
- Note that we are only covering boosting for regression trees. The details of boosting for classification trees are omitted from the ISLR textbook.

Algorithm 8.2. Boosting for Regression Trees

1. Set $\hat{f}(\mathrm{x}) = 0$ and $r_i = y_i$ for all i in the training data set.
2. For $b = 1, 2, \ldots, \mathrm{B}$, repeat:
   a) Fit a tree $\hat{f}^b$ with d splits (so it will have $\mathrm{d}+1$ terminal nodes) to the training data (X, r).
   b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(\mathrm{x})$$
   c) Update the residuals:
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
3. Output the boosted model:
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

Boosting's three Tuning Parameters

1. The number of trees B.
   - If B is too large, it could result in overfitting in boosting.
   - Cross-validation is used to select B.
2. λ, the shrinkage parameter
   - A small positive number – usual values are 0.01 or 0.001.
   - Very small λ can require using a large value of B to get good prediction performance
3. The number of splits *d*, which controls the complexity of the boosted ensemble.
   - *d* = 1 work well, in which case each tree is a **stump** with a single split. This boosted ensemble is fitting an additive model, since each term involves a single variable.
   - More generally, *d* is the interaction depth and controls the interaction order of the boosted model, since *d* splits involve at most *d* variables.

Quiz

1. in boosting, because the growth of a particular tree takes into account the other trees that have already been grown, smaller trees are typically sufficient as compared to random forest
   A. True
   B. False

Correct Answer: A

2. Smaller trees
   A. Increase overfitting
   B. Reduce overfitting

Correct Answer: B

GTx

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D
Professor
Scheller College of Business

Bagging, Random Forests, and Boosting Examples

Bagging and Random Forests on Boston Data Set

- We apply bagging and random forests to the Boston data set, using the **randomForest** package in R.
- The exact results may vary depending on your computer's installed versions of R and randomForest package.
- Remember that bagging is a special case of random forests with $m = p$
- We set up a training and test set for the Boston data set.

GTx

Boston Data Set

```
if (!require(MASS)) install.packages("MASS")
library(MASS)
if (!require(randomForest)) install.packages("randomForest")
library(randomForest)
set.seed (1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
boston.test=Boston[-train ,"medv"]
bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,im
portance=TRUE)
bag.boston
```

mtry=13 indicates that all 13 predictors should be considered for each split of the tree, which means that bagging should be done

GTx

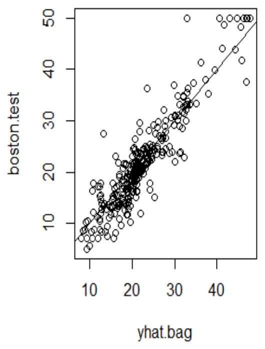
Bagging Using the randomForest Package

```
> bag.boston

Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13,
importance = TRUE,      subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 14.65748
                    % Var explained: 83.42
```

Bagging Performance on Test Data Set

```
yhat.bag =
predict(bag.boston,newdata=
Boston[-train,])
plot(yhat.bag, boston.test)
abline(0,1)

mean((yhat.bag-boston.test)^2)
[1] 13.08278
```

The test set MSE associated with the bagged regression tree is 13.1, almost half that obtained using an optimally-pruned single tree (investigate this on your own).



randomForest on Boston Dataset

```
set.seed(1)
rf.boston=randomForest(medv~.,data=Boston,subset=train, mtry=4)
yhat.rf = predict(rf.boston ,newdata=Boston[-train ,])
mean((yhat.rf-boston.test)^2)

## [1] 13.29981
```

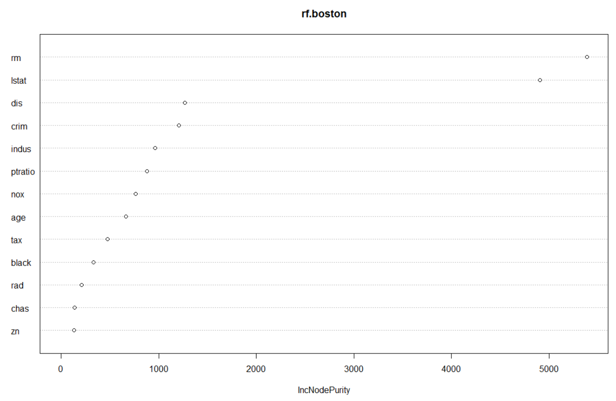
The test MSE using randomForest is 13.3
Compare this MSE to bagging whose MSE was 13.1

importance() function

```
> importance(rf.boston, type=2)
        IncNodePurity
crim        1206.0289
zn           131.2964
indus        961.9492
chas         134.4376
nox          760.9545
rm          5391.3982
age          663.6239
dis         1265.7396
rad          206.9177
tax          475.9189
ptratio      880.9906
black        328.7396
lstat       4909.8992
```

- The importance function shows the total decrease in node impurity that results from splits over that variable, averaged over all trees.
- In the case of regression trees, the node impurity is measured by the training SSE and for classification trees by the deviance.

varImpPlot(rf.boston)



Across all the trees considered in the random forest, the two most important variables are:

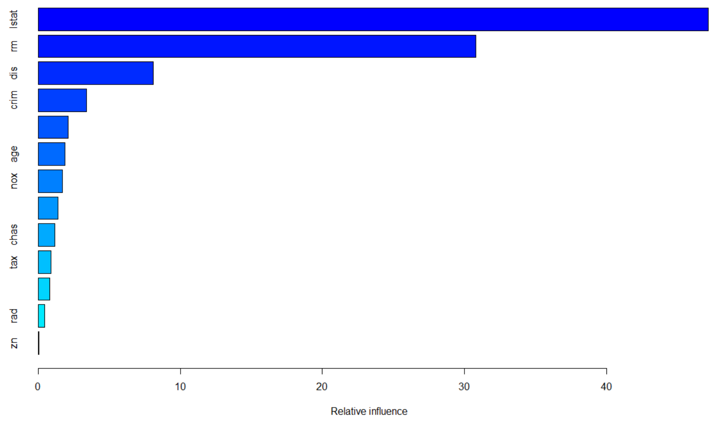
- **rm** – average number of rooms per dwelling.
- **lstat** - the wealth level of the community as measured by the lower status of the population (percent).

Boosting on Boston Data Set

- We use the **gbm** package and its gbm() function to fit boosted regression trees to the Boston data set
- Run gbm() with the option **distribution = "gaussian"** since this is a regression problem; if it were classification then the option would be **distribution = "bernoulli"**
- Set the option **n.trees=5000** to indicate that we want 5000 trees
- Set the option **interaction.depth=4** to limit the depth of each tree

```
set.seed (1)
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian",
          n.trees=5000, interaction.depth = 4)
summary(boost.boston)
```

```
> summary(boost.boston)
            var    rel.inf
lstat     lstat 47.1662116
rm           rm 30.8041745
dis         dis  8.1179563
crim       crim  3.4240776
ptratio ptratio  2.0979225
age         age  1.8913236
nox         nox  1.7068723
black     black  1.4067541
chas       chas  1.1739250
tax         tax  0.9062077
indus     indus  0.7930559
rad         rad  0.4543214
```

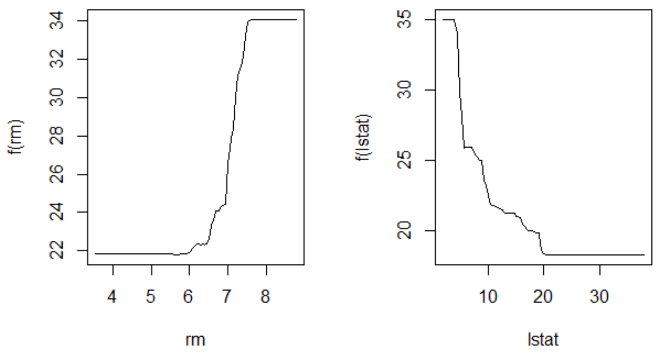


Partial Dependence Plots

- lstat and rm are the most important variables
- We can also produce partial dependence plots for these two variables
- These plots produce the marginal effect of the selected variables on the response after integrating out the other variables
- As can be expected, median house prices are increasing with rm and decreasing with lstat

Partial Dependence Plots

```
# partial dependence plots

par(mfrow=c(1,2))
plot(boost.boston, i="rm")
plot(boost.boston, i="lstat")
```



- As can be expected, median house prices are increasing with rm and decreasing with lstat

Test MSE

```
# Now use the boosted model to predict medv on the test set
> boston.test=Boston[-train ,"medv"]
> mean((yhat.boost -boston.test)^2)
[1] 14.65
```

- The test MSE obtained is 14.65.

Shrinkage Parameter λ Set to 0.02 (Default is .001)

```
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian"
, n.trees=5000, interaction.depth = 4, shrinkage=0.02, verbose=F)
yhat.boost=predict(boost.boston,newdata=Boston[-train,],
n.trees=5000)
mean((yhat.boost -boston.test)^2)

[1] 17.23
```

- With λ = 0.02, the test MSE is 17.23

Recap of Lessons in this Module

A. Introduction to Tree Based Methods
B. Regression Trees
C. Regression Tree Examples
D. Classification Trees
E. Carseats Example for Classification Trees
F. Bagging
G. Random Forests
H. Boosting
I. Bagging, Random Forests, and Boosting Examples

* The materials in this module are from Chapter 8 of the ISLR textbook

Data Analytics in Business

Classification and Regression Trees (CART)

Sridhar Narasimhan, Ph.D

Professor

Scheller College of Business

End