Statistics 452: Statistical Learning and Prediction

Chapter 8, Part 4: Regression Trees Lab

Brad McNeney

Boston Data

- Recall the `Boston` dataset in which the response is the median house price in $1000 and and there are 13 predictors.
- I've replaced the variable `black` by `predAA`, an indicator that takes value 1 if the town is predominantly African American and 0 otherwise.

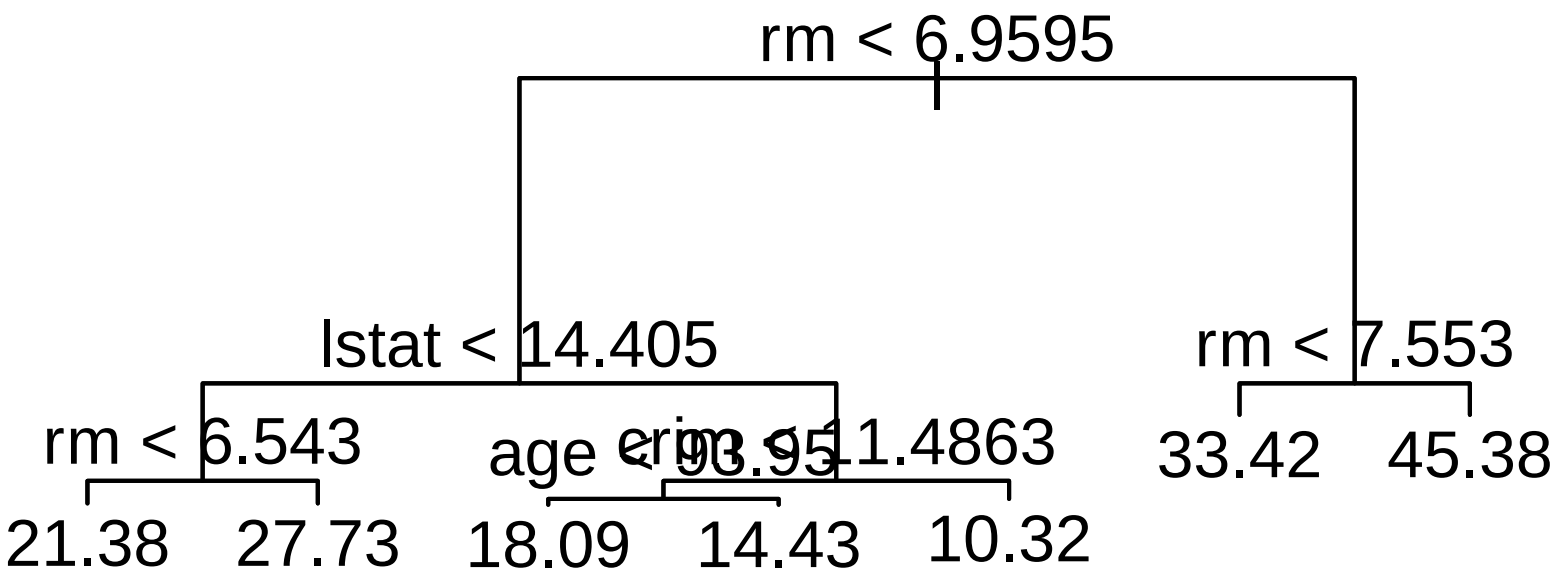
Training and Test Data

- Split the data in half for training and testing.

```
set.seed(1)
train <- sample(1:nrow(Boston),nrow(Boston)/2)
```

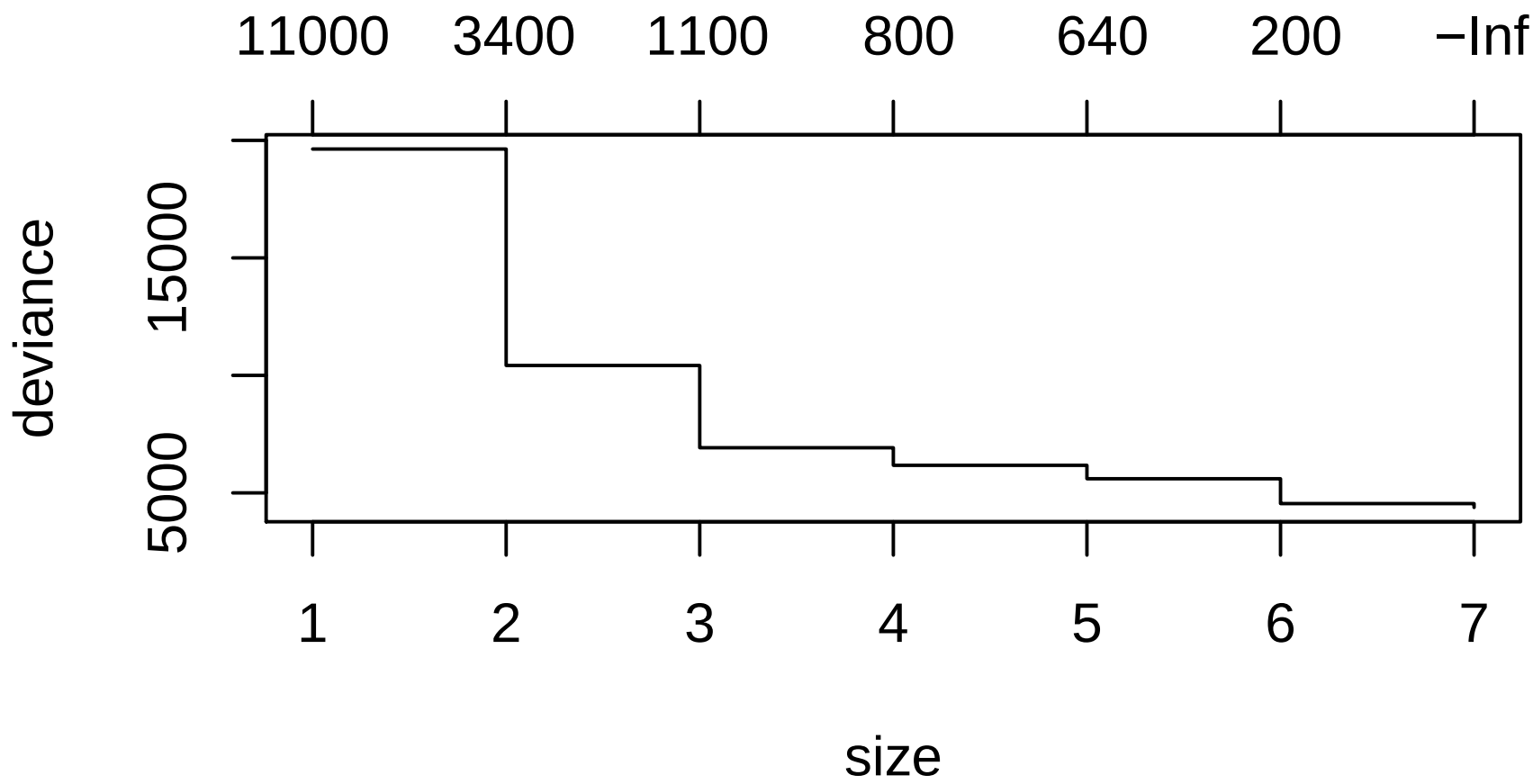
Regression Tree

```
library(tree)
tt <- tree(medv ~ ., data=Boston, subset=train)
plot(tt) # rm, lsat, crim and age used
text(tt)
```



Cross-Validation to Prune Tree

```
cvt <- cv.tree(tt)
plot(cvt) # No pruning required -- could use size 5 for parsimony
```



Test Set Error

- Use the unpruned tree

```
yhat <- predict(tt,newdata=Boston[-train,])
y <- Boston[-train,"medv"]
mean((y-yhat)^2)
```

```
## [1] 35.28688
```

Bagging

```
library(randomForest)
set.seed(1) # for bootstrapping
btt <- randomForest(medv ~ ., data=Boston,subset=train,
                    mtry=13) # mtry=p for bagging
yhat <- predict(btt,newdata=Boston[-train,])
mean((y-yhat)^2)
```

```
## [1] 23.34321
```

Random Forest

```
set.seed(1) # for bootstrapping
rtt <- randomForest(medv ~ ., data=Boston,subset=train,
                    mtry=sqrt(13),importance=TRUE)
yhat <- predict(rtt,newdata=Boston[-train,])
mean((y-yhat)^2)
```

```
## [1] 18.56577
```

```
importance(rtt)
```

```
##           %IncMSE IncNodePurity
## crim    16.224623    1208.50019
## zn       3.477727     187.46046
## indus    6.761188     742.21431
## chas     2.350918     103.29396
## nox     13.214861    1069.82251
## rm      28.589149    6477.77842
## age     12.211473     761.16192
## dis      9.809813     944.63009
## rad      4.004756     149.61208
## tax      9.687284     678.62951
## ptratio 10.277997    1319.36163
## lstat   25.566944    5435.75676
## predAA   8.224915      60.43943
```

Boosting

```
library(gbm)
bott <- gbm(medv ~ ., data=Boston[train,],
                      distribution="gaussian",n.trees=1000) #M
yhat <- predict(bott,newdata=Boston[-train,],n.trees=1000)
mean((y-yhat)^2)
```

```
## [1] 18.95041
```

Boosting with Greater Interaction Depth

```
bott <- gbm(medv ~ ., data=Boston[train,], interaction.depth=4,
                    distribution="gaussian",n.trees=1000)
yhat <- predict(bott,newdata=Boston[-train,],n.trees=1000)
mean((y-yhat)^2)
```

```
## [1] 16.97259
```

Boosting with Smaller λ

```
bott.cv <- gbm(medv ~ ., data=Boston[train,],
               shrinkage=0.001, # decrease from default 0.1
               distribution="gaussian",n.trees=100000,
               cv.folds=5)
gbm.perf(bott.cv,method="cv") # 97,736 trees !
yhat <- predict(bott,newdata=Boston[-train,],n.trees=100000)
mean((y-yhat)^2) # 85.1 -- overfit

bott <- gbm(medv ~ ., data=Boston[train,],
               shrinkage=0.001,
               distribution="gaussian",n.trees=10000)
yhat <- predict(bott,newdata=Boston[-train,],n.trees=10000)
mean((y-yhat)^2) # 22.2 -- still not great
```

Using the Test Set for Tuning

- Note: If we use boosting with all of `gbm()`'s "factory settings", we beat the other approaches.
- However, if we plan to tune the boosting algorithm (shrinkage, interaction depth) we can't use the test set in this way.
  - We are essentially fitting the test data.
- If we require a test set for tuning, it should not be the one we use to evaluate the tuned algorithm.
  - We should split the data into three parts: (i) training (can be the largest part), (ii) tuning test set, and (iii) test set.