Chapter 8 Exercises

Adeline Casali

2023-11-30

# Question 8

Load packages

library(ISLR2)  
library(tree)  
library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

library(BART)

## Loading required package: nlme

## Loading required package: nnet

## Loading required package: survival

library(gbm)

## Loaded gbm 2.1.8.1

library(pls)

##   
## Attaching package: 'pls'

## The following object is masked from 'package:stats':  
##   
## loadings

1. I split the dataset into separate test and training sets

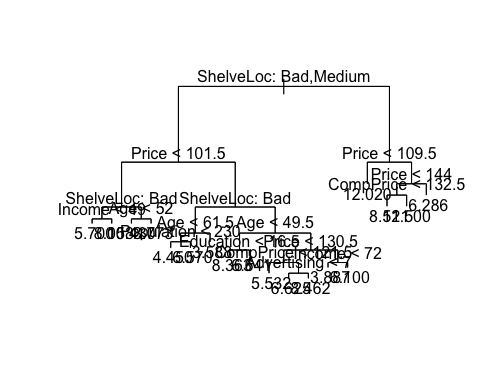
set.seed(123)  
carseats\_train <- sample(1:nrow(Carseats), nrow(Carseats) / 2)  
carseats\_test <- Carseats[-carseats\_train, "Sales"]

1. The MSE for the regression tree model is 4.395.

tree\_carseats <- tree(Sales ~ ., Carseats, subset = carseats\_train)  
summary(tree\_carseats)

##   
## Regression tree:  
## tree(formula = Sales ~ ., data = Carseats, subset = carseats\_train)  
## Variables actually used in tree construction:  
## [1] "ShelveLoc" "Price" "Income" "Age" "Population"   
## [6] "Education" "CompPrice" "Advertising"  
## Number of terminal nodes: 18   
## Residual mean deviance: 2.132 = 388.1 / 182   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -4.08000 -0.92870 0.06244 0.00000 0.87020 3.71700

plot(tree\_carseats)  
text(tree\_carseats, pretty = 0)

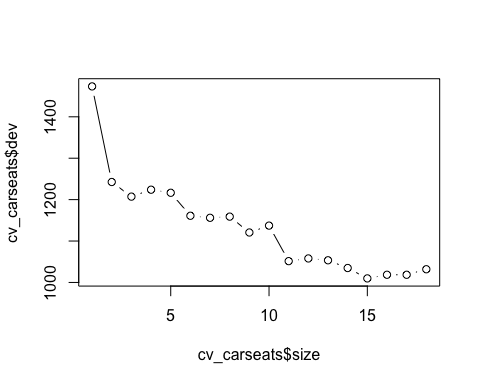


yhat\_carseats <- predict(tree\_carseats, newdata = Carseats[-carseats\_train, ])  
cat("MSE:", mean((yhat\_carseats - carseats\_test)^2), "\n")

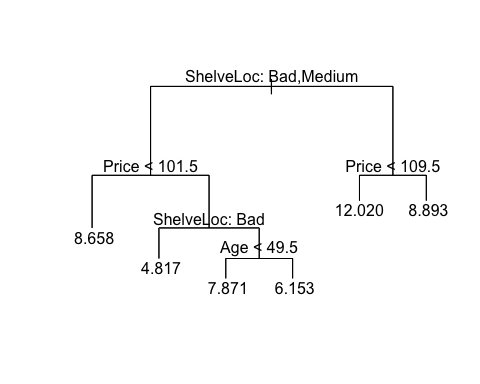
## MSE: 4.395357

1. The new MSE is 4.669, which is greater than the MSE before pruning.

cv\_carseats <- cv.tree(tree\_carseats)  
plot(cv\_carseats$size, cv\_carseats$dev, type = "b")



# The best value is 6  
prune\_carseats <- prune.tree(tree\_carseats, best = 6)  
plot(prune\_carseats)  
text(prune\_carseats, pretty = 0)



yhat\_cv\_carseats <- predict(prune\_carseats, newdata = Carseats[-carseats\_train, ])  
cat("MSE:", mean((yhat\_cv\_carseats - carseats\_test)^2), "\n")

## MSE: 4.668759

1. The MSE for the model utilizing bagging is much lower at 2.761. The most important variables are shelf location and price.

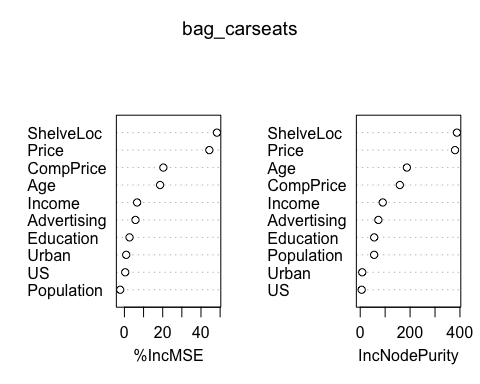
set.seed(123)  
bag\_carseats <- randomForest(Sales ~ ., data = Carseats, subset = carseats\_train, mtry = 10, importance = TRUE)  
bag\_carseats

##   
## Call:  
## randomForest(formula = Sales ~ ., data = Carseats, mtry = 10, importance = TRUE, subset = carseats\_train)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 10  
##   
## Mean of squared residuals: 2.912487  
## % Var explained: 59.53

yhat\_bag\_carseats <- predict(bag\_carseats, newdata = Carseats[-carseats\_train, ])  
importance(bag\_carseats)

## %IncMSE IncNodePurity  
## CompPrice 20.3414969 158.911610  
## Income 6.6237140 90.369331  
## Advertising 5.7777253 72.793558  
## Population -2.2001506 55.786278  
## Price 44.3578602 380.255094  
## ShelveLoc 48.3345635 387.886972  
## Age 18.6296851 187.107660  
## Education 2.6619834 55.987493  
## Urban 0.9276070 8.152320  
## US 0.4202302 5.900097

varImpPlot(bag\_carseats)



cat("MSE:", mean((yhat\_bag\_carseats - carseats\_test)^2), "\n")

## MSE: 2.76144

1. With the random forests method, the MSE is slightly higher than with bagging at 3.017. This is still lower than the original MSE. The most important variables are still price and shelf location, although now price is considered slightly more important. I determined the optimal mtry (m) value by experimenting with different values. Typically, m is roughly equivalent to the sqrt of p (10), which in this case would make m = 3. However, a slightly higher value of 5 produced a lower MSE.

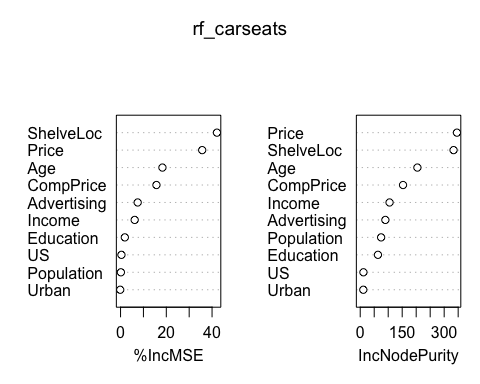
set.seed(123)  
rf\_carseats <- randomForest(Sales ~ ., data = Carseats, subset = carseats\_train, mtry = 5, importance = TRUE)  
yhat\_rf\_carseats <- predict(rf\_carseats, newdata = Carseats[-carseats\_train, ])  
cat("MSE:", mean((yhat\_rf\_carseats - carseats\_test)^2), "\n")

## MSE: 3.01707

importance(rf\_carseats)

## %IncMSE IncNodePurity  
## CompPrice 15.6874802 152.97700  
## Income 6.1782093 104.83134  
## Advertising 7.4822778 89.84070  
## Population 0.1519680 74.70762  
## Price 35.6880806 345.59391  
## ShelveLoc 42.1336259 333.55620  
## Age 18.3059283 204.29930  
## Education 1.8670124 63.31604  
## Urban -0.1467837 11.01253  
## US 0.3849865 11.47038

varImpPlot(rf\_carseats)



f) The MSE for the BART method is significantly lower than with any of the other methods at 0.205.

x\_carseats <- Carseats[, 1:11]  
y\_carseats <- Carseats[, "Sales"]  
xtrain\_carseats <- x\_carseats[carseats\_train, ]  
ytrain\_carseats <- y\_carseats[carseats\_train]  
xtest\_carseats <- x\_carseats[-carseats\_train, ]  
ytest\_carseats <- y\_carseats[-carseats\_train]  
set.seed(123)  
bart\_carseats <- gbart(xtrain\_carseats, ytrain\_carseats, x.test = xtest\_carseats)

## \*\*\*\*\*Calling gbart: type=1  
## \*\*\*\*\*Data:  
## data:n,p,np: 200, 15, 200  
## y1,yn: 3.230000, 3.070000  
## x1,x[n\*p]: 10.660000, 1.000000  
## xp1,xp[np\*p]: 9.500000, 1.000000  
## \*\*\*\*\*Number of Trees: 200  
## \*\*\*\*\*Number of Cut Points: 100 ... 1  
## \*\*\*\*\*burn,nd,thin: 100,1000,1  
## \*\*\*\*\*Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.260569,3,2.87617e-30,7.43  
## \*\*\*\*\*sigma: 0.000000  
## \*\*\*\*\*w (weights): 1.000000 ... 1.000000  
## \*\*\*\*\*Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,15,0  
## \*\*\*\*\*printevery: 100  
##   
## MCMC  
## done 0 (out of 1100)  
## done 100 (out of 1100)  
## done 200 (out of 1100)  
## done 300 (out of 1100)  
## done 400 (out of 1100)  
## done 500 (out of 1100)  
## done 600 (out of 1100)  
## done 700 (out of 1100)  
## done 800 (out of 1100)  
## done 900 (out of 1100)  
## done 1000 (out of 1100)  
## time: 2s  
## trcnt,tecnt: 1000,1000

yhat\_bart\_carseats <- bart\_carseats$yhat.test.mean  
cat("MSE:", mean((ytest\_carseats - yhat\_bart\_carseats)^2), "\n")

## MSE: 0.2050057

# Question 9

1. I split the data into a training set containing 800 observations and a test set containing the remainder.

set.seed(123)  
oj\_train <- sample(1:nrow(OJ), 800)  
oj\_test <- OJ[-oj\_train, "Purchase"]

1. In this model, the training error rate is 16.5% and the tree has 8 terminal nodes.

tree\_oj <- tree(Purchase ~ ., OJ, subset = oj\_train)  
summary(tree\_oj)

##   
## Classification tree:  
## tree(formula = Purchase ~ ., data = OJ, subset = oj\_train)  
## Variables actually used in tree construction:  
## [1] "LoyalCH" "PriceDiff"  
## Number of terminal nodes: 8   
## Residual mean deviance: 0.7625 = 603.9 / 792   
## Misclassification error rate: 0.165 = 132 / 800

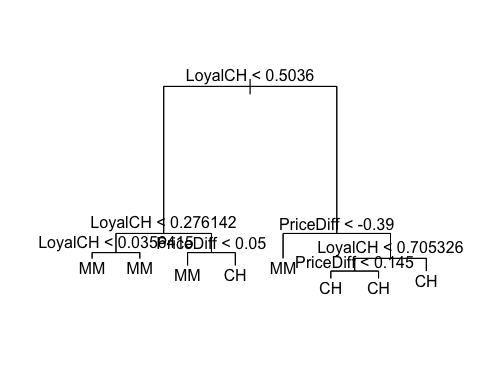
1. One of the terminal nodes, number 6, shows that the split criteria is a price difference of < -0.39 and there are 27 observations in that branch. The deviance is 32.82 and the overall prediction for that branch is MM (minute maid orange juice). Furthermore, 29.63% of the observations in the branch are CH and 70.37% are MM.

tree\_oj

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 800 1071.00 CH ( 0.60875 0.39125 )   
## 2) LoyalCH < 0.5036 350 415.10 MM ( 0.28000 0.72000 )   
## 4) LoyalCH < 0.276142 170 131.00 MM ( 0.12941 0.87059 )   
## 8) LoyalCH < 0.0356415 56 10.03 MM ( 0.01786 0.98214 ) \*  
## 9) LoyalCH > 0.0356415 114 108.90 MM ( 0.18421 0.81579 ) \*  
## 5) LoyalCH > 0.276142 180 245.20 MM ( 0.42222 0.57778 )   
## 10) PriceDiff < 0.05 74 74.61 MM ( 0.20270 0.79730 ) \*  
## 11) PriceDiff > 0.05 106 144.50 CH ( 0.57547 0.42453 ) \*  
## 3) LoyalCH > 0.5036 450 357.10 CH ( 0.86444 0.13556 )   
## 6) PriceDiff < -0.39 27 32.82 MM ( 0.29630 0.70370 ) \*  
## 7) PriceDiff > -0.39 423 273.70 CH ( 0.90071 0.09929 )   
## 14) LoyalCH < 0.705326 130 135.50 CH ( 0.78462 0.21538 )   
## 28) PriceDiff < 0.145 43 58.47 CH ( 0.58140 0.41860 ) \*  
## 29) PriceDiff > 0.145 87 62.07 CH ( 0.88506 0.11494 ) \*  
## 15) LoyalCH > 0.705326 293 112.50 CH ( 0.95222 0.04778 ) \*

1. The most important indicator seems to be loyalty, as the first branch differentiates between loyalty to CH of < 0.5036 or greater than. You can see all split criteria and the 8 terminal nodes.

plot(tree\_oj)  
text(tree\_oj, pretty = 0)



e) The test accuracy rate is 81.48% and the test error rate is 18.52%.

set.seed(123)  
oj\_train\_index <- sample(1:nrow(OJ), 800)  
oj\_train <- OJ[oj\_train\_index, ]  
oj\_test <- OJ[-oj\_train\_index, ]  
tree\_pred\_oj <- predict(tree\_oj, newdata = oj\_test, type = "class")  
table(tree\_pred\_oj, oj\_test$Purchase)

##   
## tree\_pred\_oj CH MM  
## CH 150 34  
## MM 16 70

oj\_test\_acc <- ((150 + 70) / 270) \* 100  
cat("Test accuracy rate:", oj\_test\_acc, "%", "\n")

## Test accuracy rate: 81.48148 %

oj\_test\_err <- 100 - oj\_test\_acc  
cat("Test error rate:", oj\_test\_err, "%", "\n")

## Test error rate: 18.51852 %

1. Here is a cv tree.

set.seed(123)  
cv\_oj <- cv.tree(tree\_oj, FUN = prune.misclass)  
names(cv\_oj)

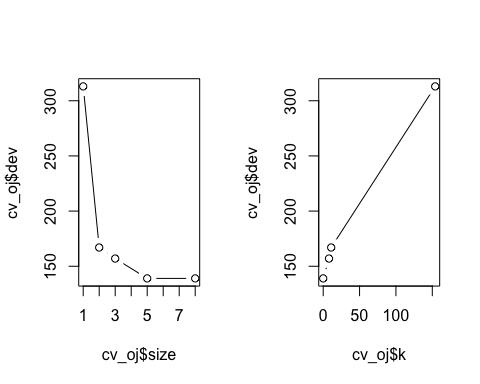
## [1] "size" "dev" "k" "method"

cv\_oj

## $size  
## [1] 8 5 3 2 1  
##   
## $dev  
## [1] 139 139 157 167 313  
##   
## $k  
## [1] -Inf 0 8 11 154  
##   
## $method  
## [1] "misclass"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

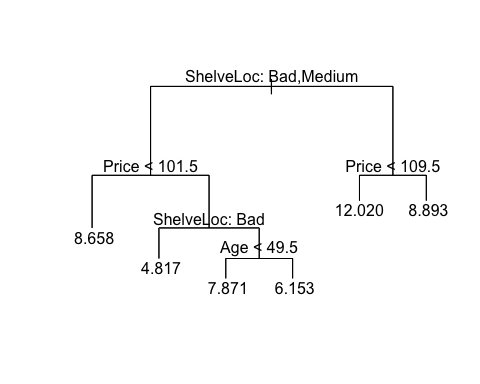
1. Here is a plot - the best tree size is 5.

par(mfrow = c(1,2))  
plot(cv\_oj$size, cv\_oj$dev, type = "b")  
plot(cv\_oj$k, cv\_oj$dev, type = "b")



h) A tree size of 5 seems to perform the best. It looks like a tree size of 8 would also perform well, but not by much more than 5 so I would rather keep the model more simple and understandable. i) Here is a pruned tree with the optimal tree size of 5.

prune\_oj <- prune.misclass(tree\_oj, best = 5)  
plot(prune\_carseats)  
text(prune\_carseats, pretty = 0)



j) The misclassification error (training set error) of the pruned model is exactly the same as with the original model at 16.5%.

summary(prune\_oj)

##   
## Classification tree:  
## snip.tree(tree = tree\_oj, nodes = c(4L, 7L))  
## Variables actually used in tree construction:  
## [1] "LoyalCH" "PriceDiff"  
## Number of terminal nodes: 5   
## Residual mean deviance: 0.826 = 656.6 / 795   
## Misclassification error rate: 0.165 = 132 / 800

1. The test error rates are also exactly the same at 18.52%.

tree\_pred\_oj\_cv <- predict(prune\_oj, oj\_test, type = "class")  
table(tree\_pred\_oj\_cv, oj\_test$Purchase)

##   
## tree\_pred\_oj\_cv CH MM  
## CH 150 34  
## MM 16 70

oj\_cv\_test\_acc <- ((150 + 70) / 270) \* 100  
cat("Test accuracy rate:", oj\_cv\_test\_acc, "%", "\n")

## Test accuracy rate: 81.48148 %

oj\_cv\_test\_err <- 100 - oj\_cv\_test\_acc  
cat("Test error rate:", oj\_cv\_test\_err, "%", "\n")

## Test error rate: 18.51852 %

# Question 10

1. Removing rows with unknown salaries, and log transforming salaries

Hitters <- na.omit(Hitters)  
Hitters$Salary <- log(Hitters$Salary)

1. Creating a training and test set

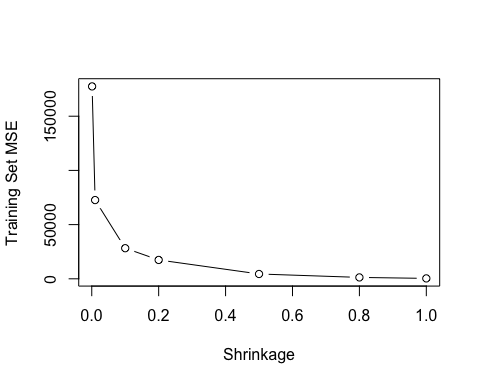
hitters\_train <- Hitters[1:200, ]  
hitters\_test <- Hitters[201:nrow(Hitters), ]

1. A range of trees with boosting on the training set

set.seed(123)  
shrinkage\_values <- c(0.001, 0.01, 0.1, 0.2, 0.5, 0.8, 1)  
results <- data.frame(shrinkage = shrinkage\_values, MSE = numeric(length(shrinkage\_values)))  
  
for (i in seq\_along(shrinkage\_values)) {  
 boost\_hitters <- gbm(Salary ~ ., data = hitters\_train, distribution = "gaussian",  
 n.trees = 1000, shrinkage = shrinkage\_values[i])  
 predictions <- predict(boost\_hitters, hitters\_train, n.trees = 1000)  
 mse <- mean((exp(predictions) - exp(hitters\_train$Salary))^2) # Back-transform from log scale  
 results$MSE[i] <- mse  
}

1. Here is a plot with shrinkage values on the x-axis and test MSE on the y-axis. The higher the shrinkage value, the lower the training set MSE.

plot(results$shrinkage, results$MSE, type = "b", xlab = "Shrinkage", ylab = "Training Set MSE")



e) The PCR model has the lowest MSE, followed by the linear model and then the boosted model.

set.seed(123)  
# Boosting model  
pred\_boost\_hitters <- predict(boost\_hitters, newdata = hitters\_test, n.trees = 1000, shrinkage = 1.0)  
mse\_boost\_hitters <- mean((exp(pred\_boost\_hitters) - exp(hitters\_test$Salary))^2) # Back-transform from log scale  
cat("Boosting Model Test MSE:", mse\_boost\_hitters, "\n")

## Boosting Model Test MSE: 354693.8

# Linear model  
hitters\_lm <- lm(Salary ~ ., data = hitters\_train)  
pred\_hitters\_lm <- predict(hitters\_lm, newdata = hitters\_test)  
mse\_hitters\_lm <- mean((exp(pred\_hitters\_lm) - exp(hitters\_test$Salary))^2) # Back-transform from log scale  
cat("Linear Model Test MSE:", mse\_hitters\_lm, "\n")

## Linear Model Test MSE: 137065.7

# PCR model  
hitters\_pcr <- pcr(Salary ~ ., data = hitters\_train, scale = TRUE, validation = "CV")  
validationplot(hitters\_pcr, val.type = "MSEP")

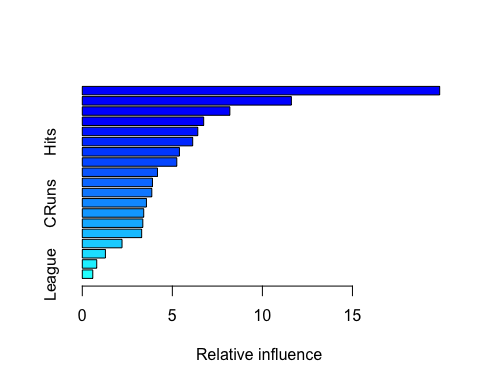


# 6 variables seems to have the lowest MSEP  
hitters\_pcr\_pred <- predict(hitters\_pcr, hitters\_test, ncomp = 6)  
mse\_hitters\_pcr <- mean((exp(hitters\_pcr\_pred) - exp(hitters\_test$Salary))^2) # Back-transform from log scale  
cat("PCR Model Test MSE:", mse\_hitters\_pcr, "\n")

## PCR Model Test MSE: 115643.7

1. The number of times at bat (CAtBat) is the most influential, followed by the number of put outs (PutOuts).

summary(boost\_hitters)



## var rel.inf  
## CAtBat CAtBat 19.8434204  
## PutOuts PutOuts 11.6081945  
## Assists Assists 8.1930720  
## Walks Walks 6.7398515  
## CWalks CWalks 6.4122610  
## Hits Hits 6.1297918  
## Years Years 5.3931904  
## AtBat AtBat 5.2471408  
## RBI RBI 4.1752480  
## CRBI CRBI 3.8959382  
## CHmRun CHmRun 3.8594817  
## CRuns CRuns 3.5617463  
## HmRun HmRun 3.4105441  
## CHits CHits 3.3602365  
## Errors Errors 3.2965162  
## Runs Runs 2.2058110  
## Division Division 1.2836438  
## NewLeague NewLeague 0.8015086  
## League League 0.5824032

1. The bagged model MSE is the best by far, at almost half of the PCR model MSE.

set.seed(123)  
bag\_hitters <- randomForest(Salary ~ ., data = hitters\_train, mtry = 19, importance = TRUE)  
bag\_hitters

##   
## Call:  
## randomForest(formula = Salary ~ ., data = hitters\_train, mtry = 19, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 19  
##   
## Mean of squared residuals: 0.2183193  
## % Var explained: 73.76

yhat\_bag\_hitters <- predict(bag\_hitters, newdata = hitters\_test)  
mse\_hitters\_bag <- mean((exp(yhat\_bag\_hitters) - exp(hitters\_test$Salary))^2) # Back-transform from log scale  
cat("Bagged Model Test MSE:", mse\_hitters\_bag, "\n")

## Bagged Model Test MSE: 52733.36