Exploratory Data Mining via Search Strategies Lab #3

Ross Jacobucci & Kevin J. Grimm

Outline

This script will go over Decision Trees and generalizations in R.
Resources:
Basic intro to Decision Trees: `http://www.statmethods.net/advstats/cart.html`
Full list of data mining packages in R:
`http://cran.r-project.org/web/views/MachineLearning.html`
Two packages will be used and their caret equivalents:

- **rpart** (tree accomplishes very similar thing):`http://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf`

- **party**:
  `http://cran.r-project.org/web/packages/party/vignettes/party.pdf`

  In **caret**, method =
- "rpart" – tuning = cp (complexity parameter)
- "rpart2" – tuning = maxdepth
- "rpartCost" – tuning = cp and cost
- "ctree" – tuning = mincriterion (p value thresholds)
- "ctree2" – tuning = maxdepth

Lets load the main packages

```
library(caret)
library(rpart)
library(pROC)
library(randomForest)
library(ada)
library(ISLR)
library(party)
library(MASS) # for boston data
data(Boston)
```

Regression (continous outcome)

Use rpart first with the Boston data use regression first – predicting median value of homes

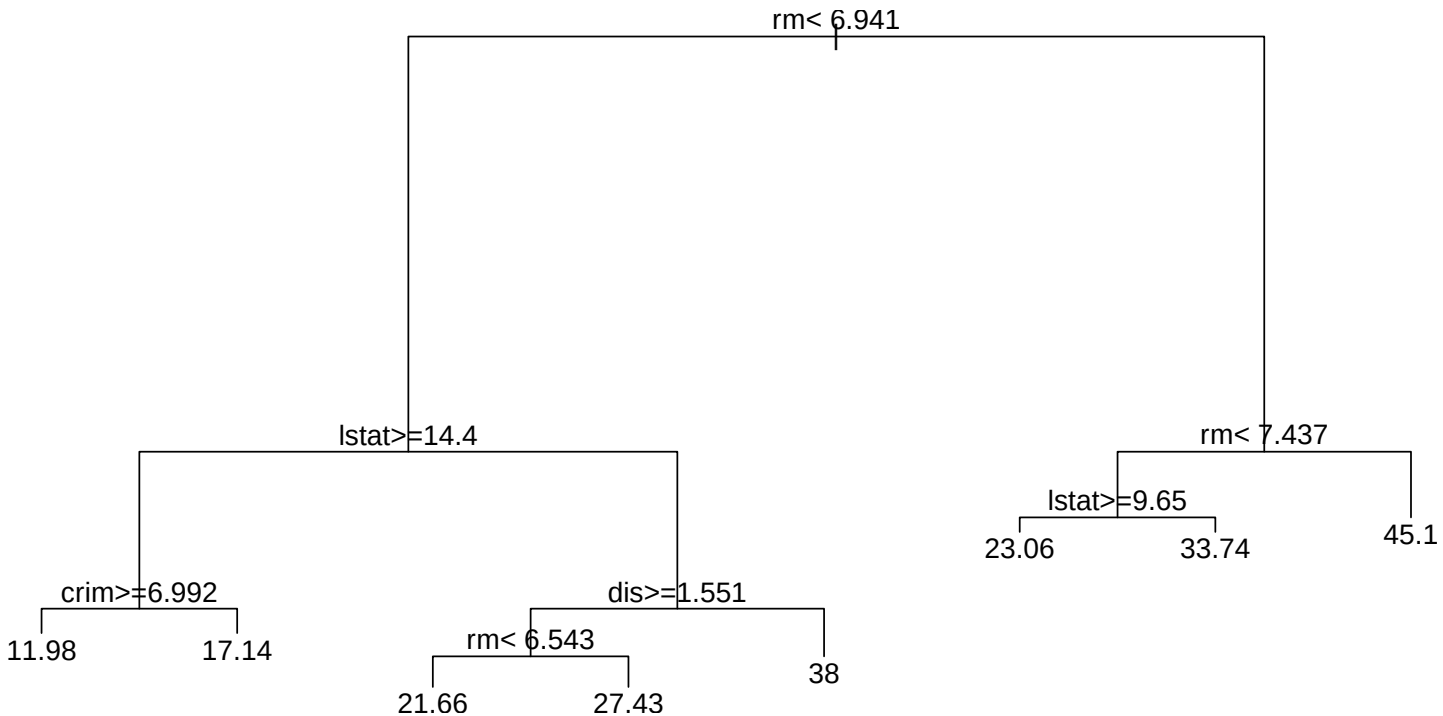
```
#str(Boston)

# lets get a baseline with linear regression
lm.Boston <- lm(medv ~., data=Boston)
#summary(lm.Boston)
```

We do pretty well with linear regression R-squared of .74

CART

How about if we just blindly apply Decision Trees

```
rpart.Boston <- rpart(medv ~., data=Boston)
#summary(rpart.Boston)
plot(rpart.Boston);text(rpart.Boston)
```



CART Continued

```
pred1 <- predict(rpart.Boston)
cor(pred1,Boston$medv)**2
```

```
## [1] 0.8075721
```

Doing really well – Rsquared = 0.81

```
# this can be hard to interpret, so I like to look at a different output
rpart.Boston
```

```
## n= 506
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 506 42716.3000 22.53281
##    2) rm< 6.941 430 17317.3200 19.93372
##      4) lstat>=14.4 175  3373.2510 14.95600
##        8) crim>=6.99237 74  1085.9050 11.97838 *
##        9) crim< 6.99237 101  1150.5370 17.13762 *
##      5) lstat< 14.4 255  6632.2170 23.34980
##       10) dis>=1.5511 248  3658.3930 22.93629
##         20) rm< 6.543 193  1589.8140 21.65648 *
##         21) rm>=6.543 55  643.1691 27.42727 *
##       11) dis< 1.5511 7  1429.0200 38.00000 *
##    3) rm>=6.941 76  6059.4190 37.23816
##      6) rm< 7.437 46  1899.6120 32.11304
```

Lasso Regression

What if we tried regularized (penalized) regression instead?
Note: for glmnet, both the x's and y have to be in separate matrices
– and all class = numeric
– don't worry about response, doesn't have to be factor for logistic
—— just specify "binomial"

```
y.B <- Boston$medv
x.B <- sapply(Boston[,-14],as.numeric)

# alpha =1 for lasso, 0 for ridge
library(glmnet)
cv <- cv.glmnet(x.B,y.B,alpha=1)
lasso.reg <- glmnet(x.B,y.B,alpha=1,family="gaussian",lambda=cv$lambda.min)

lasso.resp <- predict(lasso.reg,newx=x.B)
cor(y.B,lasso.resp)**2
```

```
##                s0
## [1,] 0.7403768
```

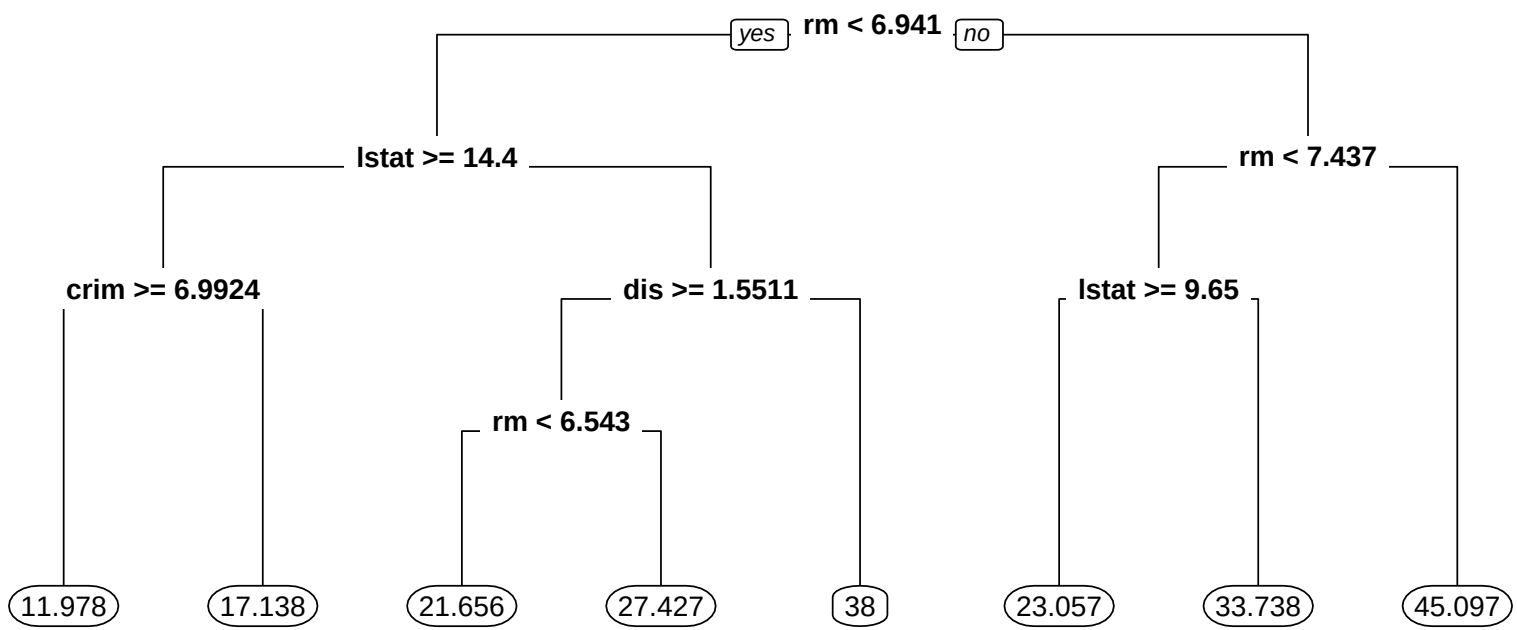
Taking into account cross-validation, we do worse compared to linear regression with no tuning.
CART Plots So the plot for rpart didn'tcome out that well.
Good news, there are better options for plotting.

http://blog.revolutionanalytics.com/2013/06/

CART Plotting Continued

Note, prp() offers many additional capabilities for tweaking the plot For instance:

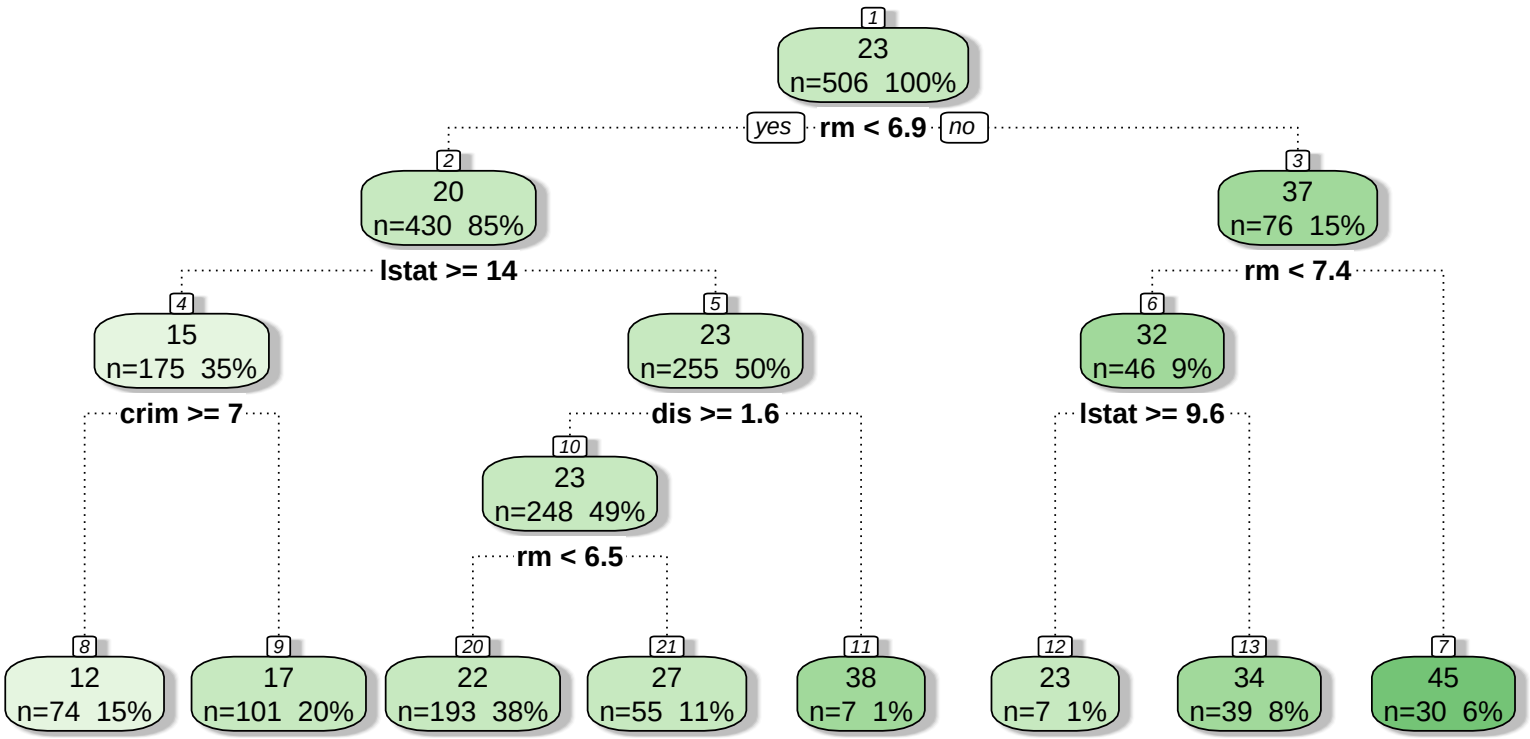
```
# ?prp
prp(rpart.Boston,varlen=10,digits=5,fallen.leaves=T)
```



CART Plotting Continued

Probably my favorite, fancyRpartPlot()

```
#fancyRpartPlot(); from rattle
fancyRpartPlot(rpart.Boston)
```



Conditional Inference Trees

So what about with conditional inference trees?
What if we want a smaller tree? This can be accomplished a number of ways. We can prespecify the maxdepth, the minimum number of people per node, as well as making more restrictive splitting criterion.
Example of prespecifying the depth with ctree()

```
ctree.Boston <- ctree(medv ~., data=Boston)
#plot(ctree.Boston) # too big of a tree
pred2 <- predict(ctree.Boston)
cor(pred2,Boston$medv)**2
```

```
## [1] 0.8746338
```

We do better than rpart, Rsquared $= 0.87$

Conditional Inference Trees Continued

Biggest difference between ctree() and rpart() is that ctree() does not demonstrate bias with respect to the number of response options, and supposedly had less of a propensity to overfit than rpart().
Note: the models are not optimizing based on Rsquared, most likely MSE

So what do we think now? Are we happy with results? Remember, decision trees are generally quite robust, so it may not be necessary to check assumptions. – See Table 10.1 ESL

But what about generalizability?

Although not as serious as with SVM for instance, Decision Trees have a propensity to overfit, meaning the tree structure won't generalize well

So let's try just creating a simple Training and Test datasets

```
train = sample(dim(Boston)[1], dim(Boston)[1]/2) # half of sample
Boston.train = Boston[train, ]
Boston.test = Boston[-train, ]
```

Linear Regression with CV

```
lm.train <- lm(medv ~., data=Boston.train)

pred.lmTest <- predict(lm.train,Boston.test)
cor(pred.lmTest,Boston.test$medv)**2
```

```
## [1] 0.7021417
```

Note: we are taking our lm object trained on the train dataset, and using these fixed coefficients to predict values on the test dataset.

In SEM, this is referred to as a tight replication strategy No difference in using a test dataset – both Rsq are 0.74

How about with rpart?

CART CV

```
rpart.train <- rpart(medv ~., data=Boston.train)

pred.rpartTest <- predict(rpart.train,Boston.test)
cor(pred.rpartTest,Boston.test$medv)**2
```

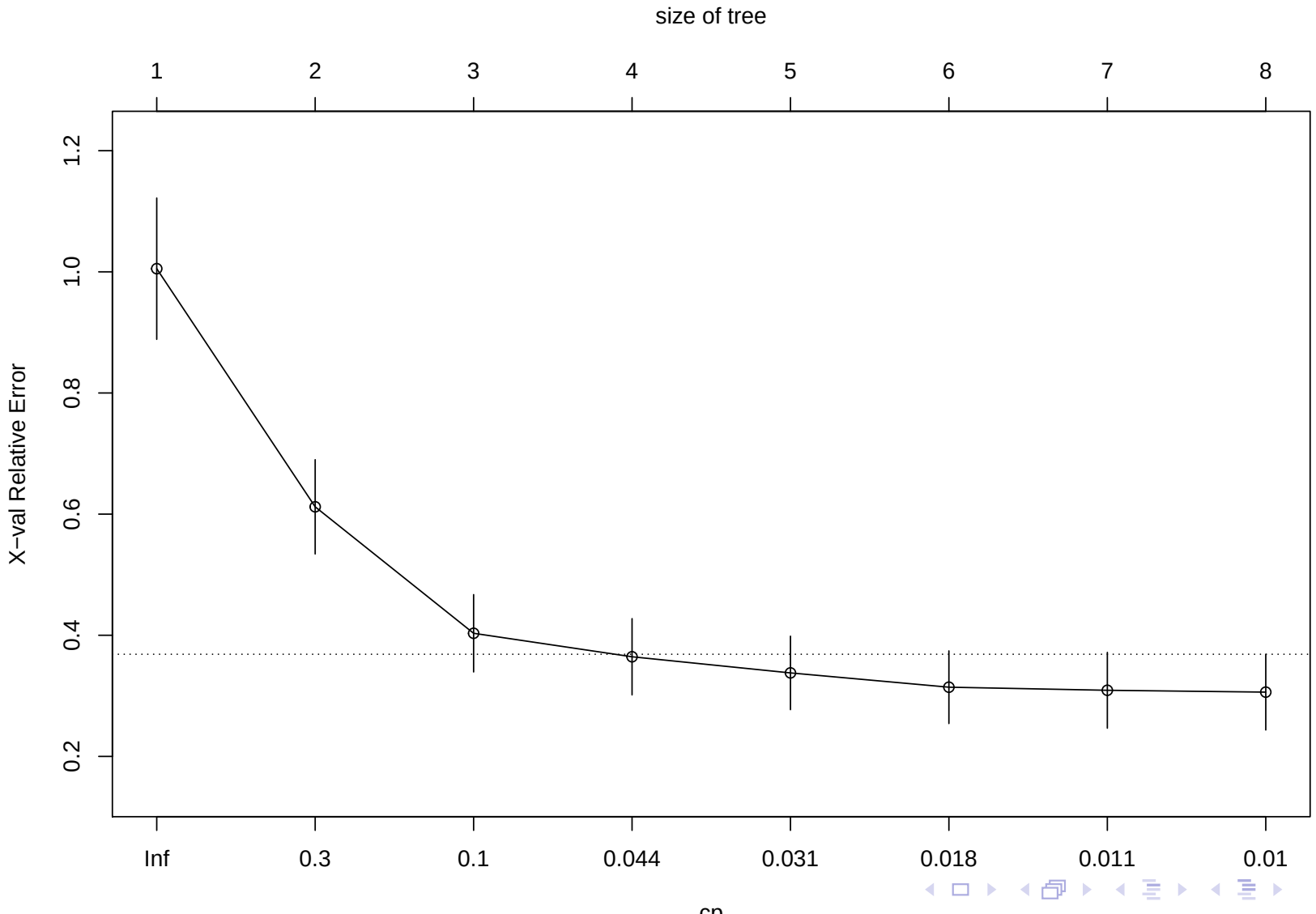
```
## [1] 0.7428159
```

Not as good – drops from 0.81 to 0.76 – still better than lm()
But with rpart, it is common to prune trees back. What if we try this, is there less of a drop in R^2?
Note: rpart automatically does internal CV, varying the complexity paramter (cp). If you use the tree package instead, you will have to use cv.tree()

CART Pruning

With plotcp() we are going to choose the error within 1 SE of the lowest cross-validated error. This will be used to prune

```
plotcp(rpart.train)
```



CART Pruning Continued

```
printcp(rpart.train)
```

```
##
## Regression tree:
## rpart(formula = medv ~ ., data = Boston.train)
##
## Variables actually used in tree construction:
## [1] crim  dis   lstat rm
##
## Root node error: 19695/253 = 77.847
##
## n= 253
##
##         CP nsplit rel error  xerror     xstd
## 1 0.474452      0   1.00000 1.00523 0.116686
## 2 0.192344      1   0.52555 0.61206 0.077845
## 3 0.055118      2   0.33320 0.40329 0.063869
## 4 0.034885      3   0.27809 0.36449 0.062982
## 5 0.027810      4   0.24320 0.33785 0.060652
## 6 0.011031      5   0.21539 0.31427 0.059976
## 7 0.010121      6   0.20436 0.30917 0.062573
## 8 0.010000      7   0.19424 0.30613 0.062535
```

What is xerror and other error?

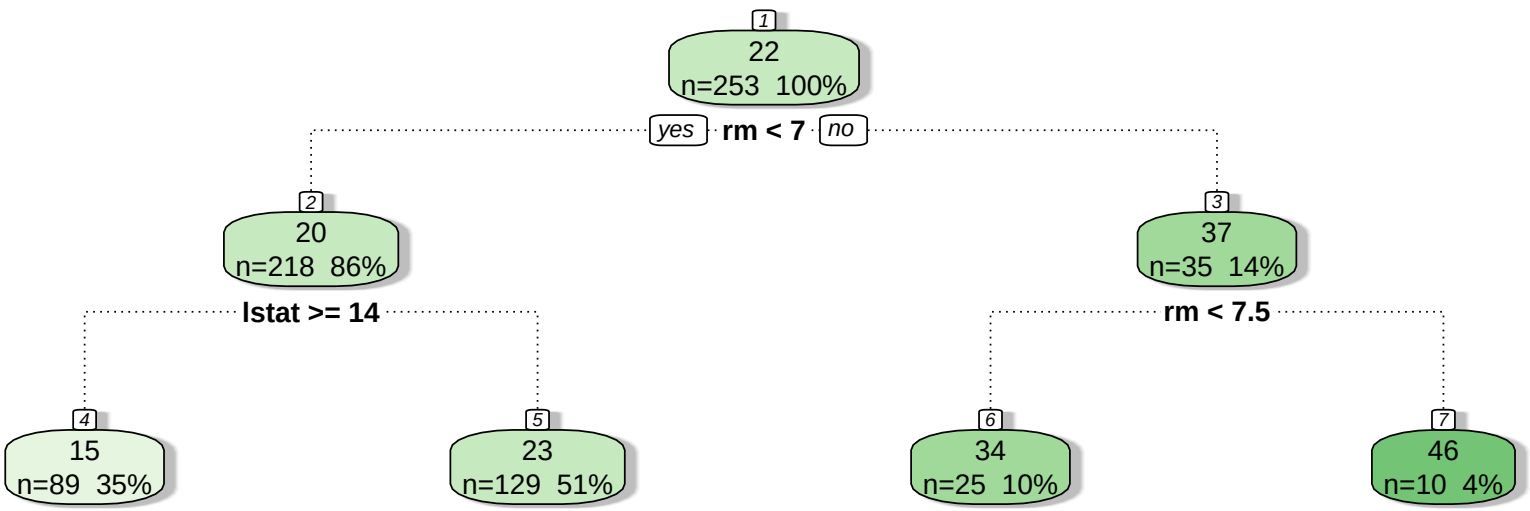
CART Pruning Continued

```
# rsq.rpart(rpart.train) ## another way to get cp plots
prune.Bos <- prune(rpart.train,0.053)

pred.prune <- predict(prune.Bos,Boston.test)
cor(pred.prune,Boston.test$medv)
```

```
## [1] 0.8027312
```

Plot Pruned Tree

```
#plot(prune.Bos);text(prune.Bos)
fancyRpartPlot(prune.Bos)
```



CTree CV

```
ctree.train <- ctree(medv ~., data=Boston.train)
#plot(ctree.train) # huge tree
pred.ctreeTest <- predict(ctree.train,Boston.test)
cor(pred.ctreeTest,Boston.test$medv)**2
```

```
## [1] 0.7525739
```

It is worth noting how much more of an effect there was for using a test dataset with the tree methods as compared to lm(), this is pretty typical, and much more important with more "flexible" methods such as random forests, gbm, svm etc. . .

Classification

Classification

Two Biggest Things To Remember:
1. Make sure functions outcome variable is categorical; as.factor(outcome)
2. Using predict() changes. Variable across packages

As a baseline, we will use logistic regression.

```
library(ISLR)
data(Default)
#head(Default)
str(Default)
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...
##  $ balance: num  730 817 1074 529 786 ...
##  $ income : num  44362 12106 31767 35704 38463 ...
```

My favorite function in R is str(), as it gives the class of each variable and other summary characteristics. Most important thing to note is that the "default" variable is already coded as a factor variable, meaning that R now knows it is categorical, and will change the cost function (thus estimator) accordingly.

This is really important because rpart,randomForest and other packages do not automatically detect whether it is a regression or classification problem. If you don't change the outcome variable to its proper class, you could get a suboptimal answer (use

ROC Curves

good intro to using ROC:
https://ccrma.stanford.edu/workshops/mir2009/references/ROCintro.pdf
These plots are a balanace of sensitivity and specificity. Ideally the curve gets as close as possible to the upper left corner.

To get this plot, we need to get our predictions from our logistic model.

```
glm.probs=predict(lr.out,type="response")
#glm.pred00=ifelse(glm.probs>0.5,1,0)

rocCurve <- roc(Default$default,glm.probs)
pROC::auc(rocCurve)
```

```
## Area under the curve: 0.9496
```

```
pROC::ci(rocCurve)
```

```
## 95% CI: 0.9402-0.959 (DeLong)
```

```
# quartz()
plot(rocCurve, legacy.axes = TRUE,print.thres=T,print.auc=T)
```

Lasso Logistic Regression

How about lasso logistic regression?

```
library(glmnet)
yy = as.numeric(Default$default)
xx = sapply(Default[,2:4],as.numeric)
lasso.out <- cv.glmnet(xx,yy,family="binomial",alpha=1,nfolds=10) #alpha=1 ==
# find best lambda
ll <- lasso.out$lambda.min

lasso.probs <- predict(lasso.out,newx=xx,s=ll,type="response")
```

Results from lasso using CV

```
rocCurve.lasso <- roc(Default$default,lasso.probs)
```

```
## Warning in roc.default(Default$default, lasso.probs): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a
## numeric vector.
```

```
pROC::auc(rocCurve.lasso)
```

```
## Area under the curve: 0.9495
```

```
pROC::ci(rocCurve.lasso)
```

```
## 95% CI: 0.9401-0.959 (DeLong)
```

```
# quartz()
```