```r
library(tree)       # tree(), cv.tree()
library(ISLR)       # OJ dataframe
library(ggplot2)

# question 1

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 ...
```

```r
# factors
OJ$StoreID = factor(OJ$StoreID)
OJ$STORE = factor(OJ$STORE)
OJ$SpecialCH = factor(OJ$SpecialCH)
OJ$SpecialMM = factor(OJ$SpecialMM)

# training/test sets
n = nrow(OJ)
n
```

```
## [1] 1070
```

```r
RNGkind(sample.kind='Rounding')
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
set.seed(1,sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
train = sample(1:n, 800)
test = (1:n)[-train]         # 270 obs

# a) single classification tree
tree.OJ = tree(Purchase ~ ., data=OJ[train,] )
summary(tree.OJ)
```

```
##
## 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
```

```
#
# training error rate 0.165
# most important classifiers: LoyalCH, PriceDiff

# test error rate
y_hat = predict(tree.OJ,newdata=OJ[test,],type="class" ) # gives classification labels
table(y_hat,OJ[test,]$Purchase)
```

```
##
## y_hat  CH  MM
##    CH 147  49
##    MM  12  62
```

```
aux=prop.table(table( y_hat, OJ[test,]$Purchase ))
round(aux,3)
```

```
##
## y_hat    CH    MM
##    CH 0.544 0.181
##    MM 0.044 0.230
```

```
#
# test error  rate 0.2259

# b) cross-validation to find optimal tree size
set.seed(2,sample.kind = 'Rounding')
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

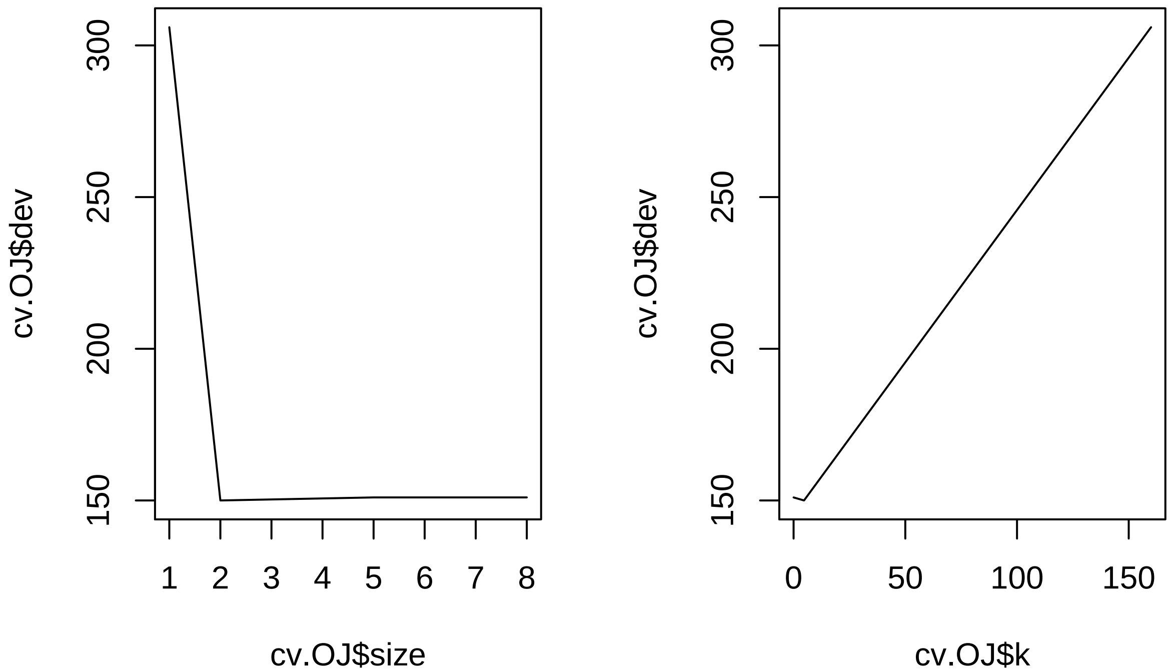
```
cv.OJ = cv.tree(tree.OJ,FUN=prune.misclass) # CV misclass based
cv.OJ$size    #[1] 8 5 2 1
```

```
## [1] 8 5 2 1
```

```
cv.OJ$dev     #[1] 151 151 150 306
```

```
## [1] 151 151 150 306
```

```
#
par(mfrow=c(1,2))
plot(cv.OJ$dev~cv.OJ$size,type='l')
plot(cv.OJ$dev~cv.OJ$k,type='l')
```



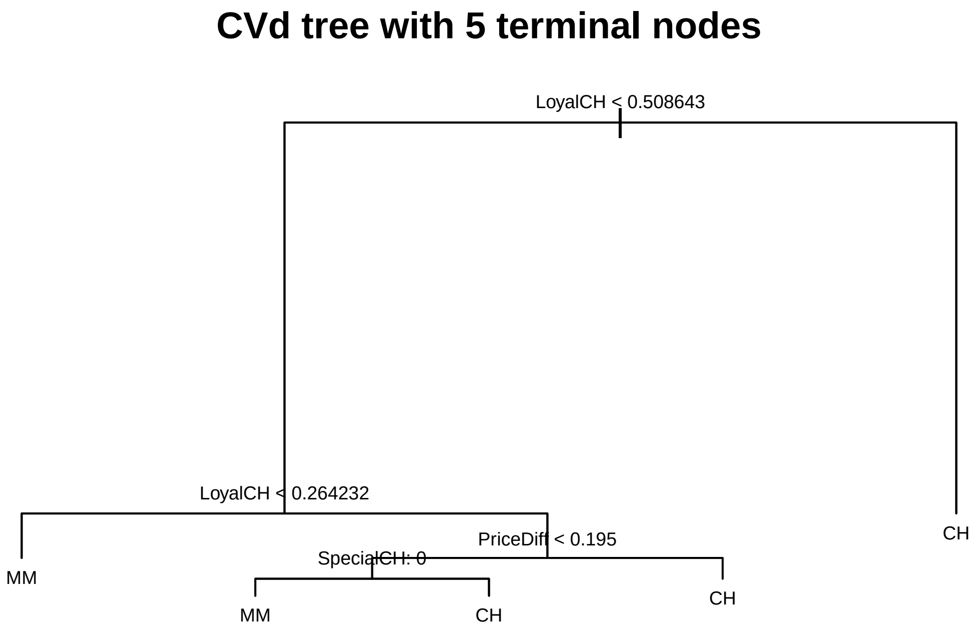

```r
par(mfrow=c(1,1))
#
# best tree with 2 terminal nodes
#
# prune training tree to 2 nodes
prune2 = prune.misclass(tree.OJ, best=2)
summary(prune2)
```

```
##
## Classification tree:
## snip.tree(tree = tree.OJ, nodes = 3:2)
## Variables actually used in tree construction:
## [1] "LoyalCH"
## Number of terminal nodes:  2
## Residual mean deviance:  0.9115 = 727.4 / 798
## Misclassification error rate: 0.1825 = 146 / 800
```

```r
#
# LoyalCH is best classifier
#
# c)  prune tree to 5 nodes

prune5b = prune.misclass(tree.OJ, best=5)
summary(prune5b)
```

```
##
## Classification tree:
## snip.tree(tree = tree.OJ, 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
```

```r
plot(prune5b)
text(prune5b,cex=0.6,pretty=0)
title("CVd tree with 5 terminal nodes")
```




```r
# test error rate
y_hat = predict( prune5b, newdata=OJ[test,], type="class" )
table( y_hat, OJ[test,]$Purchase )
```

```
##
## y_hat  CH  MM
##    CH 147  49
##    MM  12  62
```

```r
#
aux=prop.table(table( y_hat, OJ[test,]$Purchase ))
round(aux,3)
```

```
##
## y_hat    CH    MM
##    CH 0.544 0.181
##    MM 0.044 0.230
```

```r
#
# test error rate [1] 0.226

# d) Random Forest
library(randomForest)
set.seed(1)
rf1 = randomForest(Purchase~.,data = OJ, subset = train, importance=T)
#
ytest = OJ[test,]$Purchase
y_hat = predict(rf1, newdata=OJ[test,], type="class" )
table( y_hat, OJ[test,]$Purchase )
```

```
##
## y_hat  CH  MM
```

```
##    CH 138  33
##    MM  21  78
```

```
aux=prop.table(table( y_hat, OJ[test,]$Purchase ))
round(aux,3)
```

```
##
## y_hat    CH    MM
##    CH 0.511 0.122
##    MM 0.078 0.289
```

```
1-sum(diag(aux))
```

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

```
#
# test error rate is 0.20
#
# e) Boosted Trees

# install.packages('doParallel')
library(doParallel)
cl = makePSOCKcluster(4)
registerDoParallel(cl)

# install.packages("e1071")
# install.packages("caret")

library(e1071)
library(caret)   # train()
#
#
gbmGrid = expand.grid(n.minobsinnode = 10,
                      interaction.depth = c(4,7,9),
                      n.trees = seq(1200,2000,by = 200),
                      shrinkage = c(0.001, 0.01))
gbmGrid
```

```
##    n.minobsinnode interaction.depth n.trees shrinkage
## 1              10                 4    1200     0.001
## 2              10                 7    1200     0.001
## 3              10                 9    1200     0.001
## 4              10                 4    1400     0.001
## 5              10                 7    1400     0.001
## 6              10                 9    1400     0.001
## 7              10                 4    1600     0.001
## 8              10                 7    1600     0.001
## 9              10                 9    1600     0.001
## 10             10                 4    1800     0.001
## 11             10                 7    1800     0.001
## 12             10                 9    1800     0.001
## 13             10                 4    2000     0.001
## 14             10                 7    2000     0.001
## 15             10                 9    2000     0.001
## 16             10                 4    1200     0.010
## 17             10                 7    1200     0.010
```

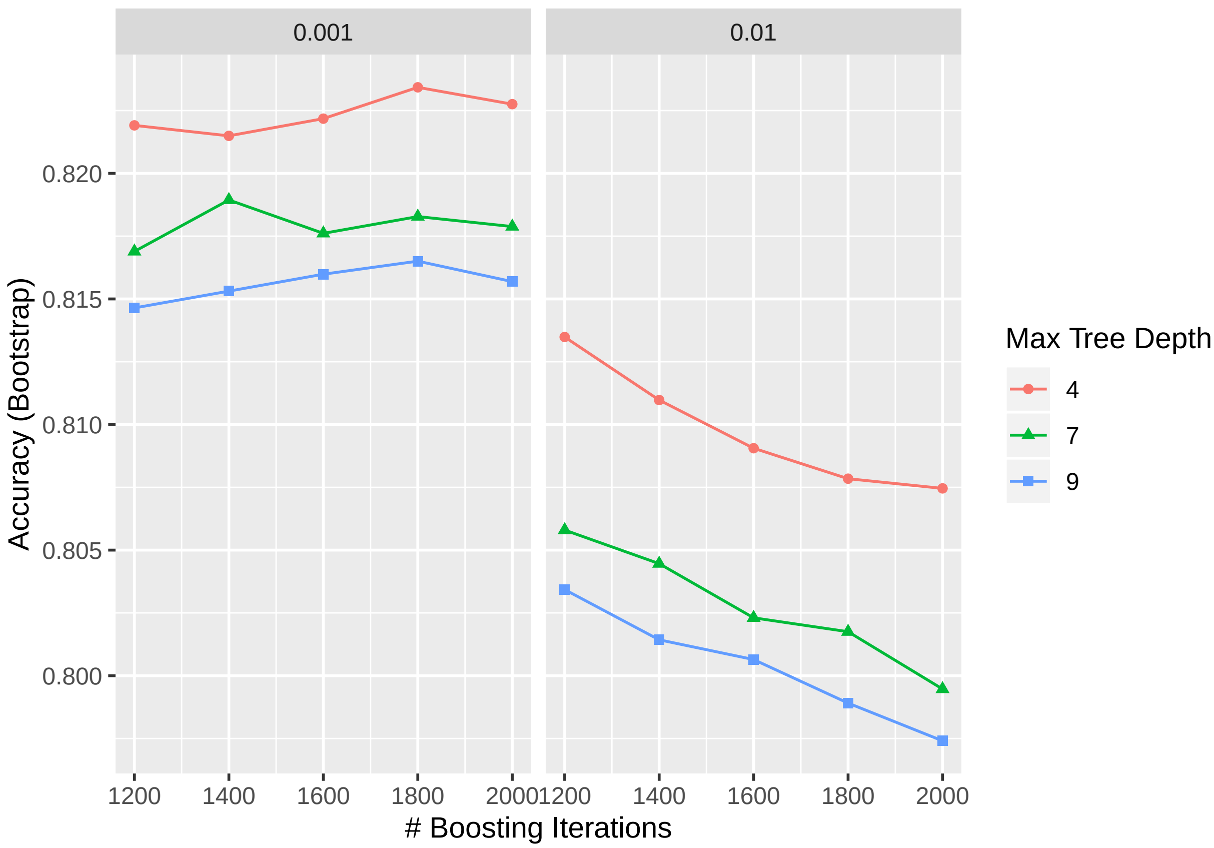
```
## 18              10                   9     1200      0.010
## 19              10                   4     1400      0.010
## 20              10                   7     1400      0.010
## 21              10                   9     1400      0.010
## 22              10                   4     1600      0.010
## 23              10                   7     1600      0.010
## 24              10                   9     1600      0.010
## 25              10                   4     1800      0.010
## 26              10                   7     1800      0.010
## 27              10                   9     1800      0.010
## 28              10                   4     2000      0.010
## 29              10                   7     2000      0.010
## 30              10                   9     2000      0.010
```

```r
ytrain = OJ[train,1]
xtrain = OJ[train,2:18]
set.seed(100)
gbmTune = train(xtrain, ytrain,method = "gbm",tuneGrid = gbmGrid,verbose = F)

gbmTune
```

```
## Stochastic Gradient Boosting
##
## 800 samples
##  17 predictor
##   2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 800, 800, 800, 800, 800, 800, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.trees  Accuracy   Kappa
##   0.001      4                  1200     0.8219105  0.6197506
##   0.001      4                  1400     0.8214937  0.6196363
##   0.001      4                  1600     0.8221785  0.6217301
##   0.001      4                  1800     0.8234257  0.6248542
##   0.001      4                  2000     0.8227548  0.6235989
##   0.001      7                  1200     0.8168915  0.6088627
##   0.001      7                  1400     0.8189407  0.6140274
##   0.001      7                  1600     0.8176114  0.6120073
##   0.001      7                  1800     0.8182804  0.6138473
##   0.001      7                  2000     0.8178823  0.6131856
##   0.001      9                  1200     0.8146422  0.6038275
##   0.001      9                  1400     0.8153115  0.6064219
##   0.001      9                  1600     0.8159850  0.6084310
##   0.001      9                  1800     0.8165043  0.6098720
##   0.001      9                  2000     0.8156910  0.6084564
##   0.010      4                  1200     0.8134845  0.6050466
##   0.010      4                  1400     0.8109752  0.5997210
##   0.010      4                  1600     0.8090566  0.5958897
##   0.010      4                  1800     0.8078435  0.5933634
##   0.010      4                  2000     0.8074577  0.5923253
##   0.010      7                  1200     0.8057980  0.5891099
##   0.010      7                  1400     0.8044611  0.5862807
```

```
##   0.010      7                     1600      0.8023049  0.5816767
##   0.010      7                     1800      0.8017511  0.5801109
##   0.010      7                     2000      0.7994695  0.5754616
##   0.010      9                     1200      0.8034277  0.5845886
##   0.010      9                     1400      0.8014290  0.5799804
##   0.010      9                     1600      0.8006433  0.5784710
##   0.010      9                     1800      0.7989097  0.5745151
##   0.010      9                     2000      0.7974069  0.5716254
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 1800, interaction.depth =
##  4, shrinkage = 0.001 and n.minobsinnode = 10.
```

```r
# This report shows the Accuracy rates for each instance in the Grid
# it also shows the parameter values for the highest train Accuracy rate

# display training Accuracy rates
ggplot(gbmTune)
```




```r
ytest = OJ[-train,1]
xtest = OJ[-train,2:18]

# test error rate
y_hat = predict(gbmTune,newdata=xtest, type="raw" )
table(y_hat,ytest)
```

```
##      ytest
```

```
## y_hat  CH  MM
##    CH 141  28
##    MM  18  83
```

```r
aux=prop.table(table(y_hat,ytest))
round(aux,3)
```

```
##      ytest
## y_hat    CH    MM
##    CH 0.522 0.104
##    MM 0.067 0.307
```

```r
1-sum(diag(aux))
```

```
## [1] 0.1703704
```

```r
# test error rate is 0.17
#
#
#
# question 2

# a)
#
population = 234564000
d0 = read.csv('cereal.csv')
str(d0)
```

```
## 'data.frame':    1250 obs. of  3 variables:
##  $ Group    : int  1 2 2 4 2 2 2 3 3 4 ...
##  $ Spend    : num  14.77 8.15 8 9.31 12.09 ...
##  $ Breakfast: int  4 4 2 1 4 4 4 4 3 3 ...
```

```r
n = nrow(d0)
table(d0$Group)
```

```
##
##   1   2   3   4
## 269 484 241 256
```

```r
x = 269
prop.test(x,n)$conf.int
```

```
## [1] 0.1929252 0.2392505
## attr(,"conf.level")
## [1] 0.95
```

```r
population*prop.test(x,n)$conf.int
```

```
## [1] 45253302 56119555
## attr(,"conf.level")
## [1] 0.95
```

```r
#
# the number of Americans who are concerned about eating healthy foods is
# somewhere between 45253302 and 56119555.
#
# b)
head(d0)
```

```
##   Group Spend Breakfast
## 1     1 14.77         4
## 2     2  8.15         4
## 3     2  8.00         2
## 4     4  9.31         1
## 5     2 12.09         4
## 6     2  7.13         4
```

```
d0$Group[d0$Group != 1] = 0
head(d0)
```

```
##   Group Spend Breakfast
## 1     1 14.77         4
## 2     0  8.15         4
## 3     0  8.00         2
## 4     0  9.31         1
## 5     0 12.09         4
## 6     0  7.13         4
```

```
table(d0$Group)
```

```
##
##   0   1
## 981 269
```

```
#
# Average Spending by Group
tapply(d0$Spend,d0$Group,mean)
```

```
##        0        1
## 10.59541 11.57926
```

```
# test Ho: average Spending Group 1 = average Spending Group 0 (all others)
Group1 = d0$Spend[d0$Group == 1]
Group0 = d0$Spend[d0$Group == 0]
t.test(Group1,Group0,'greater')
```

```
##
##  Welch Two Sample t-test
##
## data:  Group1 and Group0
## t = 4.146, df = 643.95, p-value = 1.918e-05
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.5929601       Inf
## sample estimates:
## mean of x mean of y
##  11.57926  10.59541
```

```
#
# based on p-value reject Ho in favor of Ha: true difference in means is greater than 0
# conclude that Americans concerned about eating healthy foods spend on average, more
# than other Americans
#
#
#

# question 3
```

```
#
## ID Prediction
## 1  1156.0714
## 2   141.7590
## 3   709.4666
## 4   746.4439
## 5   141.7590
## 6   141.7590
## 7   510.0157
## 8   141.7590
## 9   510.0157
```