Yonsei University, Department of Applied Statistics

Chapter 6 실습

12기 YBIGTA 신보현

February 17, 2019

Contents

1. Best Subset Selection **3**

2. Forward and Backward Stepwise Selection **7**

3. Validation set approach and CV **8**

4. Ridge Regression **11**

5. Lasso Regression **16**

6. Principal Components Regression **18**

7. Partial Least Squares **22**

1. Best Subset Selection

Hitters data에 best subset selection을 적용해본다. 이 데이터를 통해 야구 선수의 Salary을 예측하고자 한다. 우선 NA 데이터를 없애자. best subset selection은 leaps library에 있다.

```
knitr::opts_chunk$set(comment=NA, fig.width=4, fig.height=4,fig.align='center',message=FALSE)
library(ISLR)
attach(Hitters)
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"
##  [6] "Walks"     "Years"     "CAtBat"    "CHits"     "CHmRun"
## [11] "CRuns"     "CRBI"      "CWalks"    "League"    "Division"
## [16] "PutOuts"   "Assists"   "Errors"    "Salary"    "NewLeague"
```

```
Hitters = na.omit(Hitters)
library(leaps)
reg.best = regsubsets(Salary~., data=Hitters)
summary(reg.best)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
```

```
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"
##          CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 ) "*"  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 ) "*"  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 ) "*"  " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 ) "*"  " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 ) "*"  " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 ) "*"  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 ) " "  " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 ) " "  "*"    " "     "*"       "*"     " "     " "    " "
```

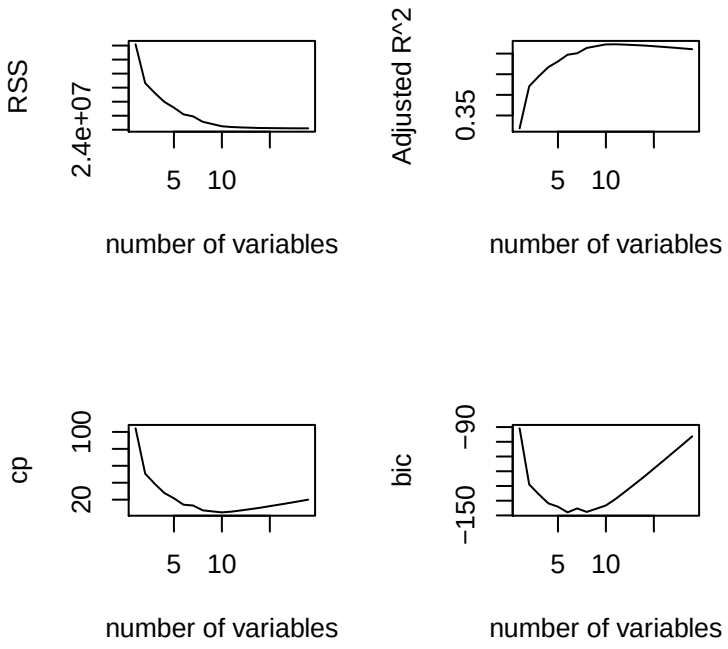
regsubsets 함수는 기본 값으로 8개의 변수까지만 보여준다. nvmax 옵션을 사용해서 이를 변경할 수 있다.

```
reg.best = regsubsets(Salary~., data=Hitters, nvmax=19)
best.summary = summary(reg.best)
names(best.summary)
```

```
[1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

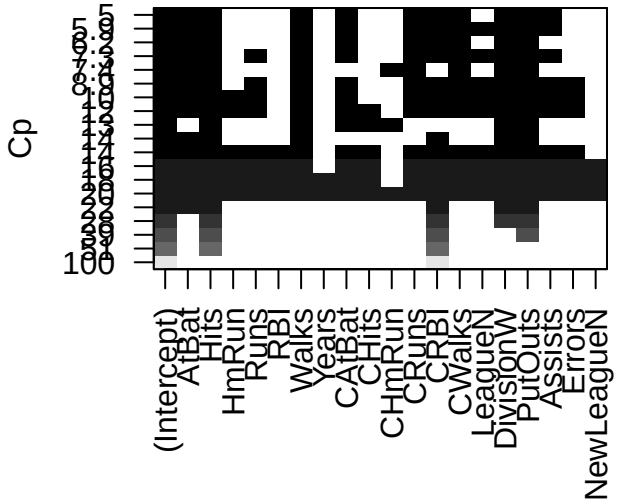
결정계수, RSS, 수정 결정계수, C_p, BIC 값을 볼 수 있는데 이를 통해서 가장 좋은 모델을 선택해야 한다.

```
par(mfrow=c(2,2))
plot(best.summary$rss,xlab="number of variables", ylab="RSS",type="l")
plot(best.summary$adjr2,xlab="number of variables", ylab="Adjusted R^2",type="l")
plot(best.summary$cp,xlab="number of variables", ylab="cp",type="l")
plot(best.summary$bic,xlab="number of variables", ylab="bic",type="l")
```

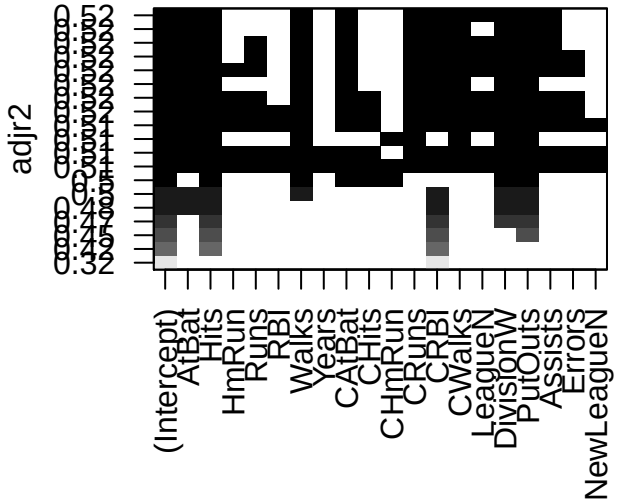


또한 내장 함수로 각 기준마다 가장 좋은 성능의 모델이 어떤 변수를 가지고 있는지 볼 수 있다.

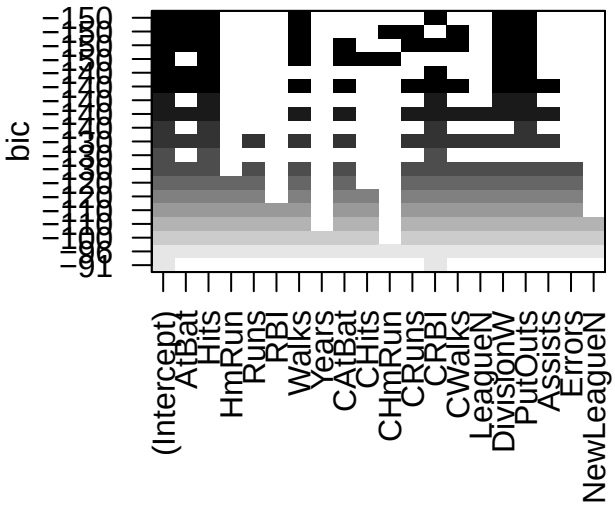
```
plot(reg.best,scale='Cp')
```




```
plot(reg.best,scale='adjr2')
```



```
plot(reg.best,scale='bic')
```



예를 들어, BIC에 대해서는 가장 작은 값, 즉 가장 위에 있는 모델이 좋은 모델이고 이 때의 변수와 그에 따른 계수를 살펴보면 아래와 같다.

```
coef(reg.best,6)
```

```
 (Intercept)        AtBat         Hits        Walks         CRBI
  91.5117981   -1.8685892    7.6043976    3.6976468    0.6430169
   DivisionW      PutOuts
-122.9515338    0.2643076
```

2. Forward and Backward Stepwise Selection

regsubsets() 함수에서 method="forward", 또는 method="backward"를 통해 실행할 수 있다.

```
reg.fwd = regsubsets(Salary~., data=Hitters, nvmax=19, method='forward')
reg.back = regsubsets(Salary~., data=Hitters, nvmax=19, method='backward')
```

그 이외의 옵션은 best subset의 옵션과 모두 동일하다.

3. Choosing among models using the validation set approach and cross-validation

위에서는 모든 데이터에 대해서 적합을 했다. 하지만 계속 살펴봤듯이, 모델을 평가하기 위해서는 그 모델을 적합하는데 쓰이지 않은 데이터로 평가를 해야한다. 따라서 5단원에서 살펴본 validation method을 사용한다. 원칙적으로는 파라미터 조정을 위해 데이터를 훈련세트, 검증세트, 테스트세트로 나누어야 하지만 subset approach에서는 파라미터 조정 과정이 없기 때문에 훈련세트, 테스트세트로만 나누어서 진행한다. 먼저 validation approach로 가장 좋은 모델을 선택해보자. validation approach는 데이터를 단순하게 두 부분으로 나눈 후, 훈련 세트로 적합을 하고 테스트 세트로 그 성능을 평가하는 방법이다.

```
set.seed(1)
train = sample(c(TRUE,FALSE),nrow(Hitters),rep=TRUE)
test = ! train
regfit.best = regsubsets(Salary~., data=Hitters[train,],nvmax=19)
```

모든 Hitters 데이터를 사용하는 것이 아니라, 그 중 일부인 train 행만을 뽑아서 적합시켰다.

```
test.mat = model.matrix(Salary~., data=Hitters[test,])
```

model.matrix() 함수는 design matrix인 X을 만들기 위해서 회귀 패키지에서 많이 사용하는 함수이다.
이제 변수가 1개부터 19개일 때의 best subset 결과를 통해서 테스트 오차의 추정치를 계산해보자.

```
val.errors = rep(0,19)
for (i in 1:19){
coefi = coef(regfit.best, id=i)
pred.value = test.mat[,names(coefi)] %*% coefi
val.errors[i] = mean((pred.value-Hitters$Salary[test])^2) }
val.errors
```

```
 [1] 220968.0 169157.1 178518.2 163426.1 168418.1 171270.6 162377.1
 [8] 157909.3 154055.7 148162.1 151156.4 151742.5 152214.5 157358.7
[15] 158541.4 158743.3 159972.7 159859.8 160105.6
```

```
which.min(val.errors)
```

```
[1] 10
```

```
coef(regfit.best,10)
```

```
(Intercept)       AtBat        Hits       Walks      CAtBat       CHits
-80.2751499  -1.4683816   7.1625314   3.6430345  -0.1855698   1.1053238
     CHmRun      CWalks     LeagueN   DivisionW     PutOuts
  1.3844863  -0.7483170  84.5576103 -53.0289658   0.2381662
```

변수를 10개 포함한 모델이 테스트 오차의 추정치가 가장 낮았으므로 이 모델이 validation approach을 통해 얻은 가장 좋은 모델이라고 말할 수 있다.
이렇게 for문을 돌린 이유는, regsubsets() 함수에는 predict()을 할 수 있는 방법이 없기 때문이었다. 이를 함수로 다시 만들어보자. 이 함수는 corss-validation에서 사용할 것이다.

```
pred.regsubsets = function(object,newdata,id,...){
form = as.formula(object$call[[2]]) #index formula usied in regsubsets
test.mat = model.matrix(form, data=newdata) # test design matrix with test data
coefi = coef(object,id=id) # estimated coefficients with id numbers of variables
xvars = names(coefi)
test.mat[,xvars] %*% coefi
}
```

마지막으로 10개의 변수를 사용하는 모델이 가장 낮은 테스트 오차의 추정치를 만들어냈다면, 이 10개의 변수와 모든 데이터 세트를 이용하여 다시 모델을 적합시킨다.

```
full.reg = regsubsets(Salary~., data=Hitters, nvmax=19)
coef(full.reg,id=10)
```

```
 (Intercept)        AtBat         Hits        Walks       CAtBat
```

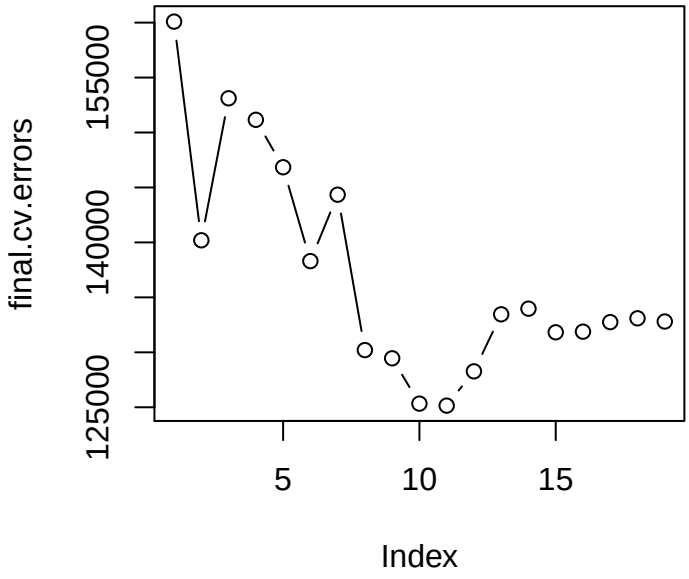
```
 162.5354420   -2.1686501    6.9180175    5.7732246   -0.1300798
       CRuns         CRBI       CWalks    DivisionW      PutOuts
   1.4082490    0.7743122   -0.8308264 -112.3800575    0.2973726
     Assists
   0.2831680
```

이제 cross-validation을 통해서 테스트 오차를 추정해보자. 10-fold cross-validation을 진행할 것이며, j개의 변수를 가지는 모델에 대해서 10번의 cross-validation을 시행할 때의 결과가 담길 행렬을 생성하여 여기에 결과를 입력한다.

```
k=10
set.seed(1)
folds = sample(1:k, nrow(Hitters), replace=TRUE)
cv.errors = matrix(0,k,19)
for (j in 1:k){
reg.fit = regsubsets(Salary~., data=Hitters[folds != j,],nvmax=19)
for (i in 1:19){
pred =  pred.regsubsets(reg.fit,newdata=Hitters[folds == j,],id=i)
cv.errors[j,i] = mean((pred - Hitters$Salary[folds == j])^2)
}
}
```

총 10×19의 행렬이 만들어졌으며 (i,j) 원소는 j개의 변수를 가지는 모델에 대해서, i번째 겹을 테스트 데이터로 했을 때의 테스트 오차 추정치를 의미한다. 따라서 최종적인 cross-validation 오차를 계산하기 위해서는 10번의 테스트 오차 추정치의 평균을 구해야 한다.

```
final.cv.errors = apply(cv.errors,2,mean)
final.cv.errors

 [1] 160093.5 140196.8 153117.0 151159.3 146841.3 138302.6 144346.2
 [8] 130207.7 129459.6 125334.7 125153.8 128273.5 133461.0 133974.6
[15] 131825.7 131882.8 132750.9 133096.2 132804.7

par(mfrow=c(1,1))
plot(final.cv.errors,type='b')
```



10개 또는 11개 변수를 포함하는 모델이 가장 낮은 cross-validation 오차를 가짐을 확인할 수 있다. 이를 확인하면, 다시 모든 데이터에 대해서 적합을 실시해야함을 잊지말자.

```
reg.best = regsubsets(Salary~., data=Hitters, nvmax=19)
coef(reg.best,id=11)
```

```
 (Intercept)        AtBat         Hits        Walks       CAtBat
135.7512195   -2.1277482    6.9236994    5.6202755   -0.1389914
      CRuns         CRBI       CWalks      LeagueN    DivisionW
  1.4553310    0.7852528   -0.8228559   43.1116152 -111.1460252
     PutOuts      Assists
  0.2894087    0.2688277
```

4. Ridge Regression

ridge 회귀는 glmnet 패키지에 있는 glmnet 함수를 통해서 실행할 수 있다. 이 함수는 다른 회귀 함수와는 다른 형태를 취한다. y~x 형태로 쓰는 것이 아니라, x는 행렬로, y는 벡터로 써야 한다. Hitters 데이터를 이에 맞게 변형하자. 결측치 또한 삭제하자(결측치는 이미 앞에서 삭제함)

```
x = model.matrix(Salary~.,Hitters)[,-1]
y = Hitters$Salary
```

model.matrix() 함수는 design matrix을 만드는데 특히 유용하다. 이 함수는 범주형 자료도 바로 dummay variables로 만들어 주기 때문에 편리하다. 특히 glmnet() 함수는 수치형 자료에 대해서만 실행할 수 있으므로 더미 변수로 반들어 주는 것을 잊지말자.

```
library(glmnet)
grid = 10^seq(10,-2,length=100)
ridge.mod = glmnet(x,y,alpha=0,lambda=grid)
names(ridge.mod)

 [1] "a0"         "beta"       "df"         "dim"        "lambda"
 [6] "dev.ratio" "nulldev"    "npasses"    "jerr"       "offset"
[11] "call"       "nobs"

dim(coef(ridge.mod))

[1]  20 100
```

기본 값으로, glmnet 함수는 ridge 회귀를 자동으로 실행한다. 여기서 우리는 λ 값을 10^{10} 부터 10^{-2}까지 설정했다. ridge 회귀나 lasso 회귀를 시행할 때, 표준화를 하는 것이 중요함을 다시 한번 상기해보자. glmnet 함수는 자동으로 표준화를 진행하여 이러한 걱정을 덜어준다.
coef(ridge.mod)는 각 λ 값에 대해서 계수 추정치를 행렬로 나타낸 것이다.
alpha는 아래의 식에서 α을 의미한다.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x\right)^2+\alpha\sum_{j=1}^{p}\mid\beta_j\mid+(1-\alpha)\sum_{i=1}^{p}\beta_j^2$$

즉, alpha=0으로 설정을 하면 ℓ_2 penalty 항만 남게 되어 ridge 회귀 식과 동일해지고 alpha=1로 설정을 하면 ℓ_1 penalty 항만 남게 되어 lasso 회귀 식과 동일해진다. α를 0과 1 사이의 수로 설정을 하면 elastic net을 시행하는 것과 동일하다.
λ 값이 크다면 penalty가 커서 계수가 그만큼 더 shrink될 것이고 λ 값이 작다면 계수가 상대적으로 클 것이다. 이를 확인해보면 아래와 같다.

```
dim(coef(ridge.mod))
```

```
[1]  20 100
```

```
ridge.mod$lambda[50]
```

```
[1] 11497.57
```

```
coef(ridge.mod)[,50]
```

```
  (Intercept)         AtBat          Hits         HmRun          Runs
407.356050200   0.036957182   0.138180344   0.524629976   0.230701523
          RBI         Walks         Years        CAtBat         CHits
  0.239841459   0.289618741   1.107702929   0.003131815   0.011653637
       CHmRun         CRuns          CRBI        CWalks       LeagueN
  0.087545670   0.023379882   0.024138320   0.025015421   0.085028114
    DivisionW       PutOuts       Assists        Errors    NewLeagueN
 -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

```
ridge.mod$lambda[70]
```

```
[1] 43.28761
```

```
coef(ridge.mod)[,70]
```

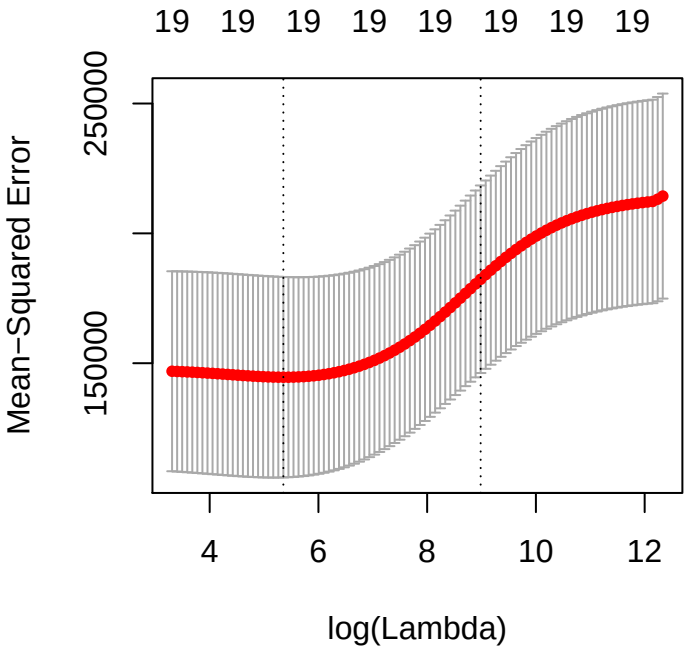
```
  (Intercept)         AtBat          Hits         HmRun          Runs
  54.97384215   -0.41480601    2.10530493   -1.34828331    1.13281252
          RBI         Walks         Years        CAtBat         CHits
   0.79219405    2.83508432   -6.85814163    0.00438123    0.11187771
       CHmRun         CRuns          CRBI        CWalks       LeagueN
   0.64020753    0.23468562    0.22724180   -0.17405551   47.59278798
    DivisionW       PutOuts       Assists        Errors    NewLeagueN
-119.52546741    0.25369445    0.13095051   -3.38369142  -11.36670636
```

이제 3 way hold out method을 통해서 ridge 회귀를 실행해보자(ISLR에는 2 way hold out method을 사용한다) 이를 위해 데이터를 훈련 세트, 테스트 세트로 나눈다. 그리고 훈련 세트에 대해서 cross-validation을 실행하여 가장 최적의 파라미터를 찾은 후에 해당 파라미터와 테스트 세트를 이용하여 모델의 최종 성능을 파악한다. lasso 회귀에 대해서도 위와 동일한 과정을 밟은 뒤에 얻은 테스트 세트에 대한 MSE를 ridge 회귀에서의 그것과 비교하여 더 적은 test MSE 추정치를 내는 모델을 선택하는 것이다. 우선 데이터를 반으로 나누어, 훈련 세트와 테스트 세트를 생성하자. 데이터를 두 부류로 나누는 방법은 두 가지가 있다. TRUE,FALSE를 포함하는 벡터에서 랜덤으로 추출하여 TRUE에 해당하는 데이터를 훈련 세트로 만드는 방법과 1부터 n까지의 subset을 랜덤으로 추출하여 이를 훈련 세트로 만드는 것이다. 이번에는 후자 방법을 사용한다.

```
set.seed(1)
train = sample(1:nrow(x), nrow(x)/2)
test = -train
y.test = y[test]
```

3 way hold out method에 의하면 평가하고자 하는 파라미터를 정한 뒤, 훈련세트에 해당 파라미터를 적용하여 모델을 생성한다. 하지만 ridge와 lasso에 관해서는, 가장 최적의 λ(가장 작은 cross-validation error을 내는)을 자동으로 찾아주기 때문에 해당 과정이 불필요하다.

```
set.seed(1)
cv.out = cv.glmnet(x[train,],y[train],alpha=0)
plot(cv.out)
```




```
bestlam = cv.out$lambda.min
bestlam


[1] 211.7416
```

가장 작은 cross-validation error을 내는 λ는 212인 것으로 확인되었다. 최적의 파라미터를 도출하였으니 훈련 세트 전체에 대해서 적합을 시킨 후, test MSE를 확인해보자.

```
ridge.mod = glmnet(x[train,],y[train],alpha=0,lambda=bestlam)
ridge.pred = predict(ridge.mod,newx=x[test,])
mean((ridge.pred - y.test)^2)


[1] 95981.59
```

ridge 회귀에 대해서는, λ를 212를 설정하는 것이 다른 파라미터보다 더 좋은 성능을 냈으며 최종 test MSE는 위와 같았다. 바로 이 test MSE 값을 다른 모델, 예를 들어 lasso 회귀에서 나온 최종 test MSE 값과 비교하여 lasso를 사용할지 ridge를 사용할지 결정하는 것이다.

이러한 과정을 거쳐서 ridge의 test MSE가 더 적음을 확인했다고 하자. 그렇다면, 이제 모든 모든 데이터를 해당 파라미터를 사용하여 적합하는 것이다.(5단원에서의 3 way hold out method을 그대로 따라함)

```
final.out = glmnet(x,y,alpha=0,lambda=bestlam)
coef(final.out)
```

```
20 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)   9.81383685
AtBat         0.03174457
Hits          1.00837355
HmRun         0.14062186
Runs          1.11296019
RBI           0.87318566
Walks         1.80365301
Years         0.13743551
CAtBat        0.01114900
CHits         0.06489415
CHmRun        0.45152976
CRuns         0.12888607
CRBI          0.13726927
CWalks        0.02919186
LeagueN      27.17300577
DivisionW   -91.62094975
PutOuts       0.19145581
Assists       0.04247849
Errors       -1.81160163
NewLeagueN    7.22425931
```

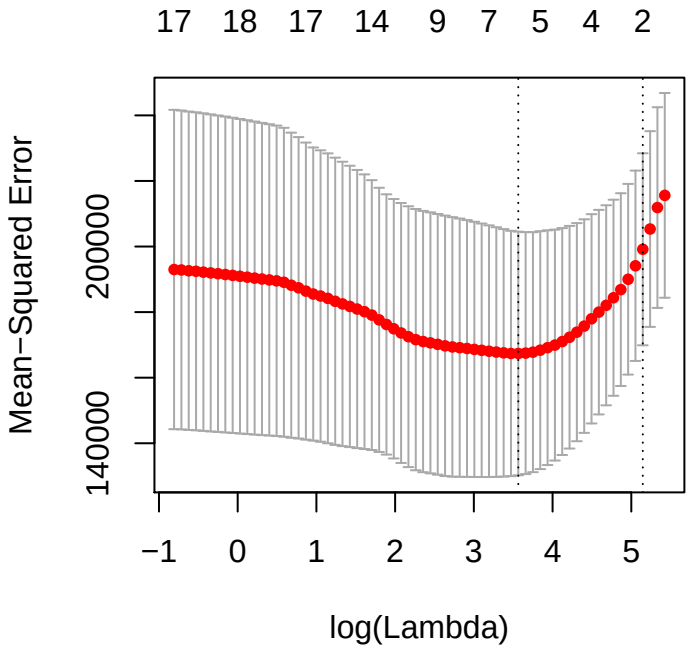
위의 계수에서 확인할 수 있듯이, 추정된 계수가 0과 동일한 변수는 없다. 즉, ridge는 변수 선택을 하지 않는 것이다!

5. Lasso Regression

lasso 회귀를 시행하기 위해서는 앞의 과정을 동일하게 밟는데, alpha=1로 바꿔주기만 하면

된다. 그러면 ridge와 동일하게 3 way hold out method을 시행해보자.

```
cv.out.lasso = cv.glmnet(x[train,],y[train],alpha=1)
plot(cv.out.lasso)
```




```
bestlam.lasso = cv.out.lasso$lambda.min
bestlam.lasso
```

```
[1] 35.32063
```

```
lasso.mod = glmnet(x[train,],y[train],alpha=1,lambda=bestlam.lasso)
lasso.pred = predict(lasso.mod, newx=x[test,])
mean((lasso.pred-y.test)^2)
```

```
[1] 102443.3
```

가장 좋은 성능을 내는 λ는 35.3으로, 이를 활용한 test MSE는 102443.으로 확인되었다.

즉, Hitters 데이터에 대해서는 lasso보다는 ridge가 더 좋은 성능을 냄을 확인할 수 있지만 그 차이는 그렇게 크지는 않다. 마지막으로 lasso의 변수선택을 살펴보자.

```
final.out.lasso = glmnet(x,y,alpha=1,lambda=bestlam.lasso)
coef(final.out.lasso)


20 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)  58.4609978
AtBat         .
Hits          1.7104183
HmRun         .
Runs          .
RBI           .
Walks         1.9951689
Years         .
CAtBat        .
CHits         .
CHmRun        .
CRuns         0.1907335
CRBI          0.3859176
CWalks        .
LeagueN       .
DivisionW   -69.8925390
PutOuts       0.1726885
Assists       .
Errors        .
NewLeagueN    .
```

6. Principal Components Regression

PCR은 pls library에 있는 pcr() 함수를 통해 실행할 수 있다.

```
library(pls)
set.seed(2)
pcr.fit = pcr(Salary~., data=Hitters, scale=TRUE, validation="CV")
```

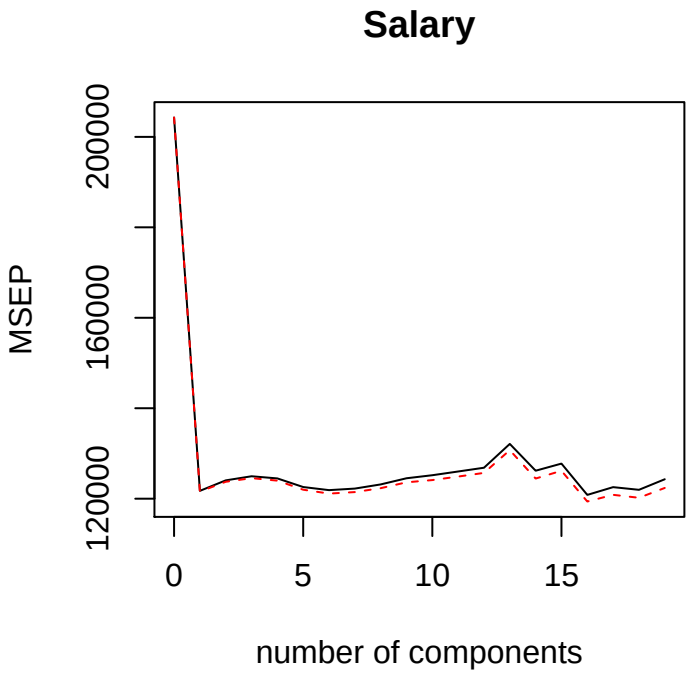
scale=TRUE을 통해 각 변수에서 표준화를 시행하고 validation="CV"를 통해 principal components의 수인 M의 각 값에 대해서 10-fold cross-validation error을 계산한다. 결과는 아래와 같다.

```
summary(pcr.fit)


Data:  X dimension: 263 19
Y dimension: 263 1
Fit method: svdpc
Number of components considered: 19

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV             452    348.9    352.2    353.5    352.8    350.1    349.1
adjCV          452    348.7    351.8    352.9    352.1    349.3    348.0
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       349.6    350.9    352.9     353.8     355.0     356.2     363.5
adjCV    348.5    349.8    351.6     352.3     353.4     354.5     361.6
       14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
CV        355.2     357.4     347.6     350.1     349.2     352.6
adjCV     352.8     355.2     345.5     347.6     346.7     349.8

TRAINING: % variance explained
        1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X         38.31    60.16    70.84    79.03    84.29    88.63    92.26
Salary    40.63    41.58    42.17    43.22    44.90    46.48    46.69
        8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X         94.96    96.28     97.26     97.98     98.65     99.15     99.47
Salary    46.75    46.86     47.76     47.82     47.85     48.10     50.40
        15 comps  16 comps  17 comps  18 comps  19 comps
X          99.75     99.89     99.97     99.99    100.00
Salary     50.55     53.01     53.85     54.61     54.61
```

CV score가 각각의 가능한 주성분 개수에 따라서 표시가 된다. 이를 내장함수를 통해 그래

프로 그릴 수도 있다.

```
validationplot(pcr.fit,val.type="MSEP")
```



위 그래프를 통해 주성분 개수가 16개일 때 가장 작은 MSE를 보였지만 이는 원래 변수의 개수인 19개와 거의 차이가 나질 않는다. 주성분 개수가 1개일 때를 보자. 이때의 MSE는 16개일 때의 그것과 거의 차이가 나지를 않는다. 차원을 줄여서 적은 수의 변수로 회귀를 시행하고자 하는 목적이라면, MSE가 아주 조금 높더라도 변수의 개수가 낮은 모델을 선택하는 것이 나을 것이다.

summary() 함수는 또한 주성분 개수에 따른 percentage of variance explained을 보여준다. 이 개념은 10단원에서 더 상세하게 논의될 것이다. 간단하게 해당 주성분 개수를 사용할 때, 예측변수 또는 반응변수의 정보를 잡아내는 정도라고 보면 된다.

이제 PCR을 3 way hold out method을 이용하여 주성분 개수를 결정하고, 최종 test MSE를 산출하여 앞서 도출한 ridge, lasso의 그것과 비교해보자. 마찬가지로, 내장 함수로써 cross-validation을 시행해주는 함수가 있기 때문에, 이를 이용하여 CV를 시행하므로 굳이 훈련 세트를 또 다시 훈련세트와 검증세트로 나눌 필요는 없다.

```
pcr.fit = pcr(Salary~., data=Hitters[train,],scale=TRUE,validation="CV")
validationplot(pcr.fit,val.type="MSEP")
```

가장 작은 CV error가 주성분이 7개일 때 발생함을 확인할 수 있다. 그렇다면 이를 이용하여 test MSE를 계산해보자.

```
pcr.fit = pcr(Salary~., data=Hitters[train,],scale=TRUE,ncomp=7)
pcr.pred = predict(pcr.fit,x[test,])
mean((pcr.pred-y.test)^2)


[1] 96970.05
```

주성분 7개로 회귀를 실시한 PCR의 test MSE는 96970으로 나타났다. 이는 앞서 ridge와 lasso의 test MSE와 크게 차이가 나지 않는 값이다. 하지만 차원을 줄였다고 해도, 결국 주성분은 원래 변수의 선형 결합으로 만들어진 것이기 때문에, 변수선택이 일어지 않았으며 오히려 원래 변수의 선형결합으로 인해서 해석이 더 어려워졌다고 볼 수 있다. 마지막으로 모든 데이터를 이용해서 PCR를 적합해보자.

```
pcr.fit = pcr(y~x, scale=TRUE, ncomp=7)
summary(pcr.fit)
```

```
Data:  X dimension: 263 19
Y dimension: 263 1
Fit method: svdpc
Number of components considered: 7
TRAINING: % variance explained
   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X    38.31    60.16    70.84    79.03    84.29    88.63    92.26
y    40.63    41.58    42.17    43.22    44.90    46.48    46.69
```

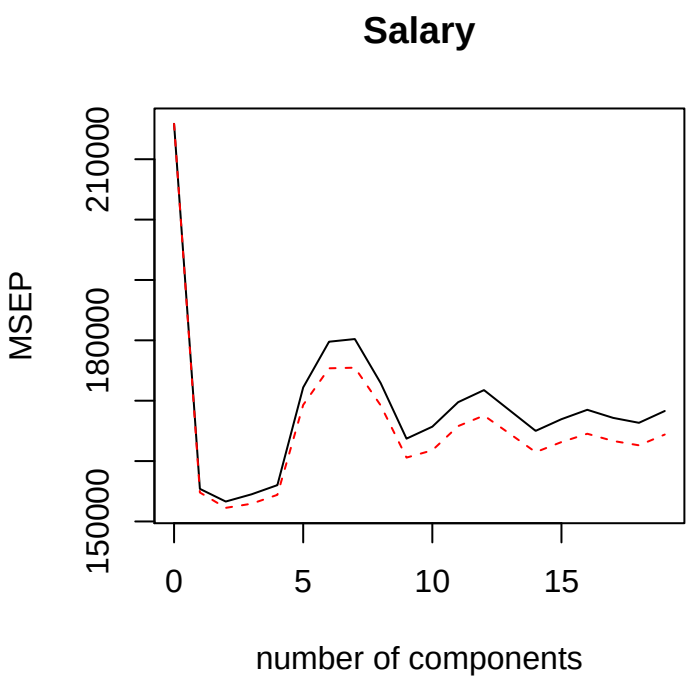
7. Partial Least Squares

PLS는 pls library에 있는 plsr() 함수를 통해 실행할 수 있다. 구문은 pcr() 함수와 유사하다.

```
set.seed(1)
pls.fit = plsr(Salary~., data=Hitters[train,], scale=TRUE, validation="CV")
summary(pls.fit)
```

```
Data:  X dimension: 131 19
Y dimension: 131 1
Fit method: kernelpls
Number of components considered: 19

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           464.6    394.2    391.5    393.1    395.0    415.0    424.0
adjCV        464.6    393.4    390.2    391.1    392.9    411.5    418.8
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       424.5    415.8    404.6     407.1     412.0     414.4     410.3
adjCV    418.9    411.4    400.7     402.2     407.2     409.3     405.6
       14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
CV        406.2     408.6     410.5     408.8     407.8     410.2
```

```
adjCV     401.8     403.9     405.6     404.1     403.2     405.5

TRAINING: % variance explained
        1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X         38.12    53.46    66.05    74.49    79.33    84.56    87.09
Salary    33.58    38.96    41.57    42.43    44.04    45.59    47.05
        8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X         90.74    92.55     93.94     97.23     97.88     98.35     98.85
Salary    47.53    48.42     49.68     50.04     50.54     50.78     50.92
        15 comps  16 comps  17 comps  18 comps  19 comps
X          99.11     99.43     99.78     99.99    100.00
Salary     51.04     51.11     51.15     51.16     51.18

validationplot(pls.fit,val.type="MSEP")
```



주성분 2개를 사용 했을 때, 가작 작은 CV error가 나왔다. 이를 이용하여 test MSE를 계산하자.

```
pls.fit = plsr(Salary~., data=Hitters[train,],scale=TRUE, ncomp=2)
pls.pred = predict(pls.fit,x[test,])
mean((pls.pred-y.test)^2)


[1] 95809.61
```

이제 전체 데이터를 이용하여 pls를 적합해보자.

```
plst.fit = plsr(Salary~., data=Hitters,scale=TRUE,ncomp=2)
summary(pls.fit)


Data:  X dimension: 131 19
Y dimension: 131 1
Fit method: kernelpls
Number of components considered: 2
TRAINING: % variance explained
        1 comps  2 comps
X         38.12    53.46
Salary    33.58    38.96
```