PQHS 471 HW 3

Gregory Powers

March 18, 2018

Chapter 7.9

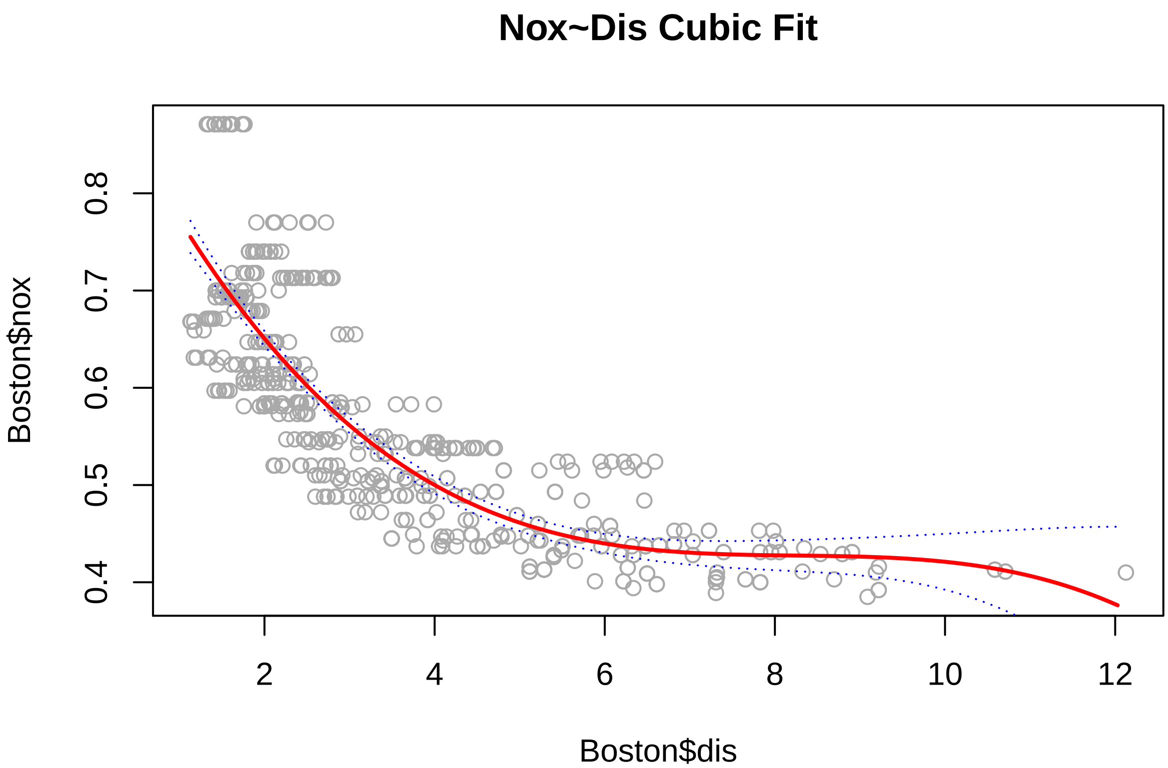
(a)

```
library(ISLR)
library(MASS)
library(ggplot2)
```

```
lm.1 <- lm(nox~poly(dis, 3), data = Boston)
summary(lm.1)
```

```
##
## Call:
## lm(formula = nox ~ poly(dis, 3), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.121130 -0.040619 -0.009738  0.023385  0.194904
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
## poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06207 on 502 degrees of freedom
## Multiple R-squared:  0.7148, Adjusted R-squared:  0.7131
## F-statistic: 419.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

```
dis.lim <- range(Boston$dis)
dis.grid <- seq(from = dis.lim[1], to = dis.lim[2], by = .1)
pred <- predict(lm.1, newdata = list(dis=dis.grid), se = TRUE)
se.bands <- cbind(pred$fit + 2*pred$se.fit, pred$fit - 2*pred$se.fit)
plot(Boston$nox~Boston$dis, xlim = dis.lim, cex = 1, col = "darkgrey")
title("Nox~Dis Cubic Fit")
lines(dis.grid, pred$fit, lwd = 2, col = "red")
matlines(dis.grid, se.bands, lwd = 1, col = "blue", lty = 3)
```



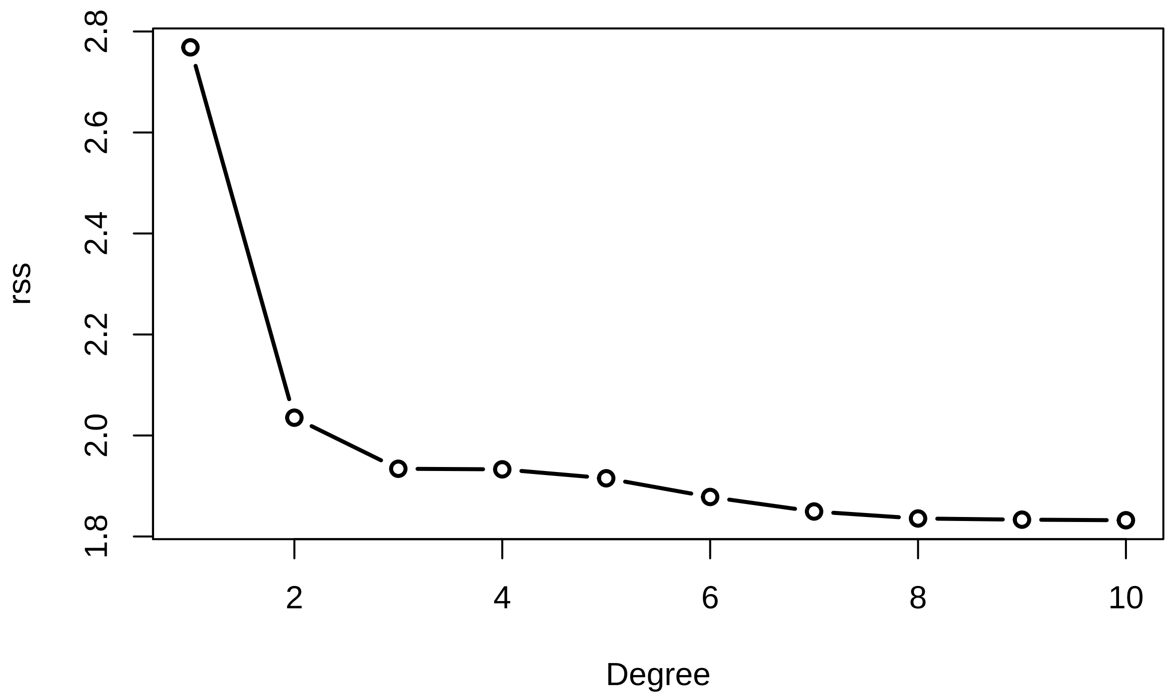
The cubic fit of nox~dis is good: all terms are significant and together account for 71% of the variance in nox. However, we can see from the plot that while the function fits well throughout most of the range of dis, the confidence bands grows larger as dis approaches 12, suggesting the presence of outliers.

(b)

```
rss = rep(0, 10)
for (i in 1:10) {
    lm.fit <- lm(nox ~ poly(dis, i), data = Boston)
    rss[i] <- sum(lm.fit$residuals^2)
}
rss
```

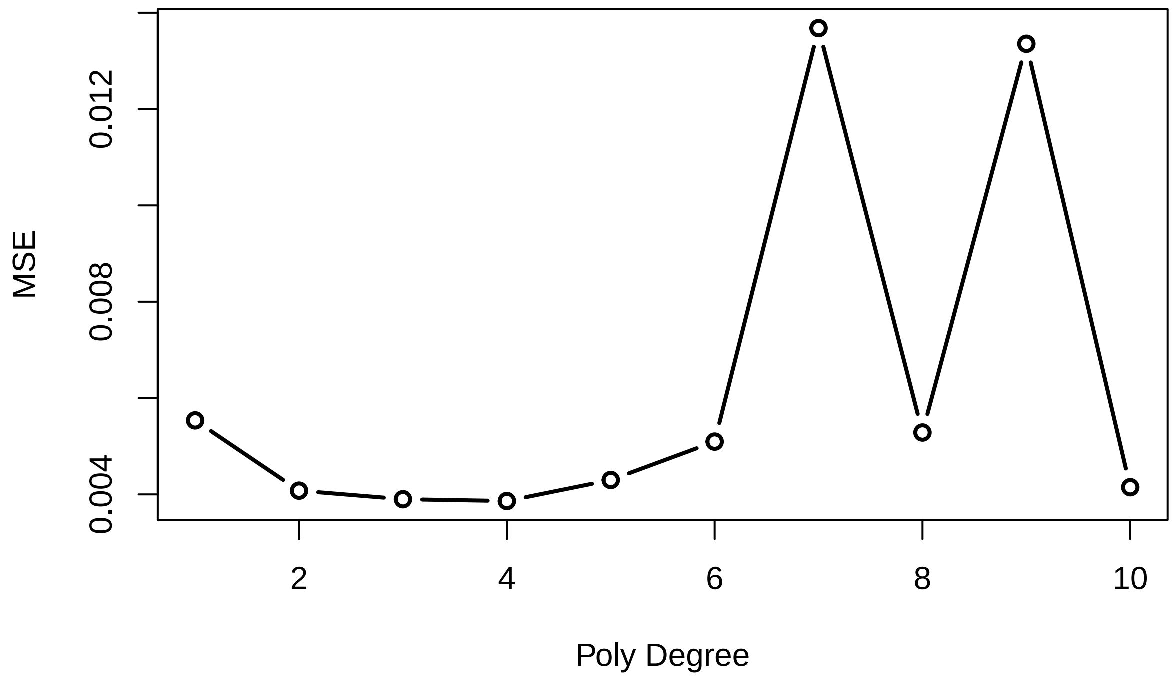
```
##  [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257 1.849484
##  [8] 1.835630 1.833331 1.832171
```

```
plot(rss, type = "b", xlab = "Degree", lwd = 2)
```




```
library(boot)
set.seed(1)
poly.cv <- rep(0, 10)
for (i in 1:10) {
    glm.fit <- glm(nox ~ poly(dis, i), data = Boston)
    poly.cv[i] <- cv.glm(Boston, glm.fit, K = 10)$delta[1] #  note to self: [1] = standard, [2] = bias-
}

plot(poly.cv, type = "b", lwd = 2, ylab = "MSE", xlab = "Poly Degree"); poly.cv
```




```
##  [1] 0.005536329 0.004077147 0.003899587 0.003862127 0.004298590
##  [6] 0.005095283 0.013680327 0.005284520 0.013355413 0.004148996
```

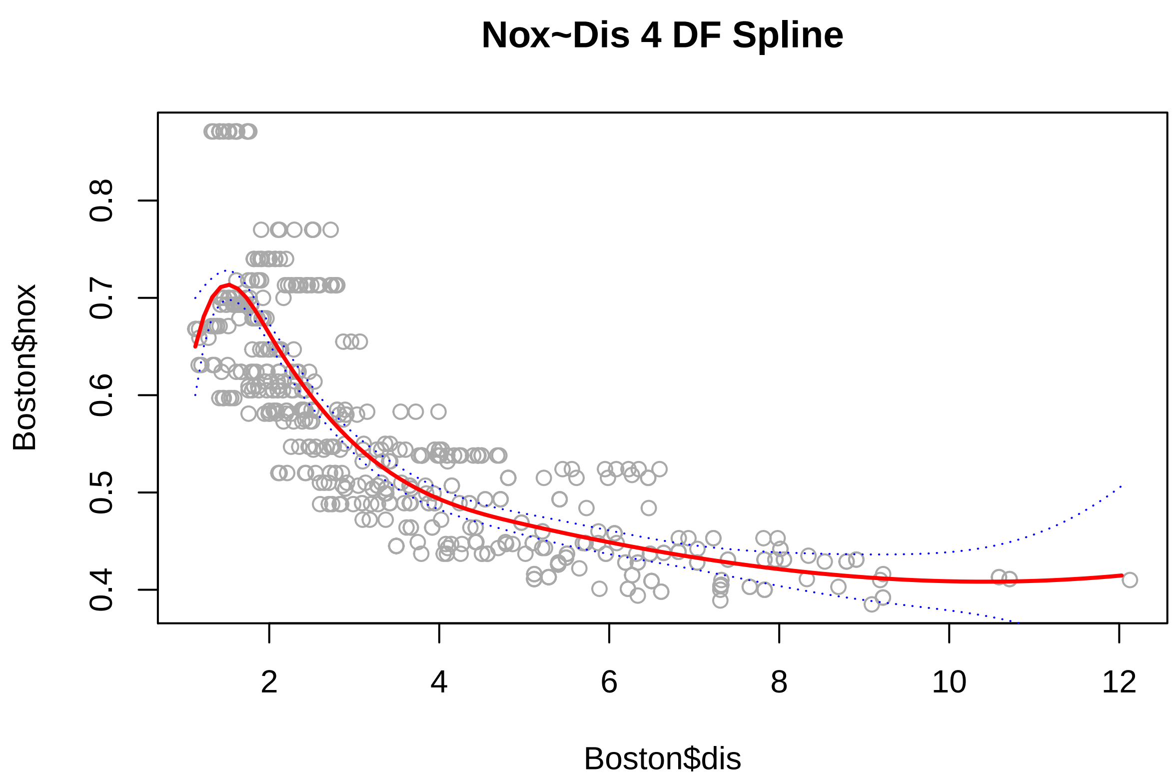
Judging from the graph and the poly.cv output, a 4th degree polynomial fit minimizes training error, albeit by a tiny margin over a 3rd degree fit.

```r
library(splines)
#length(nox)
fit.sp <- lm(nox~bs(dis, df = 4, knots = c(2,3,5)), data = Boston)
summary(fit.sp)
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, df = 4, knots = c(2, 3, 5)), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.12778 -0.03834 -0.01019  0.02273  0.19509
##
## Coefficients:
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                          0.649937   0.024948  26.052  < 2e-16
## bs(dis, df = 4, knots = c(2, 3, 5))1  0.107267   0.036455   2.942  0.00341
## bs(dis, df = 4, knots = c(2, 3, 5))2 -0.003582   0.024669  -0.145  0.88461
## bs(dis, df = 4, knots = c(2, 3, 5))3 -0.146759   0.028851  -5.087 5.16e-07
## bs(dis, df = 4, knots = c(2, 3, 5))4 -0.222917   0.034193  -6.519 1.73e-10
## bs(dis, df = 4, knots = c(2, 3, 5))5 -0.254819   0.049313  -5.167 3.44e-07
```

```
## bs(dis, df = 4, knots = c(2, 3, 5))6 -0.234438   0.054364  -4.312 1.95e-05
##
## (Intercept)                           ***
## bs(dis, df = 4, knots = c(2, 3, 5))1 **
## bs(dis, df = 4, knots = c(2, 3, 5))2
## bs(dis, df = 4, knots = c(2, 3, 5))3 ***
## bs(dis, df = 4, knots = c(2, 3, 5))4 ***
## bs(dis, df = 4, knots = c(2, 3, 5))5 ***
## bs(dis, df = 4, knots = c(2, 3, 5))6 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06057 on 499 degrees of freedom
## Multiple R-squared:  0.7301, Adjusted R-squared:  0.7268
## F-statistic: 224.9 on 6 and 499 DF,  p-value: < 2.2e-16
```

```
pred.sp <- predict(fit.sp, newdata = list(dis=dis.grid), se = TRUE)
se.bands.sp <- cbind(pred.sp$fit + 2*pred.sp$se.fit, pred.sp$fit - 2*pred.sp$se.fit)
plot(Boston$nox~Boston$dis, cex = 1, col = "darkgrey")
title("Nox~Dis 4 DF Spline")
lines(dis.grid, pred.sp$fit, lwd = 2, col = "red")
matlines(dis.grid, se.bands.sp, lwd = 1, col = "blue", lty = 3)
```



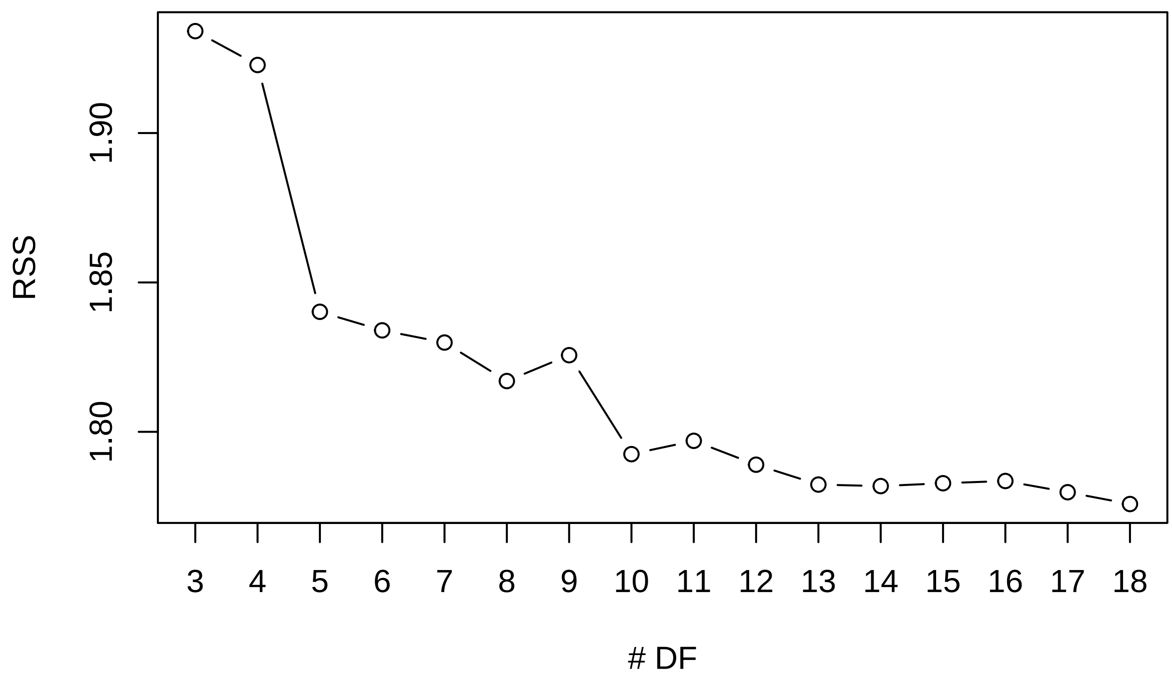
Knots were selected by summary statistics: the first, second and third quantiles.

(e)

```r
rss.sp <- rep(NA, 18)
for (i in 3:18) {
    fit.sp.rss <- lm(nox~bs(dis, df = i), data = Boston)
    rss.sp[i] <- sum(fit.sp.rss$residuals^2)
}
rss.sp <- rss.sp[-c(1,2)]; rss.sp
```

```
##  [1] 1.934107 1.922775 1.840173 1.833966 1.829884 1.816995 1.825653
##  [8] 1.792535 1.796992 1.788999 1.782350 1.781838 1.782798 1.783546
## [15] 1.779789 1.775838
```

```r
x <- c(3:18)
plot(3:18, rss.sp, type = "b", ylab = "RSS", xlab = "# DF", xaxt='n')
axis(1, at = x)
```



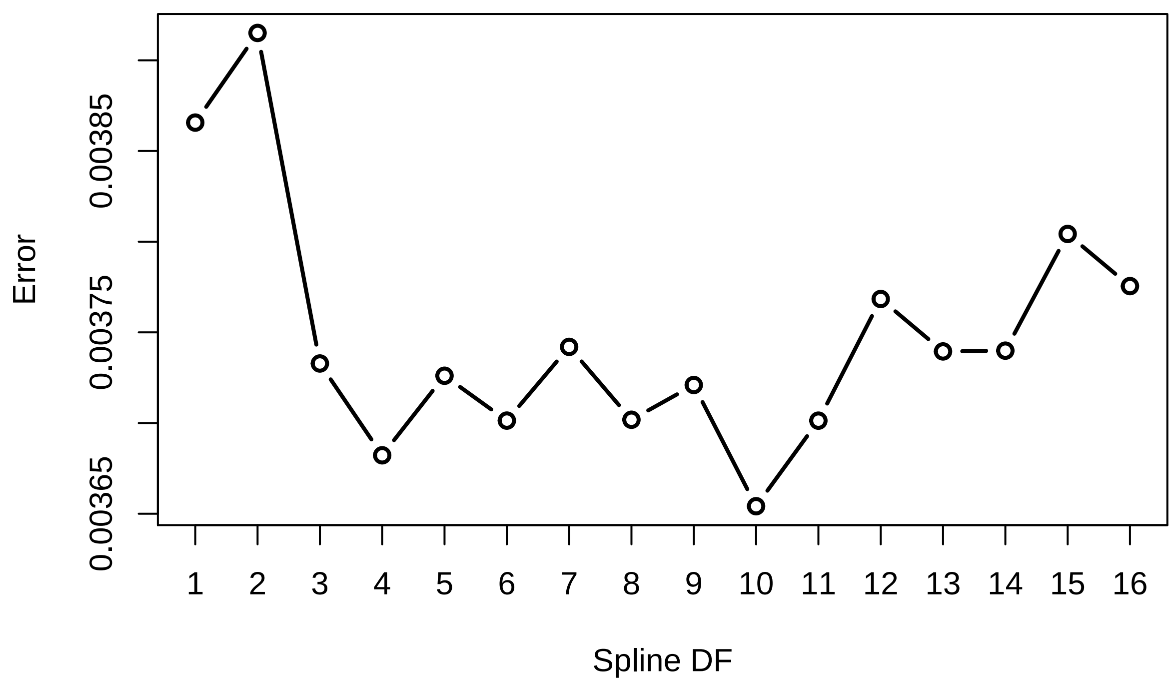
Error decreases more or less as flexibility increases, though not monotonically. There is a small increase DF 8 - DF 9, and again 10-11.

(f)

```r
set.seed(1)
bs.cv <- rep(NA, 18)
for (i in 3:18) {
    bs.fit <- glm(nox ~ bs(dis, df = i), data = Boston)
    bs.cv[i] <- cv.glm(Boston, bs.fit, K = 10)$delta[1] #  note to self: [1] = standard, [2] = bias-cor
}
bs.cv <- bs.cv[-c(1,2)]; bs.cv
```

```
##  [1] 0.003865662 0.003915074 0.003732840 0.003682221 0.003726081
##  [6] 0.003701344 0.003741995 0.003701813 0.003720982 0.003654150
## [11] 0.003701321 0.003768401 0.003739480 0.003739913 0.003804258
## [16] 0.003775504
```

```r
plot(bs.cv, type = "b", lwd = 2, ylab = "Error", xlab = "Spline DF", xaxt='n')
x <- c(1:18)
axis(1, at = x)
```



The minimum CV error is where DF = 10, though this varies depending on the RNG and may not be true if the set seed value is something other than 1. cv.glm issues a number of warnings which are suppressed for a more tidy output.

Chapter 8.9

(a)

```r
summary(OJ)
```

```
## Purchase WeekofPurchase     StoreID        PriceCH         PriceMM     
## CH:653   Min.   :227.0   Min.   :1.00   Min.   :1.690   Min.   :1.690  
## MM:417   1st Qu.:240.0   1st Qu.:2.00   1st Qu.:1.790   1st Qu.:1.990  
##          Median :257.0   Median :3.00   Median :1.860   Median :2.090  
##          Mean   :254.4   Mean   :3.96   Mean   :1.867   Mean   :2.085  
##          3rd Qu.:268.0   3rd Qu.:7.00   3rd Qu.:1.990   3rd Qu.:2.180  
##          Max.   :278.0   Max.   :7.00   Max.   :2.090   Max.   :2.290  
```

```
##      DiscCH            DiscMM          SpecialCH        SpecialMM
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.05186   Mean   :0.1234   Mean   :0.1477   Mean   :0.1617
##  3rd Qu.:0.00000   3rd Qu.:0.2300   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :0.50000   Max.   :0.8000   Max.   :1.0000   Max.   :1.0000
##     LoyalCH          SalePriceMM     SalePriceCH      PriceDiff
##  Min.   :0.000011   Min.   :1.190   Min.   :1.390   Min.   :-0.6700
##  1st Qu.:0.325257   1st Qu.:1.690   1st Qu.:1.750   1st Qu.: 0.0000
##  Median :0.600000   Median :2.090   Median :1.860   Median : 0.2300
##  Mean   :0.565782   Mean   :1.962   Mean   :1.816   Mean   : 0.1465
##  3rd Qu.:0.850873   3rd Qu.:2.130   3rd Qu.:1.890   3rd Qu.: 0.3200
##  Max.   :0.999947   Max.   :2.290   Max.   :2.090   Max.   : 0.6400
##  Store7      PctDiscMM        PctDiscCH        ListPriceDiff
##  No :714   Min.   :0.0000   Min.   :0.00000   Min.   :0.000
##  Yes:356   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.140
##            Median :0.0000   Median :0.00000   Median :0.240
##            Mean   :0.0593   Mean   :0.02731   Mean   :0.218
##            3rd Qu.:0.1127   3rd Qu.:0.00000   3rd Qu.:0.300
##            Max.   :0.4020   Max.   :0.25269   Max.   :0.440
##      STORE
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :2.000
##  Mean   :1.631
##  3rd Qu.:3.000
##  Max.   :4.000
```

```
str(OJ)
```

```
## 'data.frame':    1070 obs. of  18 variables:
##  $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
##  $ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
##  $ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
##  $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
##  $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
##  $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
##  $ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
##  $ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
##  $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
##  $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
##  $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
##  $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
##  $ PctDiscMM     : num  0 0.151 0 0 0 ...
##  $ PctDiscCH     : num  0 0 0.0914 0 0 ...
##  $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
##  $ STORE         : num  1 1 1 1 0 0 0 0 0 0 ...
```

```
anyDuplicated(OJ)
```

```
## [1] 439
```

```r
sum(is.na(OJ))
```

```
## [1] 0
```

```r
set.seed(1)
train <- sample(1:nrow(OJ), 800)
oj.train <- OJ[train,]
oj.test <- OJ[-train,]
```

(b)

```r
library(tree)
tree.fit <- tree(Purchase~., data = oj.train)
summary(tree.fit)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = oj.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7305 = 578.6 / 792
## Misclassification error rate: 0.165 = 132 / 800
```

The tree has 8 nodes and a training error rate of 0.165

(c)

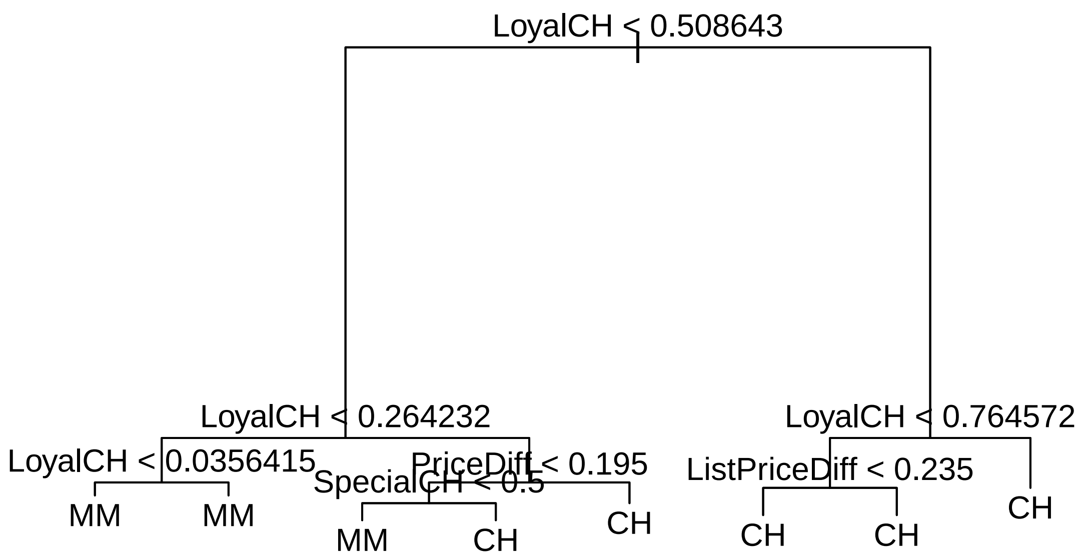
```r
tree.fit
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1064.00 CH ( 0.61750 0.38250 )
##    2) LoyalCH < 0.508643 350  409.30 MM ( 0.27143 0.72857 )
##      4) LoyalCH < 0.264232 166  122.10 MM ( 0.12048 0.87952 )
##        8) LoyalCH < 0.0356415 57   10.07 MM ( 0.01754 0.98246 ) *
##        9) LoyalCH > 0.0356415 109  100.90 MM ( 0.17431 0.82569 ) *
##      5) LoyalCH > 0.264232 184  248.80 MM ( 0.40761 0.59239 )
##       10) PriceDiff < 0.195 83   91.66 MM ( 0.24096 0.75904 )
##         20) SpecialCH < 0.5 70   60.89 MM ( 0.15714 0.84286 ) *
##         21) SpecialCH > 0.5 13   16.05 CH ( 0.69231 0.30769 ) *
##       11) PriceDiff > 0.195 101  139.20 CH ( 0.54455 0.45545 ) *
##    3) LoyalCH > 0.508643 450  318.10 CH ( 0.88667 0.11333 )
##      6) LoyalCH < 0.764572 172  188.90 CH ( 0.76163 0.23837 )
##       12) ListPriceDiff < 0.235 70   95.61 CH ( 0.57143 0.42857 ) *
##       13) ListPriceDiff > 0.235 102   69.76 CH ( 0.89216 0.10784 ) *
##      7) LoyalCH > 0.764572 278   86.14 CH ( 0.96403 0.03597 ) *
```

20 is a terminal node that predicts MM. The split criterion is SpecialCH $< .5$. The branch has 70 observations with a deviance of 60.89. About 16% of the branch has Sales = CH, 84% Sales = MM.

(d)

```r
plot(tree.fit)
text(tree.fit)
```



LoyalCH appears to be the most important predictor of Purchase.

(e)

```
tree.pred <- predict(tree.fit, oj.test, type = "class")
table(tree.pred, oj.test$Purchase)
```

```
##
## tree.pred  CH  MM
##        CH 147  49
##        MM  12  62
```

```
1-(209)/270
```

```
## [1] 0.2259259
```

Our test error rate is about 23%.

(f)

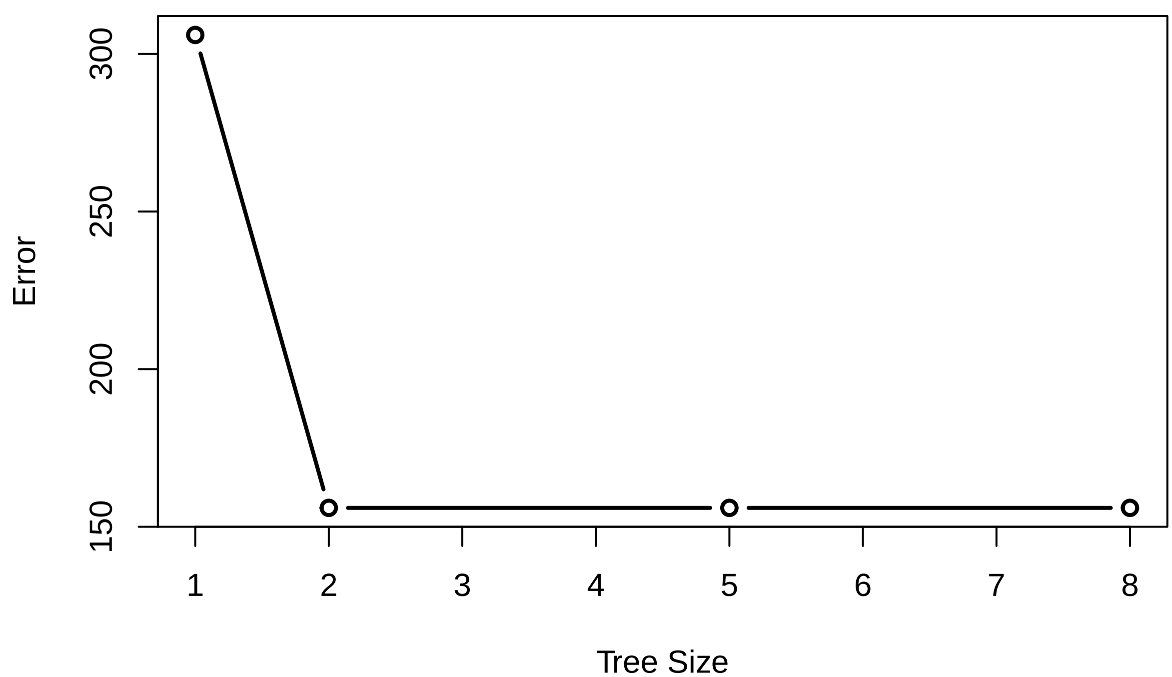
```
cv.oj <- cv.tree(tree.fit, FUN = prune.misclass); cv.oj
```

```
## $size
## [1] 8 5 2 1
##
## $dev
## [1] 156 156 156 306
##
## $k
## [1]       -Inf   0.000000   4.666667 160.000000
```

```
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

(g)

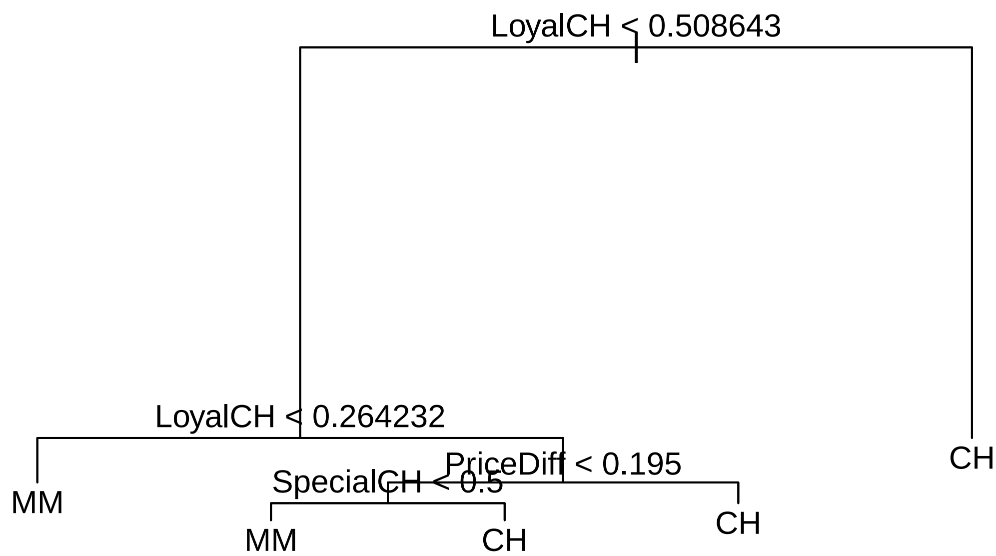
```
plot(cv.oj$size, cv.oj$dev, type = "b", lwd = 2, ylab = "Error", xlab = "Tree Size")
```



(h) A tree size of 5 is associated with the lowest error rate.

(i)

```
prune.fit <- prune.misclass(tree.fit, best = 5)
plot(prune.fit)
text(prune.fit)
```



(j)

```r
summary(tree.fit)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = oj.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7305 = 578.6 / 792
## Misclassification error rate: 0.165 = 132 / 800
```

```r
summary(prune.fit)
```

```
##
## Classification tree:
## snip.tree(tree = tree.fit, nodes = 3:4)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff" "SpecialCH"
## Number of terminal nodes:  5
## Residual mean deviance:  0.8256 = 656.4 / 795
## Misclassification error rate: 0.165 = 132 / 800
```

The train error rates are identical.

(k)

```r
prune.pred <- predict(prune.fit, oj.test, type = "class")
table(prune.pred, oj.test$Purchase)
```

```
##
## prune.pred  CH  MM
##         CH 147  49
##         MM  12  62
```

```r
1-(209)/270
```

```
## [1] 0.2259259
```

The test error rate of the pruned and unpruned models are identical.

9.1.6 Khan Data

```r
summary(Khan)
```

```
##        Length Class  Mode
## xtrain 145404 -none- numeric
## xtest   46160 -none- numeric
## ytrain     63 -none- numeric
## ytest      20 -none- numeric
```

```r
str(Khan)
```

```
## List of 4
##  $ xtrain: num [1:63, 1:2308] 0.7733 -0.0782 -0.0845 0.9656 0.0757 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:63] "V1" "V2" "V3" "V4" ...
##   .. ..$ : NULL
##  $ xtest : num [1:20, 1:2308] 0.14 1.164 0.841 0.685 -1.956 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:20] "V1" "V2" "V4" "V6" ...
##   .. ..$ : NULL
##  $ ytrain: num [1:63] 2 2 2 2 2 2 2 2 2 2 ...
##  $ ytest : num [1:20] 3 2 4 2 1 3 4 2 3 1 ...
```

```r
anyDuplicated(Khan)
```

```
## [1] 0
```

```r
sum(is.na(Khan))
```

```
## [1] 0
```

```r
table(Khan$ytrain)
```

```
##
##  1  2  3  4
##  8 23 12 20
```

```r
khan.train <- data.frame(x = Khan$xtrain, y = as.factor(Khan$ytrain))
dim(khan.train)
```

```
## [1]   63 2309
```

```r
khan.test <- data.frame(x = Khan$xtest, y = as.factor(Khan$ytest))
dim(khan.test)
```

```
## [1]   20 2309
```

```r
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'lattice'

## The following object is masked from 'package:boot':
##
##     melanoma
```

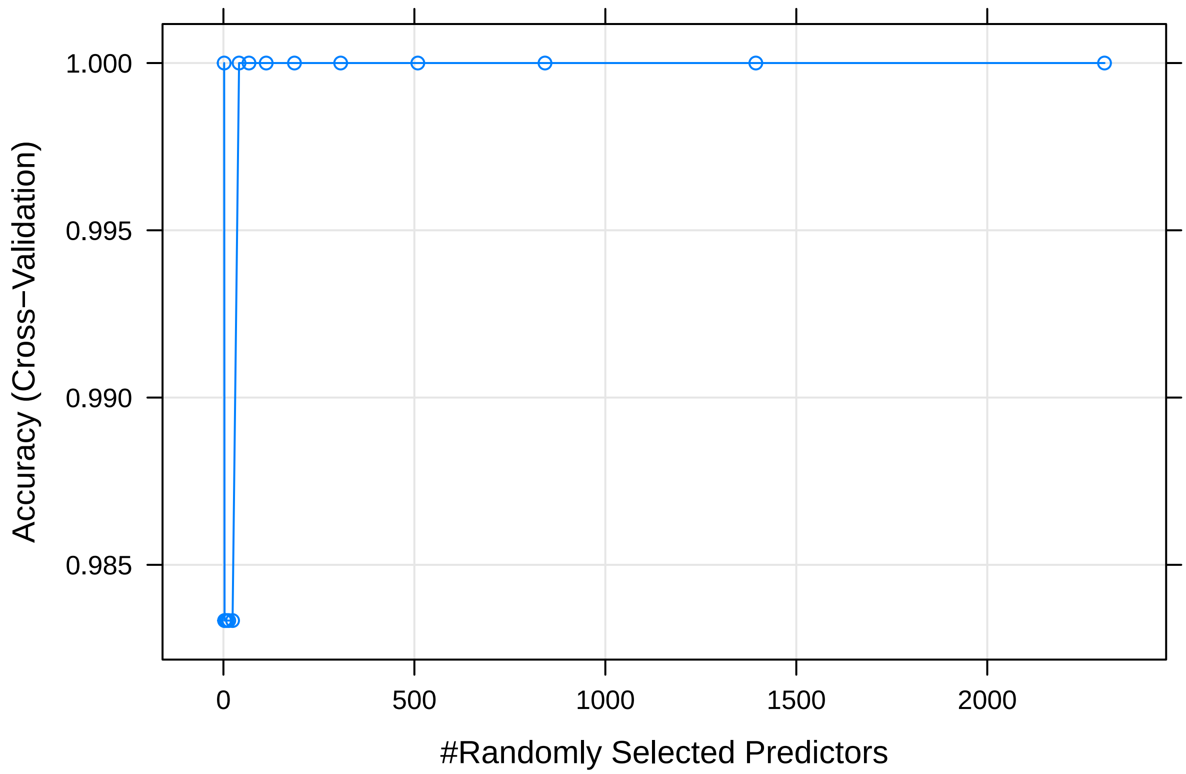
```r
set.seed(1)
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
tunegrid <- expand.grid(.mtry=c(1:10))
train.control <- trainControl(method="cv", number=10, search = "grid",  allowParallel = TRUE)
model.for <- train(y~., data=khan.train, method="rf", tuneLength = 15, truneGrid = tunegrid, trControl=t
stopCluster(cluster)
registerDoSEQ()
```

```r
print(model.for)
```

```
## Random Forest
##
##   63 samples
## 2308 predictors
##    4 classes: '1', '2', '3', '4'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 57, 56, 57, 56, 57, 58, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##     2   1.0000000  1.0000000
##     3   0.9833333  0.9769231
##     5   0.9833333  0.9769231
##     9   0.9833333  0.9769231
##    14   0.9833333  0.9769231
##    24   0.9833333  0.9769231
##    41   1.0000000  1.0000000
##    67   1.0000000  1.0000000
##   112   1.0000000  1.0000000
##   186   1.0000000  1.0000000
##   307   1.0000000  1.0000000
##   509   1.0000000  1.0000000
```

```
##    842  1.0000000  1.0000000
##   1394  1.0000000  1.0000000
##   2307  1.0000000  1.0000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```r
plot(model.for)
```




```r
summary(model.for)
```

```
##                 Length Class      Mode
## call               6   -none-     call
## type               1   -none-     character
## predicted         63   factor     numeric
## err.rate        7500   -none-     numeric
## confusion         20   -none-     numeric
## votes            252   matrix     numeric
## oob.times         63   -none-     numeric
## classes            4   -none-     character
## importance      2308   -none-     numeric
## importanceSD       0   -none-     NULL
## localImportance    0   -none-     NULL
## proximity          0   -none-     NULL
## ntree              1   -none-     numeric
## mtry               1   -none-     numeric
## forest            14   -none-     list
```

```
## y                 63   factor     numeric
## test               0   -none-     NULL
## inbag              0   -none-     NULL
## xNames          2308   -none-     character
## problemType        1   -none-     character
## tuneValue          1   data.frame list
## obsLevels          4   -none-     character
## param              2   -none-     list
```

```
predict.for <- predict.train(object=model.for, khan.test, type="raw")
confusionMatrix(predict.for, khan.test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2 3 4
##          1 3 0 0 0
##          2 0 5 1 0
##          3 0 0 1 0
##          4 0 1 4 5
##
## Overall Statistics
##
##                Accuracy : 0.7
##                  95% CI : (0.4572, 0.8811)
##     No Information Rate : 0.3
##     P-Value [Acc > NIR] : 0.000261
##
##                   Kappa : 0.5987
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity              1.00   0.8333   0.1667   1.0000
## Specificity              1.00   0.9286   1.0000   0.6667
## Pos Pred Value           1.00   0.8333   1.0000   0.5000
## Neg Pred Value           1.00   0.9286   0.7368   1.0000
## Prevalence               0.15   0.3000   0.3000   0.2500
## Detection Rate           0.15   0.2500   0.0500   0.2500
## Detection Prevalence     0.15   0.3000   0.0500   0.5000
## Balanced Accuracy        1.00   0.8810   0.5833   0.8333
```

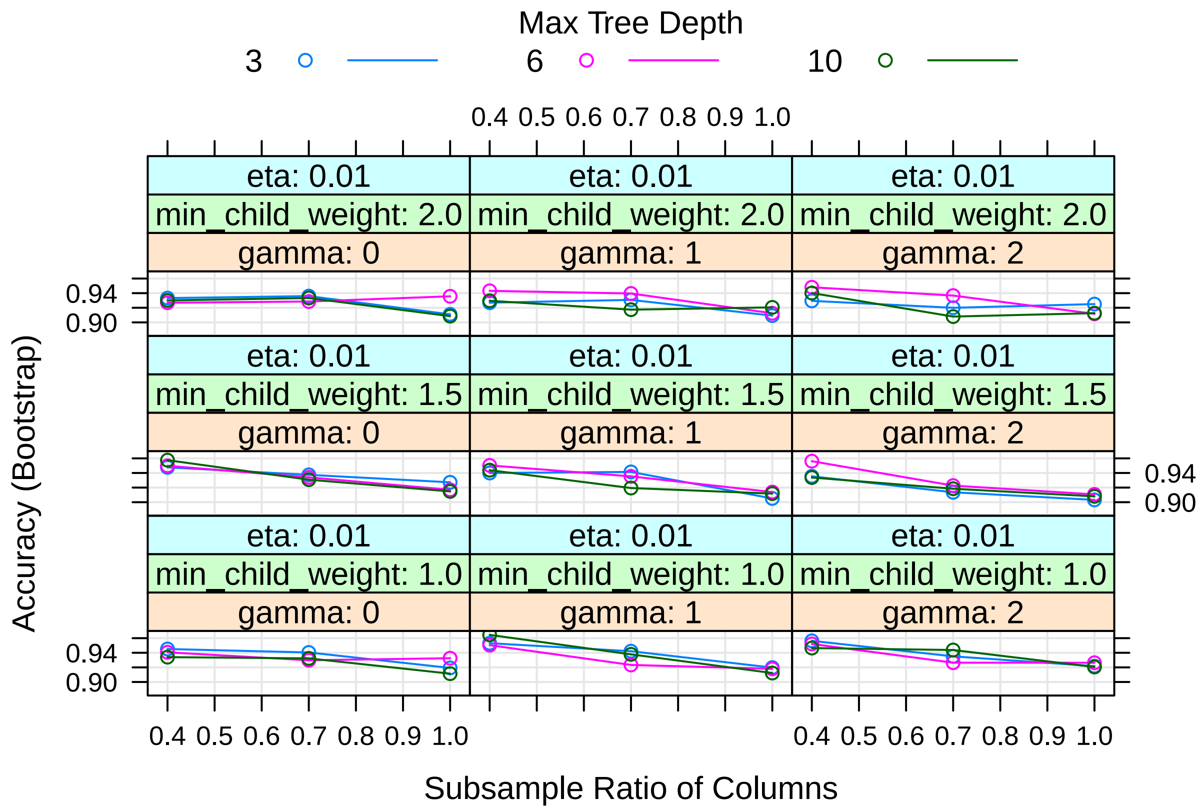
```
varImp(model.for)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 2308)
##
##        Overall
## x.545   100.00
## x.187    99.46
## x.607    93.78
## x.2050   92.42
## x.2046   85.50
```

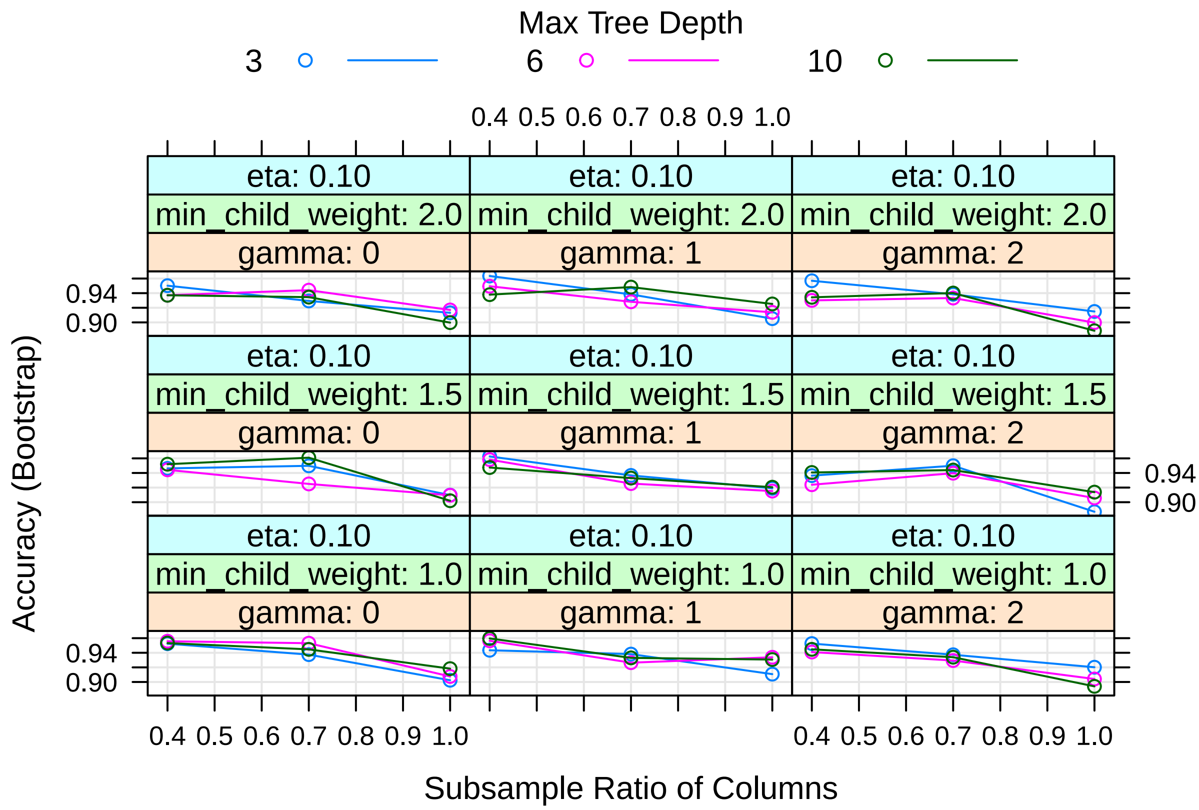
```
## x.380    84.83
## x.417    79.73
## x.1389   79.12
## x.430    76.24
## x.1706   72.98
## x.1110   72.36
## x.554    71.26
## x.1055   71.22
## x.1645   71.10
## x.246    70.64
## x.335    70.37
## x.1159   70.30
## x.1194   69.65
## x.1105   68.44
## x.166    68.30
```

```r
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(1)
grid.xg = expand.grid(
  nrounds = 10,
  max_depth = c(3, 6,10),
  eta = c(0.1, 0.01),
  gamma = c(0,1,2),
  colsample_bytree = c(0.4, 0.7, 1.0),
  min_child_weight = c(1, 1.5, 2),
  subsample = .5
)
#train.control <- trainControl(method="cv", number=5,  allowParallel = TRUE)
model.gbm = train(y ~., method = "xgbTree", tuneGrid = grid.xg, data = khan.train, verbose = FALSE);
stopCluster(cluster)
registerDoSEQ()
```

```r
plot(model.gbm)
```



Max Tree Depth
3
6
10
eta: 0.01
min_child_weight: 2.0
gamma: 0
gamma: 1
gamma: 2
min_child_weight: 1.5
min_child_weight: 1.0
0.94
0.90
0.4 0.5 0.6 0.7 0.8 0.9 1.0
Accuracy (Bootstrap)
Subsample Ratio of Columns




```r
predict.xgb <- predict.train(object=model.gbm, khan.test, type="raw")
print(head(predict.xgb))
```

```
## [1] 4 2 4 2 1 3
## Levels: 1 2 3 4
```

```r
confusionMatrix(predict.xgb, khan.test$y)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction 1 2 3 4
##          1 3 0 0 0
##          2 0 6 0 0
##          3 0 0 5 0
##          4 0 0 1 5
##
## Overall Statistics
##
##                Accuracy : 0.95
##                  95% CI : (0.7513, 0.9987)
##     No Information Rate : 0.3
##     P-Value [Acc > NIR] : 1.662e-09
##
##                   Kappa : 0.9322
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity              1.00      1.0   0.8333   1.0000
## Specificity              1.00      1.0   1.0000   0.9333
## Pos Pred Value           1.00      1.0   1.0000   0.8333
## Neg Pred Value           1.00      1.0   0.9333   1.0000
## Prevalence               0.15      0.3   0.3000   0.2500
## Detection Rate           0.15      0.3   0.2500   0.2500
## Detection Prevalence     0.15      0.3   0.2500   0.3000
## Balanced Accuracy        1.00      1.0   0.9167   0.9667
```

```
#varImp(model.gbm)
```

Both models perform well, having test error rates of 5%. That said, the CI is smaller for the random forest model.

The random forest model a very wide search grid for mtry and resulted in a 5% error rate. This model was far harder to tune because with CV it takes much longer than XGB.

Chapter 9.8

(a)

```
set.seed(11)
train <- sample(1:nrow(OJ), 800)
oj.train <- OJ[train,]
oj.test <- OJ[-train,]
```

(b)

```
library(e1071)
svm.lin <- svm(Purchase ~ ., data = oj.train, kernel = "linear", cost = 0.01)
summary(svm.lin)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj.train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  429
##
##  ( 216 213 )
##
##
## Number of Classes:  2
##
## Levels:
```

```
##  CH MM
```

There were 429 support vectors, 216 for the class CH and 213 for the class MM.

(c)

```r
svm.lin.pred <- predict(svm.lin, oj.train)
confusionMatrix(svm.lin.pred, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 454  77
##         MM  49 220
##
##                Accuracy : 0.8425
##                  95% CI : (0.8154, 0.8671)
##     No Information Rate : 0.6288
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.656
##  Mcnemar's Test P-Value : 0.01616
##
##             Sensitivity : 0.9026
##             Specificity : 0.7407
##          Pos Pred Value : 0.8550
##          Neg Pred Value : 0.8178
##              Prevalence : 0.6288
##          Detection Rate : 0.5675
##    Detection Prevalence : 0.6637
##       Balanced Accuracy : 0.8217
##
##        'Positive' Class : CH
##
```

```r
1-.8425
```

```
## [1] 0.1575
```

```r
svm.lin.pred1 <- predict(svm.lin, oj.test)
confusionMatrix(svm.lin.pred1, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 131  32
##         MM  19  88
##
##                Accuracy : 0.8111
##                  95% CI : (0.7592, 0.856)
##     No Information Rate : 0.5556
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6133
##  Mcnemar's Test P-Value : 0.09289
```

```
##
##             Sensitivity : 0.8733
##             Specificity : 0.7333
##          Pos Pred Value : 0.8037
##          Neg Pred Value : 0.8224
##              Prevalence : 0.5556
##          Detection Rate : 0.4852
##    Detection Prevalence : 0.6037
##       Balanced Accuracy : 0.8033
##
##        'Positive' Class : CH
##
```

```r
1-.8111
```

```
## [1] 0.1889
```

The training error rate is 15.75% and the test error rate is 18.89%.

(d)

```r
set.seed(1)
tune.svm <- tune(svm, Purchase~., data = oj.train, kernel = "linear", ranges = list(cost = seq(.01, 10,
summary(tune.svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.41
##
## - best performance: 0.15875
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.16250 0.01666667
## 2  0.21 0.16125 0.02389938
## 3  0.41 0.15875 0.02045490
## 4  0.61 0.15875 0.02208726
## 5  0.81 0.16125 0.02389938
## 6  1.01 0.16375 0.02729087
## 7  1.21 0.16375 0.02729087
## 8  1.41 0.16500 0.02622022
## 9  1.61 0.16500 0.02622022
## 10 1.81 0.16500 0.02622022
## 11 2.01 0.16500 0.02622022
## 12 2.21 0.16500 0.02622022
## 13 2.41 0.16500 0.02622022
## 14 2.61 0.16500 0.02622022
## 15 2.81 0.16500 0.02622022
## 16 3.01 0.16375 0.02531057
## 17 3.21 0.16375 0.02531057
## 18 3.41 0.16375 0.02531057
## 19 3.61 0.16500 0.02415229
```

```
## 20 3.81 0.16500 0.02415229
## 21 4.01 0.16500 0.02415229
## 22 4.21 0.16500 0.02415229
## 23 4.41 0.16625 0.02503470
## 24 4.61 0.16625 0.02503470
## 25 4.81 0.16625 0.02503470
## 26 5.01 0.16625 0.02503470
## 27 5.21 0.16625 0.02503470
## 28 5.41 0.16625 0.02503470
## 29 5.61 0.16625 0.02503470
## 30 5.81 0.16625 0.02503470
## 31 6.01 0.16625 0.02503470
## 32 6.21 0.16625 0.02503470
## 33 6.41 0.16625 0.02503470
## 34 6.61 0.16625 0.02503470
## 35 6.81 0.16625 0.02503470
## 36 7.01 0.16625 0.02503470
## 37 7.21 0.16625 0.02503470
## 38 7.41 0.16625 0.02503470
## 39 7.61 0.16625 0.02503470
## 40 7.81 0.16625 0.02503470
## 41 8.01 0.16625 0.02503470
## 42 8.21 0.16625 0.02503470
## 43 8.41 0.16625 0.02503470
## 44 8.61 0.16625 0.02503470
## 45 8.81 0.16750 0.02443813
## 46 9.01 0.16750 0.02443813
## 47 9.21 0.16750 0.02443813
## 48 9.41 0.16750 0.02443813
## 49 9.61 0.16750 0.02443813
## 50 9.81 0.16625 0.02503470
```

The optimal cost = 0.41.

(e)

```r
svm.lin <- svm(Purchase ~ ., data = oj.train, kernel = "linear", cost = .41)
summary(svm.lin)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj.train, kernel = "linear",
##     cost = 0.41)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.41
##       gamma:  0.05555556
##
## Number of Support Vectors:  326
##
##  ( 161 165 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
svm.lin.pred <- predict(svm.lin, oj.train)
confusionMatrix(svm.lin.pred, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 453  76
##         MM  50 221
##
##                Accuracy : 0.8425
##                  95% CI : (0.8154, 0.8671)
##     No Information Rate : 0.6288
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6565
##  Mcnemar's Test P-Value : 0.02594
##
##             Sensitivity : 0.9006
##             Specificity : 0.7441
##          Pos Pred Value : 0.8563
##          Neg Pred Value : 0.8155
##              Prevalence : 0.6288
##          Detection Rate : 0.5663
##    Detection Prevalence : 0.6613
##       Balanced Accuracy : 0.8224
##
##        'Positive' Class : CH
##
```

```r
1-.8425
```

```
## [1] 0.1575
```

```r
svm.lin.pred1 <- predict(svm.lin, oj.test)
confusionMatrix(svm.lin.pred1, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 130  33
##         MM  20  87
##
##                Accuracy : 0.8037
##                  95% CI : (0.7512, 0.8494)
##     No Information Rate : 0.5556
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5981
##  Mcnemar's Test P-Value : 0.09929
```

```
##
##             Sensitivity : 0.8667
##             Specificity : 0.7250
##          Pos Pred Value : 0.7975
##          Neg Pred Value : 0.8131
##              Prevalence : 0.5556
##          Detection Rate : 0.4815
##    Detection Prevalence : 0.6037
##       Balanced Accuracy : 0.7958
##
##        'Positive' Class : CH
##
```

```r
1-.8037
```

```
## [1] 0.1963
```

Training error = 15.75%, test error = 19.63%. Not an improvement.

(f)

```r
svm.rad <- svm(Purchase ~ ., data = oj.train, kernel = "radial")
summary(svm.rad)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  362
##
##  ( 182 180 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
svm.rad.pred <- predict(svm.rad, oj.train)
confusionMatrix(svm.rad.pred, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 463  80
##         MM  40 217
##
##                Accuracy : 0.85
##                  95% CI : (0.8233, 0.874)
```

```
##     No Information Rate : 0.6288
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6696
##  Mcnemar's Test P-Value : 0.0003706
##
##             Sensitivity : 0.9205
##             Specificity : 0.7306
##          Pos Pred Value : 0.8527
##          Neg Pred Value : 0.8444
##              Prevalence : 0.6288
##          Detection Rate : 0.5787
##    Detection Prevalence : 0.6787
##       Balanced Accuracy : 0.8256
##
##        'Positive' Class : CH
##
```

```r
1-.85
```

```
## [1] 0.15
```

```r
svm.rad.pred1 <- predict(svm.rad, oj.test)
confusionMatrix(svm.rad.pred1, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 133  34
##         MM  17  86
##
##                Accuracy : 0.8111
##                  95% CI : (0.7592, 0.856)
##     No Information Rate : 0.5556
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.612
##  Mcnemar's Test P-Value : 0.02506
##
##             Sensitivity : 0.8867
##             Specificity : 0.7167
##          Pos Pred Value : 0.7964
##          Neg Pred Value : 0.8350
##              Prevalence : 0.5556
##          Detection Rate : 0.4926
##    Detection Prevalence : 0.6185
##       Balanced Accuracy : 0.8017
##
##        'Positive' Class : CH
##
```

```r
1-.8111
```

```
## [1] 0.1889
```

Radial SVM train error is 15%, test 18.89%. This is not an improvement over linear SVM.

```r
set.seed(1)
tune.svm <- tune(svm, Purchase~., data = oj.train, kernel = "radial", ranges = list(cost = seq(.01, 10,
summary(tune.svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.61
##
## - best performance: 0.16625
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.37125 0.04752558
## 2  0.21 0.17500 0.01954340
## 3  0.41 0.17000 0.02776389
## 4  0.61 0.16625 0.02503470
## 5  0.81 0.16625 0.02703521
## 6  1.01 0.16625 0.02949223
## 7  1.21 0.16875 0.02447363
## 8  1.41 0.16875 0.02585349
## 9  1.61 0.17000 0.03184162
## 10 1.81 0.16875 0.02960973
## 11 2.01 0.17000 0.02958040
## 12 2.21 0.17250 0.02813657
## 13 2.41 0.17500 0.02886751
## 14 2.61 0.17500 0.02886751
## 15 2.81 0.17500 0.02886751
## 16 3.01 0.17875 0.03175973
## 17 3.21 0.18000 0.03238227
## 18 3.41 0.18000 0.03073181
## 19 3.61 0.17875 0.03064696
## 20 3.81 0.17875 0.03064696
## 21 4.01 0.17875 0.03064696
## 22 4.21 0.17875 0.03064696
## 23 4.41 0.17875 0.03064696
## 24 4.61 0.17875 0.03064696
## 25 4.81 0.17875 0.02829041
## 26 5.01 0.17750 0.02874698
## 27 5.21 0.17750 0.02874698
## 28 5.41 0.17750 0.02874698
## 29 5.61 0.17750 0.02874698
## 30 5.81 0.17875 0.03007514
## 31 6.01 0.17750 0.02751262
## 32 6.21 0.17750 0.02751262
## 33 6.41 0.17875 0.02889757
## 34 6.61 0.17750 0.03050501
## 35 6.81 0.18000 0.02838231
## 36 7.01 0.18000 0.02838231
## 37 7.21 0.18125 0.03076005
```

```
## 38 7.41 0.18250 0.03184162
## 39 7.61 0.18250 0.03184162
## 40 7.81 0.18375 0.03064696
## 41 8.01 0.18250 0.02898755
## 42 8.21 0.18250 0.02898755
## 43 8.41 0.18250 0.02838231
## 44 8.61 0.18250 0.02838231
## 45 8.81 0.18250 0.02838231
## 46 9.01 0.18250 0.02838231
## 47 9.21 0.18375 0.02829041
## 48 9.41 0.18375 0.02829041
## 49 9.61 0.18375 0.02829041
## 50 9.81 0.18375 0.02829041
```

```r
svm.rad <- svm(Purchase ~ ., data = oj.train, kernel = "radial", cost = .61)
summary(svm.rad)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj.train, kernel = "radial",
##     cost = 0.61)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.61
##       gamma:  0.05555556
##
## Number of Support Vectors:  384
##
##  ( 193 191 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
svm.rad.pred <- predict(svm.rad, oj.train)
confusionMatrix(svm.rad.pred, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 462  77
##         MM  41 220
##
##                Accuracy : 0.8525
##                  95% CI : (0.826, 0.8764)
##     No Information Rate : 0.6288
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.676
```

```
##  Mcnemar's Test P-Value : 0.001273
##
##             Sensitivity : 0.9185
##             Specificity : 0.7407
##          Pos Pred Value : 0.8571
##          Neg Pred Value : 0.8429
##              Prevalence : 0.6288
##          Detection Rate : 0.5775
##    Detection Prevalence : 0.6737
##       Balanced Accuracy : 0.8296
##
##        'Positive' Class : CH
##
```

```
1-.8525
```

```
## [1] 0.1475
```

```
svm.rad.pred1 <- predict(svm.rad, oj.test)
confusionMatrix(svm.rad.pred1, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 133  36
##         MM  17  84
##
##                Accuracy : 0.8037
##                  95% CI : (0.7512, 0.8494)
##     No Information Rate : 0.5556
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5961
##  Mcnemar's Test P-Value : 0.01342
##
##             Sensitivity : 0.8867
##             Specificity : 0.7000
##          Pos Pred Value : 0.7870
##          Neg Pred Value : 0.8317
##              Prevalence : 0.5556
##          Detection Rate : 0.4926
##    Detection Prevalence : 0.6259
##       Balanced Accuracy : 0.7933
##
##        'Positive' Class : CH
##
```

```
1-.8037
```

```
## [1] 0.1963
```

Again, not an improvement: tune–or at least the way I am using it–seems to increase variance.

(g)

```
svm.poly<- svm(Purchase ~ ., data = oj.train, kernel = "polynomial", degree = 2)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj.train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  425
##
##  ( 215 210 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
svm.poly.pred <- predict(svm.poly, oj.train)
confusionMatrix(svm.poly.pred, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 471 103
##         MM  32 194
##
##                Accuracy : 0.8312
##                  95% CI : (0.8035, 0.8566)
##     No Information Rate : 0.6288
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6199
##  Mcnemar's Test P-Value : 1.695e-09
##
##             Sensitivity : 0.9364
##             Specificity : 0.6532
##          Pos Pred Value : 0.8206
##          Neg Pred Value : 0.8584
##              Prevalence : 0.6288
##          Detection Rate : 0.5887
##    Detection Prevalence : 0.7175
##       Balanced Accuracy : 0.7948
##
##        'Positive' Class : CH
##
```

```r
1-.8312
```

```
## [1] 0.1688
```

```r
svm.poly.pred1 <- predict(svm.poly, oj.test)
confusionMatrix(svm.poly.pred1, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 137  49
##         MM  13  71
##
##                Accuracy : 0.7704
##                  95% CI : (0.7155, 0.8192)
##     No Information Rate : 0.5556
##     P-Value [Acc > NIR] : 1.621e-13
##
##                   Kappa : 0.5206
##  Mcnemar's Test P-Value : 8.789e-06
##
##             Sensitivity : 0.9133
##             Specificity : 0.5917
##          Pos Pred Value : 0.7366
##          Neg Pred Value : 0.8452
##              Prevalence : 0.5556
##          Detection Rate : 0.5074
##    Detection Prevalence : 0.6889
##       Balanced Accuracy : 0.7525
##
##        'Positive' Class : CH
##
```

```r
1-.7704
```

```
## [1] 0.2296
```

The polynomial SVM of degree 2 has a train error of 16.88% and a test error of 22.96%, markedly worse than the linear SVM.

```r
set.seed(1)
tune.svm <- tune(svm, Purchase~., data = oj.train, kernel = "polynomial", degree = 2, ranges = list(cos
summary(tune.svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  7.01
##
## - best performance: 0.165
##
## - Detailed performance results:
```

```
##    cost   error dispersion
## 1  0.01 0.37250 0.04779877
## 2  0.21 0.23125 0.04611655
## 3  0.41 0.19375 0.03547789
## 4  0.61 0.19500 0.03129164
## 5  0.81 0.18750 0.03004626
## 6  1.01 0.19000 0.03322900
## 7  1.21 0.18500 0.03763863
## 8  1.41 0.18125 0.03346329
## 9  1.61 0.18125 0.02960973
## 10 1.81 0.18125 0.02716334
## 11 2.01 0.18250 0.02776389
## 12 2.21 0.18375 0.02949223
## 13 2.41 0.18250 0.03291403
## 14 2.61 0.18375 0.03387579
## 15 2.81 0.18250 0.03496029
## 16 3.01 0.18250 0.03446012
## 17 3.21 0.18000 0.03736085
## 18 3.41 0.18125 0.03596391
## 19 3.61 0.17750 0.03670453
## 20 3.81 0.17750 0.03670453
## 21 4.01 0.17625 0.03557562
## 22 4.21 0.17500 0.03632416
## 23 4.41 0.17625 0.03928617
## 24 4.61 0.17500 0.03818813
## 25 4.81 0.17500 0.04082483
## 26 5.01 0.17375 0.03884174
## 27 5.21 0.17250 0.03717451
## 28 5.41 0.17250 0.03717451
## 29 5.61 0.16875 0.03547789
## 30 5.81 0.17000 0.03446012
## 31 6.01 0.17000 0.03446012
## 32 6.21 0.17125 0.03488573
## 33 6.41 0.16750 0.03782269
## 34 6.61 0.16875 0.03784563
## 35 6.81 0.16625 0.03682259
## 36 7.01 0.16500 0.03622844
## 37 7.21 0.16500 0.03574602
## 38 7.41 0.16500 0.03574602
## 39 7.61 0.16750 0.03129164
## 40 7.81 0.16750 0.03129164
## 41 8.01 0.16875 0.03240906
## 42 8.21 0.16875 0.03240906
## 43 8.41 0.16875 0.03240906
## 44 8.61 0.16875 0.03240906
## 45 8.81 0.16875 0.03240906
## 46 9.01 0.16750 0.03238227
## 47 9.21 0.16750 0.03016160
## 48 9.41 0.16625 0.03230175
## 49 9.61 0.16625 0.03230175
## 50 9.81 0.16625 0.03230175
```

```r
svm.poly<- svm(Purchase ~ ., data = oj.train, kernel = "polynomial", degree = 2, cost = 7.01)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj.train, kernel = "polynomial",
##     degree = 2, cost = 7.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  7.01
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  337
##
##  ( 170 167 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
svm.poly.pred <- predict(svm.poly, oj.train)
confusionMatrix(svm.poly.pred, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 467  82
##         MM  36 215
##
##                Accuracy : 0.8525
##                  95% CI : (0.826, 0.8764)
##     No Information Rate : 0.6288
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6737
##  Mcnemar's Test P-Value : 3.434e-05
##
##             Sensitivity : 0.9284
##             Specificity : 0.7239
##          Pos Pred Value : 0.8506
##          Neg Pred Value : 0.8566
##              Prevalence : 0.6288
##          Detection Rate : 0.5837
##    Detection Prevalence : 0.6863
##       Balanced Accuracy : 0.8262
##
##        'Positive' Class : CH
##
```

```r
1-.8525
```

```
## [1] 0.1475
```

```r
svm.poly.pred1 <- predict(svm.poly, oj.test)
confusionMatrix(svm.poly.pred1, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 136  40
##         MM  14  80
##
##                Accuracy : 0.8
##                  95% CI : (0.7472, 0.846)
##     No Information Rate : 0.5556
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.586
##  Mcnemar's Test P-Value : 0.0006688
##
##             Sensitivity : 0.9067
##             Specificity : 0.6667
##          Pos Pred Value : 0.7727
##          Neg Pred Value : 0.8511
##              Prevalence : 0.5556
##          Detection Rate : 0.5037
##    Detection Prevalence : 0.6519
##       Balanced Accuracy : 0.7867
##
##        'Positive' Class : CH
##
```

```r
1-.8
```

```
## [1] 0.2
```

Using the tuned cost parameter, the train error decreases to 14.75% and the test error to 20%.

(h)

The linear SVM when cost = .01 or the radial SVM with the default settings give the same performance.