yd2459_HW3

Yihang Ding

10/21/2019

P3 Use NewtonRaphson algorithm from Section 4.4.1, pp. 120-121, [ESL] book and perform 10 iterations. Hint: Use library(matlib) for calculating matrix inverses.

```
library(matlab)
```

```
##
## Attaching package: 'matlab'

## The following object is masked from 'package:stats':
##
##     reshape

## The following objects are masked from 'package:utils':
##
##     find, fix

## The following object is masked from 'package:base':
##
##     sum
```

```
x = c(0.0, 0.2, 0.4, 0.6, 0.8, 1.0)
y = c(0, 0, 0, 1, 0, 1)
first = function(beta){
  f1 = sum((exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x)))-y)
  f2 = sum(((exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x)))-y)*x)
  matrix(c(f1, f2), ncol=1, nrow=2)
}

second = function(beta){
  f1 = sum(exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x))^2)
  f2 = sum(exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x))^2*x)
  f3 = sum(exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x))^2*x)
  f4 = sum(exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x))^2*x*x)
  matrix(c(f1, f2, f3, f4), ncol = 2, nrow = 2)
}

beta = c(0, 0)
sequence = seq(1,10)
b1 = seq(1,10)
b2 = seq(1,10)
for(val in sequence){
  # update: new = old - second^-1 * first
  beta = beta - solve(second(beta))%*%first(beta)
  cat("Update ", val, ",beta0: ",beta[1], ", beta1:", beta[2],"\n")
  b1[val] = beta[1]
  b2[val] = beta[2]
}
```

```
## Update  1 ,beta0:  -2.380952 , beta1: 3.428571
## Update  2 ,beta0:  -3.522775 , beta1: 4.966947
## Update  3 ,beta0:  -4.022333 , beta1: 5.624766
## Update  4 ,beta0:  -4.096585 , beta1: 5.721513
## Update  5 ,beta0:  -4.09797 , beta1: 5.723308
## Update  6 ,beta0:  -4.09797 , beta1: 5.723309
## Update  7 ,beta0:  -4.09797 , beta1: 5.723309
## Update  8 ,beta0:  -4.09797 , beta1: 5.723309
## Update  9 ,beta0:  -4.09797 , beta1: 5.723309
## Update  10 ,beta0:  -4.09797 , beta1: 5.723309
```

```
b1
```

```
##  [1] -2.380952 -3.522775 -4.022333 -4.096585 -4.097970 -4.097970 -4.097970
##  [8] -4.097970 -4.097970 -4.097970
```

```
b2
```

```
##  [1] 3.428571 4.966947 5.624766 5.721513 5.723308 5.723309 5.723309
##  [8] 5.723309 5.723309 5.723309
```

```
select = glm(y~x,family = "binomial")
select$coefficients[1]
```

```
## (Intercept)
##    -4.09797
```

```
select$coefficients[2]
```

```
##        x
## 5.723309
```

Answer: β_0 = -4.098, β_1 = 5.723

P7 (a) First run set.seed(1000), and then create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
library(ISLR)
set.seed(1000)
dim(OJ)
```

```
## [1] 1070   18
```

```
sample = sample.int(n = dim(OJ)[1], size = 800, replace = F)
train = OJ[sample, ]
test = OJ[-sample, ]
dim(train)
```

```
## [1] 800  18
```

```r
dim(test)
```

```
## [1] 270  18
```

(b) Fit a tree to the training data, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

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

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = 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 is 0.16, the tree has 8 terminal nodes.

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

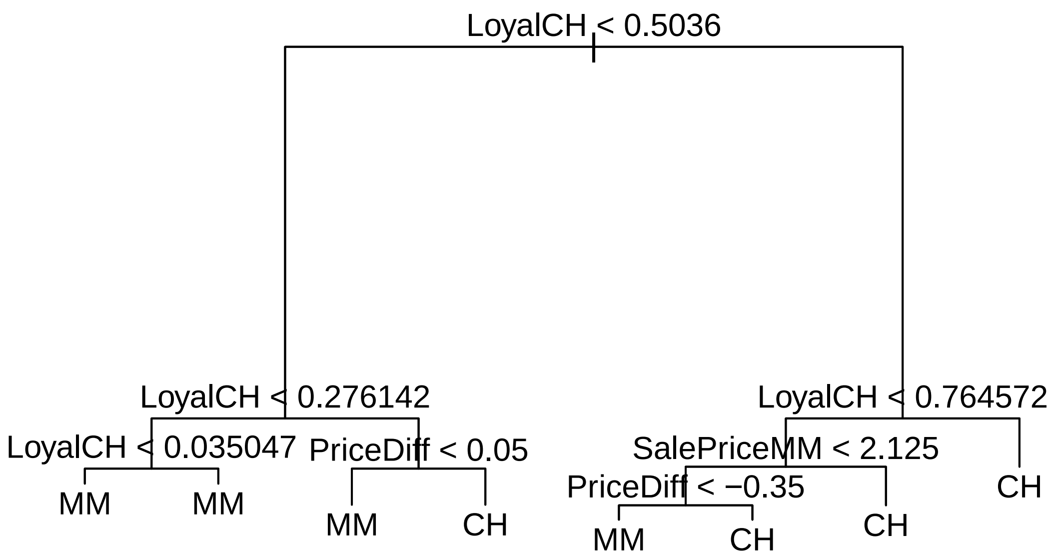
```r
fit
```

```
## 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 ) *
```

Take "9) LoyalCH > 0.035047 113 108.50 MM (0.18584 0.81416) *" for example, it uses LoyalCH with threshold 0.35047 as split feature, and get 113 samples. With deviance 108.5 and prediction value as MM.

(d) (2pt) Create a plot of the tree, and interpret the results.

```r
plot(fit, main='OJ train Decision Tree')
text(fit)
```



(e) (3pt) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```r
pred = predict(fit, test, type = 'class')
summary(pred)
```

```
##  CH  MM
## 188  82
```

```r
table(pred, test$Purchase)
```

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

```r
mean(pred != test$Purchase)
```

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

Test error rate is 0.18.

(f) (2pt) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

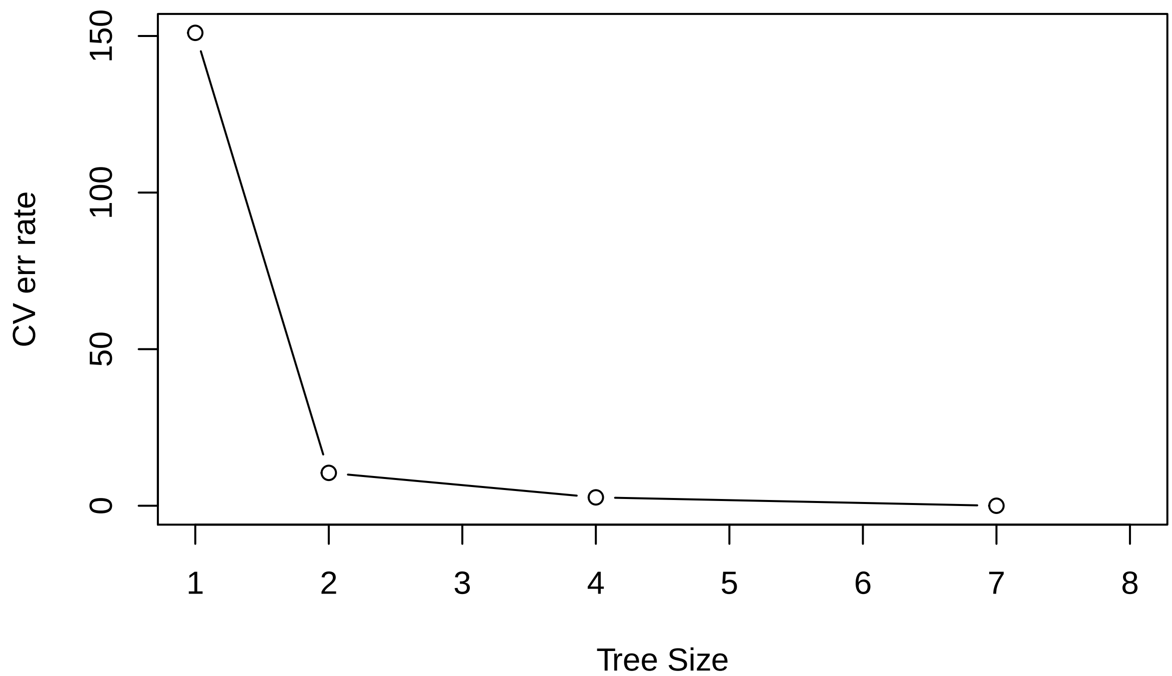
```
cv_fit = cv.tree(fit, FUN=prune.misclass)
cv_fit
```

```
## $size
## [1] 8 7 4 2 1
##
## $dev
## [1] 142 142 143 164 308
##
## $k
## [1]       -Inf   0.000000   2.666667  10.500000 151.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Optimal tree size is 4.

(g) (3pt) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
plot(x=cv_fit$size, y=cv_fit$k, type='b', xlab='Tree Size', ylab='CV err rate')
```



(h) (1pt) Which tree size corresponds to the lowest cross-validated classification error rate? Tree with size 4 has the lowest CV classification error rate.

(i) (3pt) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
prune = prune.misclass(fit, best=4)
summary(prune)
```

```
##
## Classification tree:
## snip.tree(tree = fit, nodes = 4:3)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  4
## Residual mean deviance:  0.8653 = 688.8 / 796
## Misclassification error rate: 0.17 = 136 / 800
```

(j) (2pt) Compare the training error rate between the pruned and unpruned tree. Which is higher? Pruned tree has higher train error rate.

(k) (2pt) Compare the test error rates between the pruned and unpruned trees. Which is higher?

```r
prune_pred = predict(prune, test, type = 'class')
mean(prune_pred != test$Purchase)
```

```
## [1] 0.2037037
```

Pruned tree has higher test error rate.