Classification_Sol

2018 7 11

실습 1

mpg.csv 자료를 이용하여 다음을 코딩하라.(결측값은 제거할 것) (조건: 반응 변수:mpg01, 예측 변수:cylinders, weight, displacement, horsepower로 지정)

```
Auto<-read.csv("/Users/kim/Dropbox/Grad/2018-1/R  /Classification/data/Auto.csv")
Auto=na.omit(Auto) ##
attach(Auto)
mpg01 = rep(0, length(mpg))
mpg01[mpg > median(mpg)] = 1
Auto = data.frame(Auto, mpg01)
```

자료를 불러오고, mpg01이라는 변수에 중앙값보다 큰 경우에 1, 아닌 경우에 0으로 만듭니다.

```
train = (year%%2 == 0)
test = !train
Auto.train = Auto[train, ]
Auto.test = Auto[test, ]
mpg01.test = mpg01[test]
```

짝수해를 Training set, 나머지를 Test set으로 나눕니다.

```
library(MASS)
lda.fit = lda(mpg01 ~ cylinders + weight + displacement + horsepower, data = Auto,subset=train)
lda.pred = predict(lda.fit, Auto.test) ## Predict       Classifier    Test data set  Class      .
lda.class=lda.pred$class ## LDA    Class    .
mean(lda.class!=mpg01.test) ## LDA    Class    Class          ; Misclassification error
```

```
## [1] 0.1263736
```

LDA를 수행한 결과입니다.

```
qda.fit = qda(mpg01 ~ cylinders + weight + displacement + horsepower, data = Auto,subset = train)
qda.pred = predict(qda.fit, Auto.test)
qda.class=qda.pred$class
mean(qda.pred$class != mpg01.test)
```

```
## [1] 0.1318681
```

QDA는 LDA와 사용방법이 같습니다.

```
glm.fit = glm(mpg01 ~ cylinders + weight + displacement + horsepower, data = Auto,
              family = binomial, subset = train)
glm.probs = predict(glm.fit, Auto.test, type = "response")
glm.pred = rep(0, length(glm.probs))
glm.pred[glm.probs > 0.5] = 1
mean(glm.pred != mpg01.test)
```

```
## [1] 0.1208791
```

Logistic Regression을 위해서는 glm이라는 함수를 사용하고, “family=binomial” option을 이용합니다.

Logistic Regression에서도 역시 predict함수가 Training data set으로 적합한 회귀모형을 이용해, Test data set에서의 Class를 추정해줍니다. 여기서는 type=“response”로 두었기 때문에, Class가 1일 확률을 계산해줍니다.

```
library(class)
train.x = cbind(cylinders, weight, displacement, horsepower)[train, ]
test.x = cbind(cylinders, weight, displacement, horsepower)[test, ]
train.mpg01 = mpg01[train]
test.mpg01 = mpg01[test]

set.seed(1)
knn.pred = knn(train.x, test.x, train.mpg01, k = 1)
table(knn.pred,test.mpg01)
```

```
##         test.mpg01
## knn.pred  0  1
##        0 83 11
##        1 17 71
```

```
mean(knn.pred != mpg01.test)
```

```
## [1] 0.1538462
```

```
knn.pred = knn(train.x, test.x, train.mpg01, k = 10)
mean(knn.pred != mpg01.test)
```

```
## [1] 0.1648352
```

```
knn.pred = knn(train.x, test.x, train.mpg01, k = 100)
mean(knn.pred != mpg01.test)
```

```
## [1] 0.1428571
```

KNN 방법을 이용하기 위해서 class package에 있는 knn함수를 이용하고, 수행한 것은 위와 같습니다.

knn 함수는 data set 을 통째로 받는것이 아니라, X와 Y를 나눠서 받기 때문에, 자료를 나눠서 입력해주어야 합니다.

seed를 정하는 이유는 tie가 생겼을 때, Random하게 선택해서 사용하는데, 이 부분에서 항상 같은 결과를 얻기 위함입니다.

실습 2

```
Default<-read.table("/Users/kim/Dropbox/Grad/2018-1/R  /Classification/data/default.txt")
```

```
glm.fit = glm(default ~ income + balance, data = Default, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

Logistic Regression을 적합하면 위와 같습니다.

```
boot.fn = function(data, index){
  return(coef(glm(default ~ income + balance, data = data,
                  family =binomial, subset = index)))
}
```

boot.fn() 함수를 위와 같이 data와 index를 받아서, Logistic regression 을 실행하고, 계수를 return하는 함수로 정합니다.

```
library(boot)
boot(Default, boot.fn, 50)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 50)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01  1.183984e-01 4.195395e-01
## t2*  2.080898e-05 -4.757657e-08 4.556648e-06
## t3*  5.647103e-03 -7.010355e-05 2.277989e-04
```

이렇게 boot 함수를 이용하면, bootstrap을 50번 반복해, 표준오차 추정치를 구할 수 있습니다.