tw2906_HW3

Tong Wu

2022-10-18

Problem 1

Part a

$$\hat{p}(x) = \frac{e^{-6+0.05x_1+x_2}}{1+e^{-6+0.05x_1+x_2}} \approx 0.378$$

Part b

$$\hat{p}(X) = \frac{e^{-6+0.05x_1+x_2}}{1+e^{-6+0.05x_1+x_2}} = 0.5$$

$$\hat{p}(X) = \frac{e^{-6+0.05x_1+3.5}}{1+e^{-6+0.05x_1+3.5}} = 0.5$$

$$e^{-6+0.05x_1+3.5} = 1$$

$$x_1 = \frac{ln(1)-(-6)-3.5}{0.05} = 50$$

#Problem 2 By using Bayes theorem, the equation can be written as:

$$p_1(4) = \frac{0.8e^{\frac{-(4-10)^2}{72}}}{0.8e^{\frac{-(4-10)^2}{72}} + 0.2e^{\frac{-(4-0)^2}{72}}} \approx 0.75$$

Problem 3

```
knitr::include_graphics("./src/3.jpg")
```

Problem 3

$X = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]^T$

$Y = [F, F, F, T, F, T]$

For log-likehood function.

$$P[Y = j \mid X = x] = p(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}, \quad j = \text{true}.$$

$$L(\beta_0, \beta_1) = \prod_{i=1}^{6} p(x_i)^{y_i} (1 - p(x_i))^{1 - y_i}$$

$$\ln L(\beta_0, \beta_1) = \ell(\beta_0, \beta_1) = \sum_{i=1}^{6} \left(y_i \log p(x_i) \right) + (1 - y_i) \cdot (1 - \log p(x_i))$$

$$= \sum_{i=1}^{6} -\log\left(1 + e^{\beta_0 + \beta_1 x}\right) + y_i (\beta_0 + \beta_1 x_i)$$

where the $\ell(\beta_0, \beta_1)$ gradient:

$$\nabla \ell = \begin{bmatrix} -\sum \left(\frac{e^{\beta_0 \beta_1 x_i}}{1 + e^{\beta_0 \beta_1 x_i}} - y_i \right) \\ -\sum \left(\frac{e^{\beta_0 \beta_1 x_i}}{1 + e^{\beta_0 \beta_1 x_i}} - y_i \right) x_i \end{bmatrix}$$

$$= \begin{bmatrix} -\sum \left(\frac{e^{\beta_0 \beta_1 x_i}}{1 + e^{\beta_0 \beta_1 x_i}} \right) + 2 \\ -\sum \left(\frac{e^{\beta_0 \beta_1 x_i}}{1 + e^{\beta_0 \beta_1 x_i}} x_i \right) + 1.6 \end{bmatrix}$$

In GSL, the second (update) step of Newton step is:

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

$$\begin{bmatrix} \sum \left(e^{\beta_0 \beta_1 x_i} / \left(1 + e^{\beta_0 + \beta_1 x_i}\right)^2 \right), & \sum \left(e^{\beta_0 \beta_1 x_i} / \left(1 + e^{\beta_0 + \beta_1 x_i}\right)^2 \cdot x_i \right) \\ \sum \left(e^{\beta_0 \beta_1 x_i} / \left(1 + e^{\beta_0 + \beta_1 x_i}\right)^2 \cdot x_i \right), & \sum \left(e^{\beta_0 \beta_1 x_i} / \left(1 + e^{\beta_0 + \beta_1 x_i}\right)^2 \cdot x_i^2 \right) \end{bmatrix}$$

Figure 1: Problem 3 solution

```
library(matlib)
# For x and y
x <- c(0, 0.2, 0.4, 0.6, 0.8, 1.0)
y <- c(0, 0, 0, 1, 0, 1)
# For div_l
div_l <- function(beta) {
  m_1 <- sum( ((exp(beta[1] + beta[2]*x)) / (1 + exp(beta[1]+beta[2]*x))) - y )
  m_2 <- sum( (((exp(beta[1] + beta[2]*x)) / (1 + exp(beta[1]+beta[2]*x))) - y)*x )
  matrix(c(m_1,m_2),ncol=1,nrow = 2)
}
# For Newtown update matrix
newton_update <- function(beta) {
  mm_1 <- sum( (exp(beta[1] + beta[2]*x)) / (1 + exp(beta[1]+beta[2]*x))^2 )
  mm_2 <- sum( exp(beta[1] + beta[2]*x)*x / ((1 + exp(beta[1] + beta[2]*x))^2) )
  mm_3 <- sum( exp(beta[1] + beta[2]*x)*x / ((1 + exp(beta[1] + beta[2]*x))^2) )
  mm_4 <- sum( exp(beta[1] + beta[2]*x)*(x^2) / ((1 + exp(beta[1] + beta[2]*x))^2) )
  matrix(c(mm_1,mm_3,mm_2,mm_4),ncol = 2,nrow = 2)
}
beta <- c(-1, 1)
sprintf("Beta_0 = %s, Beta_1 = %s", beta[1], beta[2])
```

```
## [1] "Beta_0 = -1, Beta_1 = 1"
```

```
for(i in 1:10) {
  beta <- beta - inv(newton_update(beta))%*%div_l(beta)
  cat("For the iteration " , i, ", beta0 = ", beta[1], ", beta1 = ",beta[2], "\n")
}
```

```
## For the iteration  1 , beta0 =  -2.705639 , beta1 =  3.841178
## For the iteration  2 , beta0 =  -3.70133 , beta1 =  5.200004
## For the iteration  3 , beta0 =  -4.061292 , beta1 =  5.675472
## For the iteration  4 , beta0 =  -4.097643 , beta1 =  5.722885
## For the iteration  5 , beta0 =  -4.09797 , beta1 =  5.723309
## For the iteration  6 , beta0 =  -4.09797 , beta1 =  5.723309
## For the iteration  7 , beta0 =  -4.09797 , beta1 =  5.723309
## For the iteration  8 , beta0 =  -4.09797 , beta1 =  5.723309
## For the iteration  9 , beta0 =  -4.09797 , beta1 =  5.723309
## For the iteration  10 , beta0 =  -4.09797 , beta1 =  5.723309
```

```
model<-glm(y~x, family = "binomial")
sprintf("Beta_0= %s, Beta_1 = %s when using logistic regression",model$coefficients[1], model$coefficie
```

```
## [1] "Beta_0= -4.09797004234181, Beta_1 = 5.723308541489 when using logistic regression"
```

Problem 4

```
knitr::include_graphics("./src/4.jpg")
```

For distribution $Y = AX$, $E[Y] = E[AX] = 0$.

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

$$\begin{aligned}
\det(\Sigma - \lambda I) &= (\lambda - \sigma_1^2)(\lambda - \sigma_2^2) - \rho^2 \sigma_1^2 \sigma_2^2 \\
&= \lambda^2 - \lambda(\sigma_1^2 + \sigma_2^2) + (1 - \rho^2)(\sigma_1 \sigma_2)^2 \\
&= 0.
\end{aligned}$$

where $\lambda_1 = \dfrac{\sigma_1^2 + \sigma_2^2 + \sqrt{(\sigma_1^2 + \sigma_2^2)^2 - 4(1-\rho^2)(\sigma_1 \sigma_2)^2}}{2}$

$\lambda_2 = \dfrac{\sigma_1^2 + \sigma_2^2 - \sqrt{(\sigma_1^2 + \sigma_2^2)^2 - 4(1-\rho^2)(\sigma_1 \sigma_2)^2}}{2}$

Q is eigenfactor of Σ eigenvalue.

$A\Sigma A^T = A(Q\sqrt{\Lambda})(Q\sqrt{\Lambda})^T A^T$

$(Q\sqrt{\Lambda})^{-1} = \frac{1}{\sqrt{\Lambda}} Q^T$

$A \Sigma A^T = I.$

Figure 2: Problem 4 solution

Problem 5

```
knitr::include_graphics("./src/5.jpg")
```

Since $\hat{\sigma}^2 = \sum_{k=1}^{K} a_k \hat{\sigma}_k^2$; $n = \sum_{k=1}^{K} n_k$

$$Var(\hat{\sigma}^2) = \sum_{k=1}^{K} a_k^2 Var(\hat{\sigma}_k^2) = \sum_{k=1}^{K} a_k^2 \frac{2\sigma^4}{n_{k-1}}$$

For $\min \sum_{k=1}^{K} \frac{a_k^2}{n_{k-1}}$, the unconstraind for ferm $\frac{a_k}{n_{k-1}}$ is:

$$\sum_{k=1}^{K} \frac{a_k^2}{n_{k-1}} + \lambda \left(\sum_{k=1}^{K} a_k - 1 \right)$$

For the first order opeimaility condiion:

$$\frac{2a_k}{n_k - 1} + \lambda = 0$$

$$a_k = \frac{-\lambda}{2}(n_k - 1)$$

Shows that $a_k \propto (n_k - 1)$, so:

$$a_k = \frac{(n_k - 1)}{(n - k)}$$

Figure 3: Problem 5 solution

Problem 6

```
samples <- c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
# For majority vote approach
# Choose the majority probability sample
majority <- sum(samples > 1/2)
minority <- sum(samples < 1/2)
majority
```

```
## [1] 6
```

```
minority
```

```
## [1] 4
```

```
minority > majority
```

```
## [1] FALSE
```

```
majority > minority
```

```
## [1] TRUE
```

```
# For average probability
mean(samples)
```

```
## [1] 0.45
```

So from the code, in the majority approach, the red probability, which is 0.6, is greater than the green probability, which is 0.4. So the classification result is red. For the average probability approach, the mean of the sample probability is 0.45, which is smaller than 50%, so the classification result is green.

Problem 7

a

```
library(ISLR)
attach(OJ)
set.seed(1000)
train <- sample(dim(OJ)[1], 800)
training_set <- OJ[train, ]
testing_set <- OJ[-train, ]
```

b

```
library(tree)
tree <- tree(formula = Purchase ~., data = training_set)
summary(tree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = training_set)
## 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 training error rate is 0.16, and the number of terminal node is 8. The result has mean deviance with 0.7486 and 3 variables are used in tree construction.

c

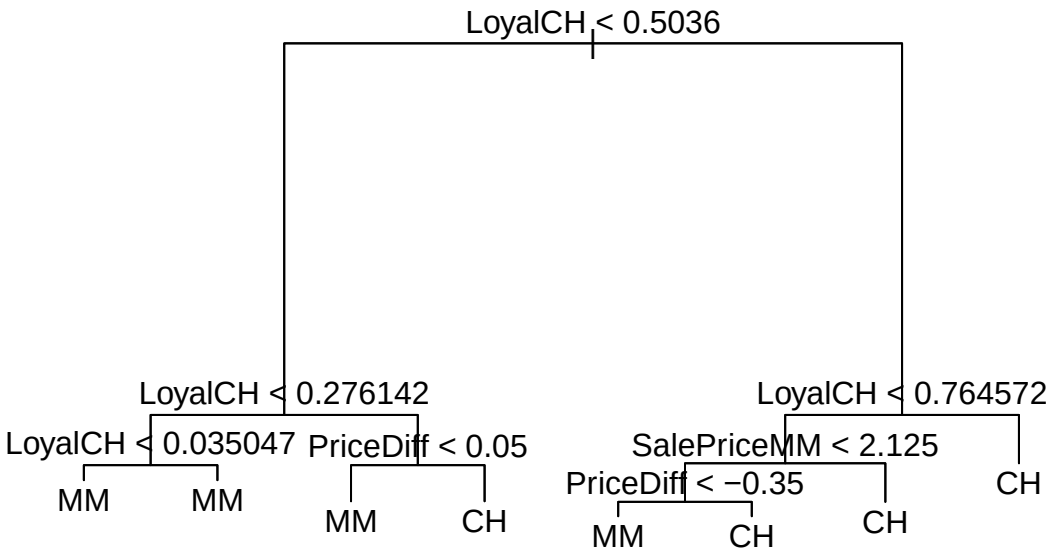
```
tree
```

```
## 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 ) *
```

For the node 'SpecialMM' with number 12, the split value is 2.125 and the 120 points below from this. 156.6 shows the deviance for points, is shows that the prediction is CH, where the 64.167% will be CH and another 35.833% is MM.

d

```
plot(tree)
text(tree)
```



From the tree diagram, can found that the variable 'LoyalCH' is most important. And for example, if the LoyalCH bigger than 0.764572, than the model predicts CH, and if PriceDiff $<$ -0.35 the model precicts MM.

e

```
predict <- predict(tree, testing_set, type = "class")
table(testing_set$Purchase, predict)
```

```
##     predict
##       CH  MM
##   CH 150  11
##   MM  38  71
```

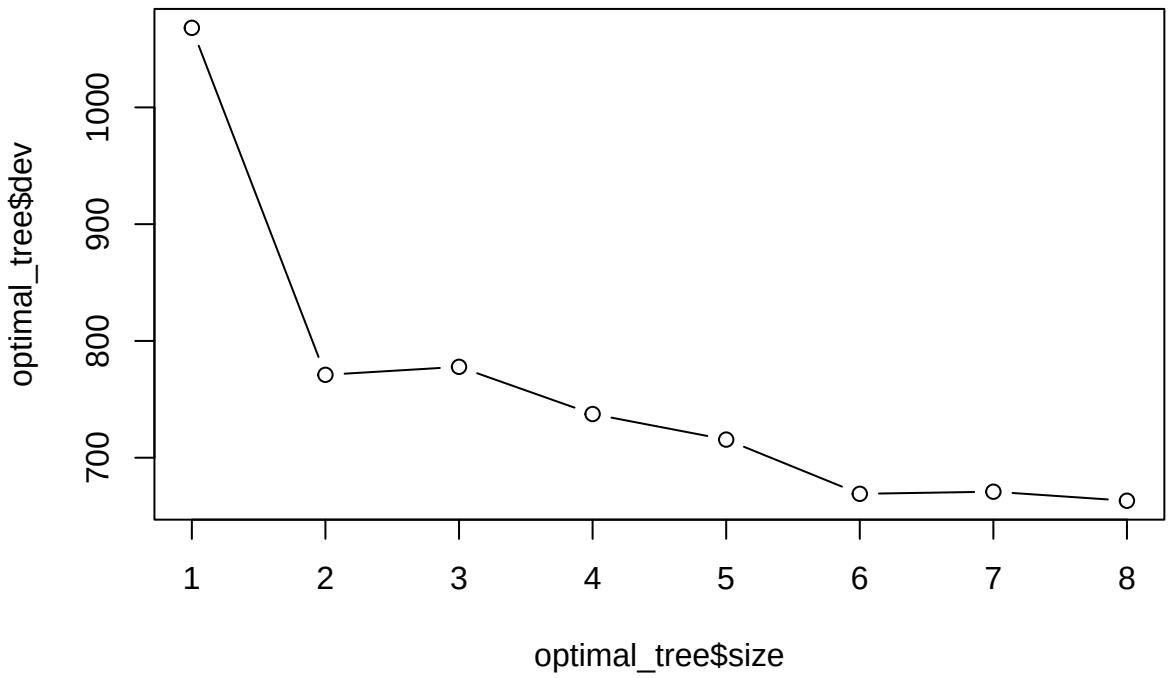
Where the test error rate is $\frac{11+38}{150+11+38+71} = \frac{49}{270} = 0.1815$

f

```
optimal_tree <- cv.tree(tree, FUN=prune.tree)
```

g

```
plot(optimal_tree$size, optimal_tree$dev, type = "b")
```



h

Size 8 has lowest cross-validated classification error rate

i

```
pruned_tree <- prune.tree(tree, best = 8)
```

j

```
summary(tree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = training_set)
```

```
## 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
```

```
summary(pruned_tree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = training_set)
## 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 error for the two tree is same.

k

```
# For pruned tree
predict_pruned = predict(pruned_tree, testing_set, type = "class")
sum(testing_set$Purchase != predict_pruned)/length(predict_pruned)
```

```
## [1] 0.1814815
```

```
# For unpruned tree
sum(testing_set$Purchase != predict)/length(predict)
```

```
## [1] 0.1814815
```

Where the test error rate of pruned and unpruned tree are the same.