EECS E6690 hw3

Chong Hu ch3467

Sep 22, 2019

P1

(a)

$$\begin{aligned}\mathbb{P}[Y = \text{true}] &= \frac{\exp(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2)}{1 + \exp(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2)} \\ &= \frac{\exp(-6 + 0.05 * 40 + 1 * 3.5)}{1 + \exp(-6 + 0.05 * 40 + 1 * 3.5)} \\ &= 0.3775407\end{aligned}$$

Hence, $\mathbb{P}[Y = \text{true}] = 0.3775407$.

(b)

$$\begin{aligned}\mathbb{P}[Y = \text{true}] = \frac{\exp(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2)}{1 + \exp(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2)} &= 0.5 \\ \exp(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2) &= 1 \\ -6 + 0.05 * x_1 + 1 * 3.5 &= 0 \\ x_1 &= 50\end{aligned}$$

The student in part (a) need to study 50 hours to have a 50% chance of getting an A in the class.

P2

$$\begin{aligned}f_{\text{Yes}}(x) &= \frac{1}{\sqrt{2\pi\hat{\sigma}^2}} \exp -\frac{(x - \mu_{\text{Yes}})^2}{2\hat{\sigma}^2} \\ f_{\text{No}}(x) &= \frac{1}{\sqrt{2\pi\hat{\sigma}^2}} \exp -\frac{(x - \mu_{\text{No}})^2}{2\hat{\sigma}^2} \\ \mathbb{P}[Y = \text{Yes} \mid X = 4] &= \frac{\pi_{\text{Yes}} f_{\text{Yes}}(x)}{\pi_{\text{Yes}} f_{\text{Yes}}(x) + \pi_{\text{No}} f_{\text{No}}(x)} \\ &= \frac{0.8 * 0.04032845}{0.8 * 0.04032845 + 0.2 * 0.05324133} \\ &= 0.7518524\end{aligned}$$

P3

$$\begin{aligned}\text{likelihood} &= \prod_{i=1}^{n} \left(\frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)}\right)^{y_i} \left(\frac{1}{1 + \exp(\beta_0 + \beta_1 x_i)}\right)^{1 - y_i} \\ l(\beta_0, \beta_1) &= \sum_{i=1}^{n} y_i(\beta_0 + \beta_1 x_i) - \ln\big(1 + \exp(\beta_0 + \beta_1 x_i)\big)\end{aligned}$$

To maximize the likelihood, we need to set

$$\frac{\partial l(\beta_0,\beta_1)}{\partial\beta_0} = \sum_{i=1}^{n}\left(y_i - \frac{\exp\left(\hat\beta_0+\hat\beta_1 x_i\right)}{1+\exp\left(\hat\beta_0+\hat\beta_1 x_i\right)}\right) = 0$$
$$\frac{\partial l(\beta_0,\beta_1)}{\partial\beta_0} = \sum_{i=1}^{n} x_i\left(y_i - \frac{\exp\left(\hat\beta_0+\hat\beta_1 x_i\right)}{1+\exp\left(\hat\beta_0+\hat\beta_1 x_i\right)}\right) = 0$$

Using the Newton–Raphson algorithm, I write all variables in matrix form. $\mathbf{W}$ is $n \times n$ diagonal matrix of weights with ith diagonal element $p(x;\beta^{\text{old}})(1-p(x;\beta^{\text{old}}))$.

$$\mathbf{z} = \mathbf{X}\beta^{\text{old}} + \mathbf{W}^{-1}(\mathbf{y}-\mathbf{p})$$
$$\beta^{\text{new}} = (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{z}$$

Then we start with $\beta_0 = 0$ and $\beta_1 = 0$ and perform 10 iterations.

```
library(matlib)
```

```
x = c(0.0, 0.2, 0.4, 0.6, 0.8 , 1.0)
X = matrix(cbind(1, x), ncol = 2)
Y = matrix(c(0, 0, 0, 1, 0, 1), ncol = 1)
beta_old = matrix(c(0, 0), ncol = 1)
beta_new = beta_old
for(i in 1:10){
  p = exp(X %*% beta_old) / (1 + exp(X %*% beta_old))
  w = as.vector(p * (1 - p))
  W = diag(w)
  z = X %*% beta_old + inv(W) %*% (Y - p)
  beta_new = inv(t(X)  %*% W %*% X) %*% t(X) %*% W %*% z
  beta_old = beta_new
}
beta_new
```

```
##           [,1]
## [1,] -4.097970
## [2,]  5.723309
```

After 10 iterations, we can get $\hat\beta_0 =$ -4.09797 and $\hat\beta_1 =$ 5.7233085.

P4

$$\begin{aligned}\text{Cov}[\mathbf{Y}] &= \text{Cov}[\mathbf{A}\mathbf{X}]\\ &= \mathbf{A}\text{Cov}[\mathbf{X}]\mathbf{A}^T\\ &= \mathbf{A}\mathbf{\Sigma}\mathbf{A}^T\end{aligned}$$

Then, since cov[$\mathbf{Y}$] is an identity matrix, we have

$$\mathbf{A}\mathbf{\Sigma}\mathbf{A}^T = \mathbf{I}$$

Since $\mathbf{\Sigma}$ is symmetric, we can get

$$\mathbf{\Sigma} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$$

where $\mathbf{Q}$ is an orthogonal matrix whose columns are the eigenvectors of $\mathbf{\Sigma}$, and Λ is a diagonal matrix whose entries are the eigenvalues of $\mathbf{\Sigma}$.\ And,

$$\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$$

Then,

$$\begin{aligned}
\mathbf{AQ\Lambda Q}^T\mathbf{A}^T &= \mathbf{I} \\
(\mathbf{AQ})\mathbf{\Lambda}(\mathbf{AQ})^T &= \mathbf{I} \\
\mathbf{AQ} &= (\mathbf{\Lambda}^{-1})^{\frac{1}{2}} \\
\mathbf{A} &= (\mathbf{\Lambda}^{-1})^{\frac{1}{2}}\mathbf{Q}^{-1}
\end{aligned}$$

$$\mathbf{\Lambda} = \begin{bmatrix} \sigma_1^2/2 - (4*\rho^2\sigma_1^2\sigma_2^2 + \sigma_1^4 - 2\sigma_1^2\sigma_2^2 + \sigma_2^4)^{1/2}/2 + \sigma_2^2/2 & 0 \\ 0 & (4\rho^2\sigma_1^2\sigma_2^2 + \sigma_1^4 - 2\sigma_1^2\sigma_2^2 + \sigma_2^4)^{1/2}/2 + \sigma_1^2/2 + \sigma_2^2/2 \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} (\sigma_1^2/2 - (4\rho^2\sigma_1^2\sigma_2^2 + \sigma_1^4 - 2\sigma_1^2\sigma_2^2 + \sigma_2^4)^{1/2}/2 + \sigma_2^2/2)/(\rho\sigma_1\sigma_2) - \sigma_2/(\rho\sigma_1) & ((4\rho^2\sigma_1^2\sigma_2^2 + \sigma_1^4 - 2\sigma_1^2\sigma_2^2 + \sigma_2^4)^{1/2}/2 + \sigma_1^2/2 + \sigma_2^2/2)/(\rho\sigma_1\sigma_2) - \sigma_2/(\rho\sigma_1) \\ 1 & 1 \end{bmatrix}$$

Therefore,

$$\mathbf{A} = \begin{bmatrix} -\frac{(2^{1/2}\rho\sigma_1\sigma_2(1/(\sigma_1^2-(4*\rho^2*\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2}+\sigma_2^2))^{1/2})}{(4*\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2}} & \frac{(2^{1/2}*(1/(\sigma_1^2-(4\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^(1/2)+\sigma_2^2))^{1/2}((4\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2}+\sigma_1^2-\sigma_2^2))}{(2(4\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2})} \\ \frac{2^{1/2}\rho\sigma_1\sigma_2(1/((4\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2}+\sigma_1^2+\sigma_2^2))^{1/2})}{(4\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2}} & \frac{(2^{1/2}*(1/((4\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2}+\sigma_1^2+\sigma_2^2))^{1/2}((4\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2}-\sigma_1^2+\sigma_2^2))}{(2(4\rho^2\sigma_1^2\sigma_2^2+\sigma_1^4-2\sigma_1^2\sigma_2^2+\sigma_2^4)^{1/2})} \end{bmatrix}$$

P5

Since

$$\hat{\mu}_k = \frac{1}{n_k}\sum_{i:\ y_i=k} x_i \quad \text{and} \quad \hat{\sigma}_k^2 = \frac{1}{n_k-1}\sum_{i:\ y_i=k}(x_i-\hat{\mu}_k)^2$$

and under the Gaussian assumption, the variance of each population is σ^2, we can easily get

$$\mathbb{E}[\hat{\sigma}_k^2] = \sigma^2$$

Hence,

$$\begin{aligned}
\mathbb{E}[\hat{\sigma}^2] = \mathbb{E}\Big[\sum_{k=1}^K \alpha_k\hat{\sigma}_k^2\Big] &= \sum_{k=1}^K \alpha_k \mathbb{E}\big[\hat{\sigma}_k^2\big] \\
&= \sigma^2\sum_{k=1}^K \alpha_k = \sigma^2
\end{aligned}$$

Also, under the Gaussian assumptions, we have

$$\begin{aligned}
\frac{(n_k-1)\hat{\sigma}_k^2}{\sigma^2} &\sim \mathcal{X}^2_{n_k-1} \\
\text{Var}\left[\frac{(n_k-1)\hat{\sigma}_k^2}{\sigma^2}\right] &= \text{Var}\big[\mathcal{X}_{n_k-1}\big] \\
\frac{(n_k-1)^2}{\sigma^4}\text{Var}[\hat{\sigma}_k^2] &= 2(n_k-1) \\
\text{Var}[\hat{\sigma}_k^2] &= \frac{2\sigma^4}{n_k-1}
\end{aligned}$$

Then,

$$\begin{aligned}
\text{Var}\big[\hat{\sigma}^2\big] &= \text{Var}\left[\sum_{k=1}^{K}\alpha_k\hat{\sigma}_k^2\right] \\
&= \sum_{k=1}^{K}\alpha_k^2\text{Var}\big[\hat{\sigma}_k^2\big] \\
&= \sum_{k=1}^{K}\alpha_k^2\frac{2\sigma^4}{n_k-1} \\
&= 2\sigma^4\sum_{k=1}^{K}\frac{\alpha_k^2}{n_k-1} \\
&= \frac{2\sigma^4}{n-K}\sum_{k=1}^{K}\alpha_k^2\frac{n-K}{n_k-1} \\
\text{where } &\sum_{k=1}^{K}\alpha_k = 1 \\
\text{and } &\sum_{k=1}^{K}\frac{n_k-1}{n-K} = 1 \\
\text{Based on Jensen's } &\text{inequality, we can get} \\
&\geq \frac{2\sigma^4}{n-K}
\end{aligned}$$

where equality happens if and only if $\alpha_k = (n_k-1)/(n-K)$. Therefore, the variance of $\hat{\sigma}^2$ is minimized when $\alpha_k = (n_k-1)/(n-K)$.

P6

Through majority vote approach, six out of ten estimates are this class is Red. Therefore, the final classification is Red. Through average probability, the average estimate is 0.45, therefore the final classification is Green.

P7

```
library(ISLR)
library(tree)
```

```
attach(OJ)
set.seed(1000)
```

(a)

```
train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

(b)

```
mod.tr <- tree(Purchase ~ ., data = OJ.train)
summary(mod.tr)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"     "PriceDiff"   "SalePriceMM"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7486 = 592.9 / 792
## Misclassification error rate: 0.16 = 128 / 800
```

Training error rate for the tree is 0.16. The tree only use three variables `LoyalCH`, `PriceDiff` and `SalePriceMM`. There are 8 terminal nodes in the tree.

(c)

```
mod.tr
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1066.00 CH ( 0.61500 0.38500 )
##    2) LoyalCH < 0.5036 353  422.60 MM ( 0.28612 0.71388 )
##      4) LoyalCH < 0.276142 170  131.00 MM ( 0.12941 0.87059 )
##        8) LoyalCH < 0.035047 57   10.07 MM ( 0.01754 0.98246 ) *
##        9) LoyalCH > 0.035047 113  108.50 MM ( 0.18584 0.81416 ) *
##      5) LoyalCH > 0.276142 183  250.30 MM ( 0.43169 0.56831 )
##       10) PriceDiff < 0.05 78   79.16 MM ( 0.20513 0.79487 ) *
##       11) PriceDiff > 0.05 105  141.30 CH ( 0.60000 0.40000 ) *
##    3) LoyalCH > 0.5036 447  337.30 CH ( 0.87472 0.12528 )
##      6) LoyalCH < 0.764572 187  206.40 CH ( 0.75936 0.24064 )
##       12) SalePriceMM < 2.125 120  156.60 CH ( 0.64167 0.35833 )
##         24) PriceDiff < -0.35 16   17.99 MM ( 0.25000 0.75000 ) *
##         25) PriceDiff > -0.35 104  126.70 CH ( 0.70192 0.29808 ) *
##       13) SalePriceMM > 2.125 67   17.99 CH ( 0.97015 0.02985 ) *
##      7) LoyalCH > 0.764572 260   91.11 CH ( 0.95769 0.04231 ) *
```

Let's interpret the terminal node with "10)". The splitting variable of this node is `PriceDiff`. `PriceDiff < 0.05` will be predicted as `MM` for the response `Purchase`. There are 78 points in total below this node. The deviation of those points is 79.16. 0.20513% points have `CH` as value for `Purchase` and 0.79487% points have `MM` as value for `purchase`. The * in the end means this is a terminal node.

(d)

```r
plot(mod.tr)
text(mod.tr, pretty = 5)
```

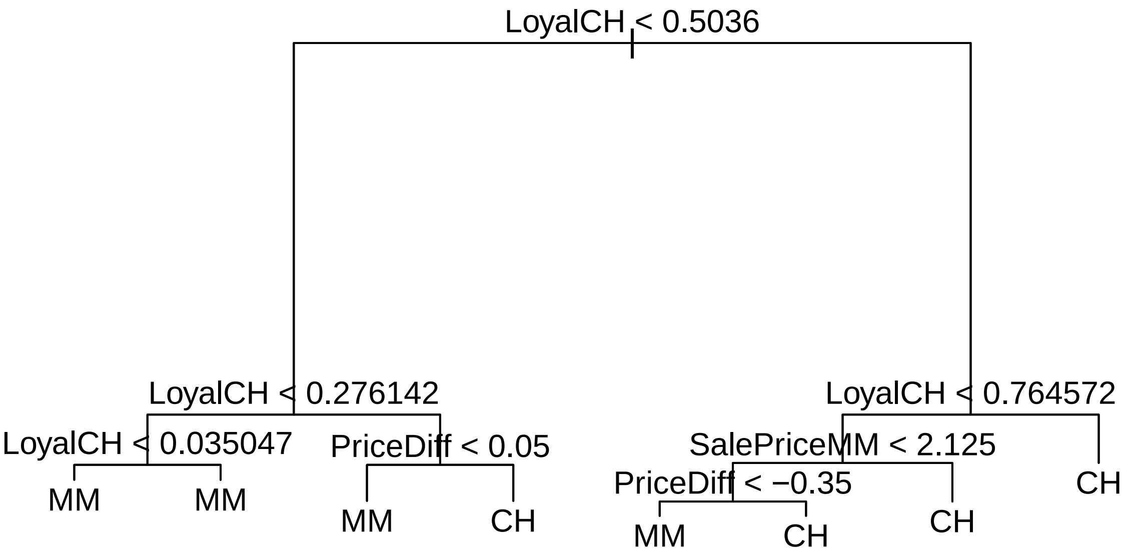

Figure 1: Plot of the tree

Three split data points into two sets based on `LoyalCH < 0.5036` at the root.Then, the subtrees split data points into different leaves based on different conditions. Among those conditions, we can clearly see that variable `LoyalCH` is used very often. In the end, we get eight terminal nodes to predict response `Purchase` as `CH` or `MM`.

(e)

```r
mod.pred = predict(mod.tr, OJ.test, type = "class")
err_rate = sum(OJ.test$Purchase != mod.pred) / dim(OJ.test)[1]
print(sprintf("err rate: %.2f%% ",  err_rate*100))
```

```
## [1] "err rate: 18.15% "
```

```r
table(OJ.test$Purchase, mod.pred)
```

```
##     mod.pred
##       CH  MM
##   CH 150  11
##   MM  38  71
```

Error rate on test set is 18.15%.

(f)

```
tree.cv = cv.tree(mod.tr, FUN = prune.tree)
tree.cv
```

```
## $size
## [1] 8 7 6 5 4 3 2 1
##
## $dev
## [1]  663.1416  670.8785  669.0620  715.5108  737.4116  777.8179  770.9825
## [8] 1068.1253
##
## $k
## [1]      -Inf  11.87503  12.41171  29.77434  31.80546  39.82936  41.38321
## [8] 306.37571
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

The optimal size is 8 based on deviation.

(g)

```
plot(tree.cv$size, tree.cv$dev, col = "blue", pch = 3,
     type = "b", xlab = "Tree Size", ylab = "cross-validated classification error")
```

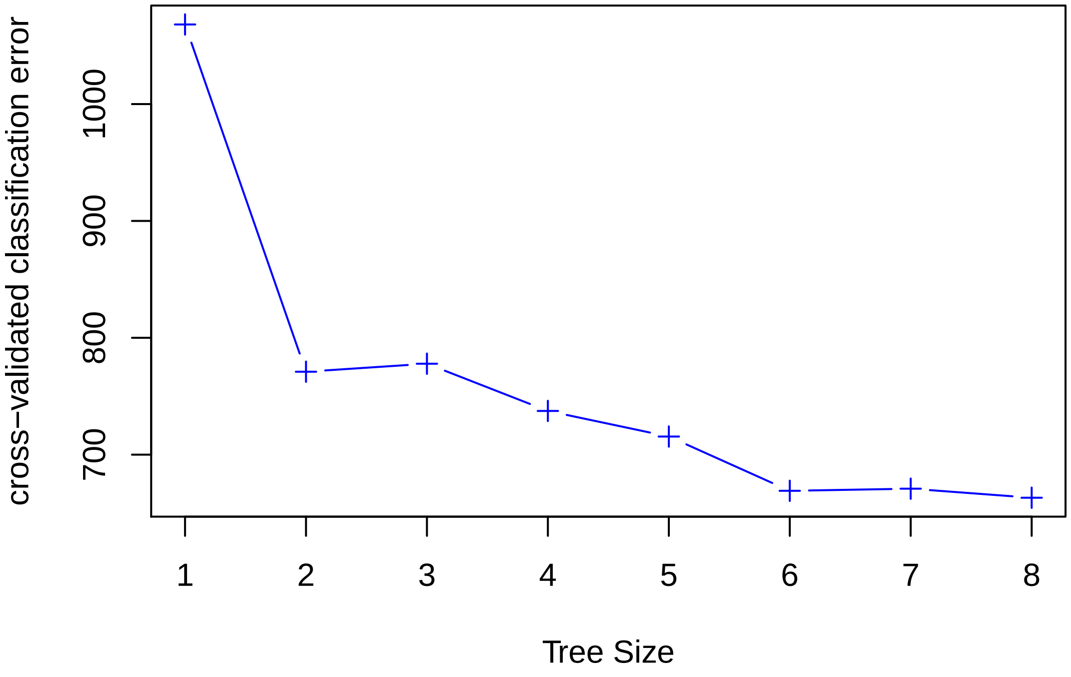

Figure 2: cross-validated classification error vs. Tree size

(h)

When the tree size is 8, cross-validation error is the lowest.

(i)

```r
tree.pruned = prune.tree(mod.tr, best = 8)
tree.pruned
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1066.00 CH ( 0.61500 0.38500 )
##    2) LoyalCH < 0.5036 353  422.60 MM ( 0.28612 0.71388 )
##      4) LoyalCH < 0.276142 170  131.00 MM ( 0.12941 0.87059 )
##        8) LoyalCH < 0.035047 57   10.07 MM ( 0.01754 0.98246 ) *
##        9) LoyalCH > 0.035047 113  108.50 MM ( 0.18584 0.81416 ) *
##      5) LoyalCH > 0.276142 183  250.30 MM ( 0.43169 0.56831 )
##       10) PriceDiff < 0.05 78   79.16 MM ( 0.20513 0.79487 ) *
##       11) PriceDiff > 0.05 105  141.30 CH ( 0.60000 0.40000 ) *
##    3) LoyalCH > 0.5036 447  337.30 CH ( 0.87472 0.12528 )
##      6) LoyalCH < 0.764572 187  206.40 CH ( 0.75936 0.24064 )
##       12) SalePriceMM < 2.125 120  156.60 CH ( 0.64167 0.35833 )
##         24) PriceDiff < -0.35 16   17.99 MM ( 0.25000 0.75000 ) *
##         25) PriceDiff > -0.35 104  126.70 CH ( 0.70192 0.29808 ) *
##       13) SalePriceMM > 2.125 67   17.99 CH ( 0.97015 0.02985 ) *
##      7) LoyalCH > 0.764572 260   91.11 CH ( 0.95769 0.04231 ) *
```

```r
#  create a pruned tree with five terminal nodes.
tree.pruned = prune.tree(mod.tr, best = 5)
tree.pruned
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1066.00 CH ( 0.61500 0.38500 )
##    2) LoyalCH < 0.5036 353  422.60 MM ( 0.28612 0.71388 )
##      4) LoyalCH < 0.276142 170  131.00 MM ( 0.12941 0.87059 ) *
##      5) LoyalCH > 0.276142 183  250.30 MM ( 0.43169 0.56831 ) *
##    3) LoyalCH > 0.5036 447  337.30 CH ( 0.87472 0.12528 )
##      6) LoyalCH < 0.764572 187  206.40 CH ( 0.75936 0.24064 )
##       12) SalePriceMM < 2.125 120  156.60 CH ( 0.64167 0.35833 ) *
##       13) SalePriceMM > 2.125 67   17.99 CH ( 0.97015 0.02985 ) *
##      7) LoyalCH > 0.764572 260   91.11 CH ( 0.95769 0.04231 ) *
```

(j)

```r
summary(tree.pruned)
```

```
##
## Classification tree:
## snip.tree(tree = mod.tr, nodes = c(12L, 4L, 5L))
## Variables actually used in tree construction:
## [1] "LoyalCH"     "SalePriceMM"
## Number of terminal nodes:  5
## Residual mean deviance:  0.8138 = 646.9 / 795
## Misclassification error rate: 0.1962 = 157 / 800
```

The error rate of pruned tree on training set is 19.62% while the error rate of unpruned tree is 16%. After pruning, the error rate on training set is becoming higher. This is because we are removing some nodes in the tree and reducing the overfitting on trainning set.

(k)

```
pruned.pred = predict(tree.pruned, OJ.test, type = "class")
err_rate = sum(OJ.test$Purchase != pruned.pred) / dim(OJ.test)[1]
print(sprintf("err rate: %.2f%% ",  err_rate*100))
```

```
## [1] "err rate: 21.11% "
```

The error rate of pruned tree on training set is 21.11% while the error rate of unpruned tree is 18.15%. In this scenario, the error rate of pruned tree is higher.