# Homework 3

Yi Yang (yy3089)

November 2, 2021

## 1 P1

### 1.1 a

$Y = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2$. The possibility that $\mathrm{P}(Y = \text{true})$ is:

$$\begin{aligned}\mathrm{P}(Y = \text{true}) &= \frac{e^{(\hat{\beta}_0+\hat{\beta}_1 X_1+\hat{\beta}_2 X_2)}}{1+e^{(\hat{\beta}_0+\hat{\beta}_1 X_1+\hat{\beta}_2 X_2)}} \\ &= \frac{e^{(-6+0.05*40+1*3.5)}}{1+e^{(-6+0.05*40+1*3.5)}} \\ &= 0.378\end{aligned} \tag{1.1}$$

### 1.2 b

We can write the equation that:

$$\begin{aligned}\mathrm{P}(Y = \text{true}) &= \frac{e^{(-6+0.05*X_2+1*3.5)}}{1+e^{(-6+0.05*X_2+1*3.5)}} \\ &= 0.5\end{aligned} \tag{1.2}$$

## 2 P2

$$
\begin{aligned}
\mathrm{P}(Y=\mathrm{div} \mid X=4) &= \frac{\mathrm{P}(Y=\mathrm{div}, X=4)}{\mathrm{P}(X=4)} \\
&= \frac{\mathrm{P}(X=4 \mid Y=\mathrm{div})\,\mathrm{P}(Y=\mathrm{div})}{\mathrm{P}(X=4)} \\
&= \frac{\mathrm{P}(\mathrm{div}) * p(4 \mid \mathrm{div})}{\mathrm{P}(\mathrm{div}) * p(4 \mid \mathrm{div}) + \mathrm{P}(\text{not-div}) * p(4 \mid \text{not-div})} \\
&= \frac{0.8 * \frac{1}{\sqrt{2\pi\hat{\sigma}^2}} e^{-\frac{(4-\mu_{\mathrm{div}})^2}{2\hat{\sigma}^2}}}{0.8 * \frac{1}{\sqrt{2\pi\hat{\sigma}^2}} e^{-\frac{(4-\mu_{\mathrm{div}})^2}{2\hat{\sigma}^2}} + 0.2 * \frac{1}{\sqrt{2\pi\hat{\sigma}^2}} e^{-\frac{(4-\mu_{\text{not-div}})^2}{2\hat{\sigma}^2}}} \\
&= \frac{0.8 * 0.04032845}{0.8 * 0.04032845 + 0.2 * 0.05324133} \\
&= 0.752
\end{aligned}
\tag{2.1}
$$

## 3 P3

$P(Y = \text{true}) = \frac{e^{(\beta_0+\beta_1 x_i)}}{1+e^{(\beta_0+\beta_1 x_i)}}$, $P(Y = \text{false}) = \frac{1}{1+e^{(\beta_0+\beta_1 x_i)}}$. Assume that $k_i = 1$ if $y_i$ = true, else $k_i = 0$

$$\mathscr{L}(Y \mid \beta) = \prod_{i=1}^{6} \left( \frac{e^{(\beta_0+\beta_1 x_i)}}{1+e^{(\beta_0+\beta_1 x_i)}} \right)^{k_i} \left( \frac{1}{1+e^{(\beta_0+\beta_1 x_i)}} \right)^{1-k_i} \tag{3.1}$$

To find the optimal value of likelihood function, the question is equal to find the optimal value for:

$$\ln(\mathscr{L}(Y \mid \beta)) = \sum_{i=1}^{6} k_i(\beta_0 + \beta_1 x_i) - \ln\left(1+e^{(\beta_0+\beta_1 x_i)}\right) \tag{3.2}$$

Which is equal to solve:

$$\begin{aligned} \frac{\partial l(\beta_0, \beta_1)}{\partial \beta_0} &= \sum_{i=1}^{n} \left( k_i - \frac{e^{(\beta_0+\hat{\beta}_1 x_i)}}{1+e^{(\beta_0+\beta_1 x_i)}} \right) = 0 \\ \frac{\partial l(\beta_0, \beta_1)}{\partial \beta_1} &= \sum_{i=1}^{n} \left( x_i k_i - \frac{x_i e^{(\beta_0+\beta_1 x_i)}}{1+e^{(\beta_0+\beta_1 x_i)}} \right) = 0 \end{aligned} \tag{3.3}$$

According to Newton-Raphson Algorithm, s single Newton Update is:

$$\beta^{\text{new}} = \beta^{\text{old}} - \left( \frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta^T} \right)^{-1} \frac{\partial \ell(\beta)}{\partial \beta} \tag{3.4}$$

The matrix form of this problem is:

$$\begin{aligned} \mathbf{z} &= \mathbf{X}\beta^{\text{old}} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p}) \\ \beta^{\text{new}} &= \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{z} \end{aligned} \tag{3.5}$$

```r
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 = matrix(c(0, 0), ncol = 1)

cat("Initial Beta values: ", "beta0 = ", beta[1], ",",
    " beta1 = ", beta[2], "\n")
# Initial Beta values:  beta0 =  0 ,  beta1 =  0

maxIter = 10

for(i in 1:maxIter){
  p = exp(X %*% beta) / (1 + exp(X %*% beta))
  w = as.vector(p * (1 - p))
  W = diag(w)
  z = X %*% beta + inv(W) %*% (Y - p)
  beta = inv(t(X) %*% W %*% X) %*% t(X) %*% W %*% z
```

```
  cat("Beta␣values␣at␣iter␣#:␣", i, "beta0␣=␣", beta[1], ",",
      "␣beta1␣=␣", beta[2], "\n")
}
# Beta values at iter #:  1 beta0 =  -2.380952 ,  beta1 =  3.428571
# Beta values at iter #:  2 beta0 =  -3.522775 ,  beta1 =  4.966947
# Beta values at iter #:  3 beta0 =  -4.022333 ,  beta1 =  5.624766
# Beta values at iter #:  4 beta0 =  -4.096585 ,  beta1 =  5.721513
# Beta values at iter #:  5 beta0 =  -4.09797 ,  beta1 =  5.723308
# Beta values at iter #:  6 beta0 =  -4.09797 ,  beta1 =  5.723309
# Beta values at iter #:  7 beta0 =  -4.09797 ,  beta1 =  5.723309
# Beta values at iter #:  8 beta0 =  -4.09797 ,  beta1 =  5.723309
# Beta values at iter #:  9 beta0 =  -4.09797 ,  beta1 =  5.723309
# Beta values at iter #:  10 beta0 =  -4.09797 ,  beta1 =  5.723309
```

## 4 P4

Cov[**Y**] can be written as:

$$\begin{aligned}
\mathrm{Cov}[\mathbf{Y}] &= \mathrm{Cov}[\mathbf{AX}] \\
&= \mathrm{E}[(\mathbf{AX} - \mathrm{E}[\mathbf{AX}])(\mathbf{AX} - \mathrm{E}[\mathbf{AX}])^T] \\
&= \mathbf{A}\mathrm{E}[(\mathbf{X} - \mathrm{E}[\mathbf{X}])(\mathbf{X} - \mathrm{E}[\mathbf{X}])^T]\mathbf{A}^T \\
&= \mathbf{A}\mathrm{Cov}[\mathbf{X}]\mathbf{A}^T \\
&= \mathbf{A}\Sigma\mathbf{A}^T
\end{aligned} \tag{4.1}$$

To get the eigenvalues of the matrix, we need to compute that:

$$\begin{aligned}
|\lambda E - \Sigma| &= (\lambda - \sigma_1^2)(\lambda - \sigma_2^2) - \rho^2\sigma_1^2\sigma_2^2 \\
&= \lambda^2 - (\sigma_1^2 + \sigma_2^2)\lambda + (1 - \rho^2)(\sigma_1^2 + \sigma_2^2) \\
\lambda &= \frac{\sigma_1^2 + \sigma_2^2 \pm \sqrt{(\sigma_1^2 + \sigma_2^2)^2 - 4(1 - \rho^2)\sigma_1^2\sigma_2^2}}{2}
\end{aligned} \tag{4.2}$$

Thus, matrix $\Sigma$ and $\mathbf{A}\Sigma\mathbf{A}^T$ can be written as:

$$\begin{aligned}
\Sigma &= V\mathrm{diag}(\lambda)V^T \\
\mathbf{A}\Sigma\mathbf{A}^T &= \mathbf{A}V\mathrm{diag}(\lambda)V^T\mathbf{A}^T \\
&= \mathbf{A}V\mathrm{diag}(\lambda)^{1/2}((\lambda)^{1/2})^T\mathbf{A}^T
\end{aligned} \tag{4.3}$$

Where $V$ is the eigenvector of matrix $\Sigma$. if we let $\mathbf{A} = (V\mathrm{diag}(\lambda)^{1/2})^{-1}$, we can get that $\mathrm{Cov}[\mathbf{Y}] = \mathbf{A}\Sigma\mathbf{A}^T = E$

## 5 P5

As $\hat{\sigma}^2$ is the unbiased estimation of $\sigma^2$, we can get:

$$\begin{aligned} \mathrm{E}[\hat{\sigma}^2] &= \mathrm{E}\left[\sum_{k=1}^{K} \alpha_k \hat{\sigma}_k^2\right] = \sum_{k=1}^{K} \alpha_k \mathrm{E}\left[\hat{\sigma}_k^2\right] \\ &= \sigma^2 \end{aligned} \tag{5.1}$$

Also, according to Gaussian assumption:

$$\begin{aligned} \frac{(n_k-1)\hat{\sigma}_k^2}{\sigma^2} &\sim \chi^2_{n_k-1} \\ \mathrm{Var}\left[\frac{(n_k-1)\hat{\sigma}_k^2}{\sigma^2}\right] &= \mathrm{Var}\left[\chi^2_{n_k-1}\right] \\ \frac{(n_k-1)^2}{\sigma^4}\mathrm{Var}[\hat{\sigma}_k^2] &= 2(n_k-1) \\ \mathrm{Var}[\hat{\sigma}_k^2] &= \frac{2\sigma^4}{n_k-1} \end{aligned} \tag{5.2}$$

Thus:

$$\begin{aligned} \mathrm{Var}\left[\hat{\sigma}^2\right] &= \mathrm{Var}\left[\sum_{k=1}^{K} \alpha_k \hat{\sigma}_k^2\right] \\ &= \sum_{k=1}^{K} \alpha_k^2 \mathrm{Var}\left[\hat{\sigma}_k^2\right] \\ &= \sum_{k=1}^{K} \alpha_k^2 \frac{2\sigma^4}{n_k-1} \\ &= 2\sigma^4 \sum_{k=1}^{K} \frac{\alpha_k^2}{n_k-1} \end{aligned} \tag{5.3}$$

To find the value of $\alpha_k$ that minimizes $\mathrm{Var}\left[\hat{\sigma}^2\right]$, the problem is equal to find the $\alpha_k$ that minimizes $\sum_{k=1}^{K} \frac{\alpha_k^2}{n_k-1}$. According to the Lagrange Multiplier Method, we can set up the equations to find the optimal value:

$$\mathscr{L}(\alpha_k) = \sum_{k=1}^{K} \frac{\alpha_k^2}{n_k-1} + \lambda\left(\sum_{k=1}^{K} \alpha_k - 1\right) \tag{5.4}$$

For each $\alpha_k$, we need to guarantee that:

$$\begin{aligned} \frac{\partial \mathscr{L}(\alpha_k)}{\partial \alpha_k} &= \frac{2\alpha_k}{n_k-1} + \lambda = 0 \\ \alpha_k &= -\frac{\lambda}{2} * (n_k-1) \end{aligned} \tag{5.5}$$

Also, note the constrain that $\sum_{k=1}^{K} \alpha_k = 1$. We can calculate that $\alpha_k = \frac{n_k-1}{n-K}$.

# 6 P6

## 6.1 Majority Vote

```r
samples = c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)

red = 0
green = 0

for(i in 1:length(samples)) {
  if(samples[i] >= 0.5) {
    red = red + 1
  }
  else {
    green = green + 1
  }
}
red
# [1] 6
green
# [1] 4
```

## 6.2 Average Probability

```r
samples = c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)

mean(samples) >= 0.5
# False
```

## 6.3 Conclusion

The result of Majority Vote indicate that the answer should be red. The result of Average Probability indicate that the answer should be green.

# 7 P7

## 7.1 a

```
library(ISLR)
library(tree)
attach(OJ)
set.seed(1000)

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

## 7.2 b

```
OJ.tree = tree(Purchase ~ ., data = OJ.train)
summary(OJ.tree)
# 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
```

The tree uses LoyalCH, PriceDiff and salePriceMM. The training error rate is 0.16. The tree has 8 terminal nodes.

## 7.3 c

```
OJ.tree
# 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 ) *
```

I want to take terminal node 7 as an example. The variable got in this terminal is LoyalCH. Value of this variable is 0.764572. There are 260 nodes in the sub-trees of this node. Deviance of these nodes is 91.11. The predicted y-value for Purchase of this node is CH. There are 0.95769 of all nodes in the sub-trees giving a prediction of CH for Purchase while 0.04231 of the nodes giving a prediction of MM. The star marks this node as a terminal node.

### 7.4 d

```r
plot(OJ.tree)
text(OJ.tree)
title("Tree")
```

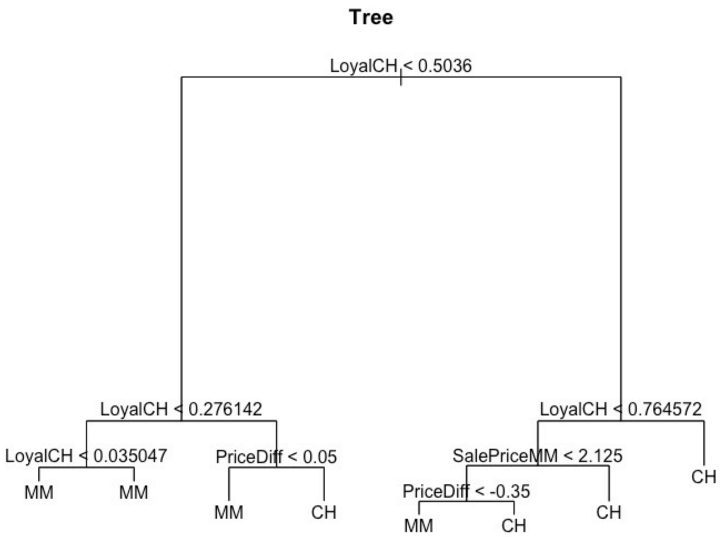


Figure 7.1: Screenshot for *'Tree Model'*

### 7.5 e

```r
OJ.pred = predict(OJ.tree, OJ.test, type = "class")
table(OJ.test$Purchase, OJ.pred)
#      OJ.pred
#        CH   MM
#    CH 150   11
#    MM  38   71
OJ.error = sum(OJ.test$Purchase != OJ.pred)/dim(OJ.test)[1]
OJ.error
# [1] 0.1814815
```

### 7.6 f

```r
OJ.cvtree = cv.tree(OJ.tree, FUN = prune.tree)
OJ.cvtree
# $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 1068.1253
#
# $k
# [1]      -Inf  11.87503  12.41171  29.77434  31.80546  39.82936  41.38321 306.37571
#
# $method
# [1] "deviance"
#
# attr(,"class")
# [1] "prune"         "tree.sequence"
```

The optimal size is 8.

### 7.7 g

```
plot(OJ.cvtree$size, OJ.cvtree$dev, pch = 20, type = "o", col = "blue",
     xlab = "Tree Size",ylab = "Dev")
title("Dev-Size Curve")
```

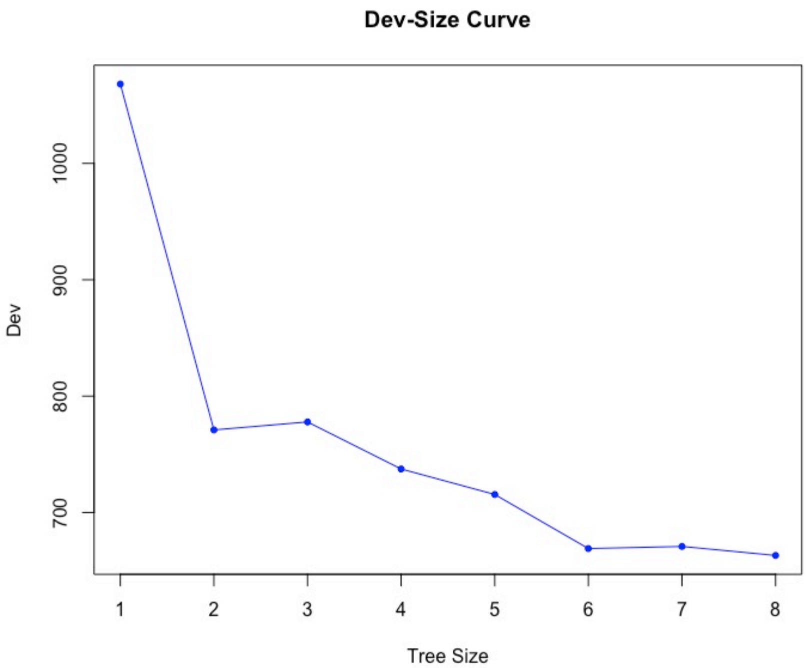


Figure 7.2: Screenshot for *'Dev-Size Curve'*

### 7.8 h

```
OJ.cvtree$size[which.min(OJ.cvtree$dev)]
# [1] 8
```

When the size is 8, we can get the lowest cross-validation error.

### 7.9 i

```
OJ.pruned = prune.tree(OJ.tree, best = OJ.cvtree$size[which.min(OJ.cvtree$
OJ.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 ) *
```

### 7.10 j

```
summary(OJ.pruned)
# 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
```

The pruned tree selects LoyalCH, PriceDiff and SalePriceMM. It has 8 terminal nodes. The test error does not change.

### 7.11 k

```
OJ.prunedpred = predict(OJ.pruned, OJ.test, type = "class")
table(OJ.test$Purchase, OJ.prunedpred)
#      OJ.prunedpred
#        CH  MM
#    CH 150  11
#    MM  38  71
OJ.prunederror = sum(OJ.test$Purchase != OJ.prunedpred)/dim(OJ.test)[1]
OJ.prunederror
# [1] 0.1814815
```

The pruned test error is the same with the original test error.