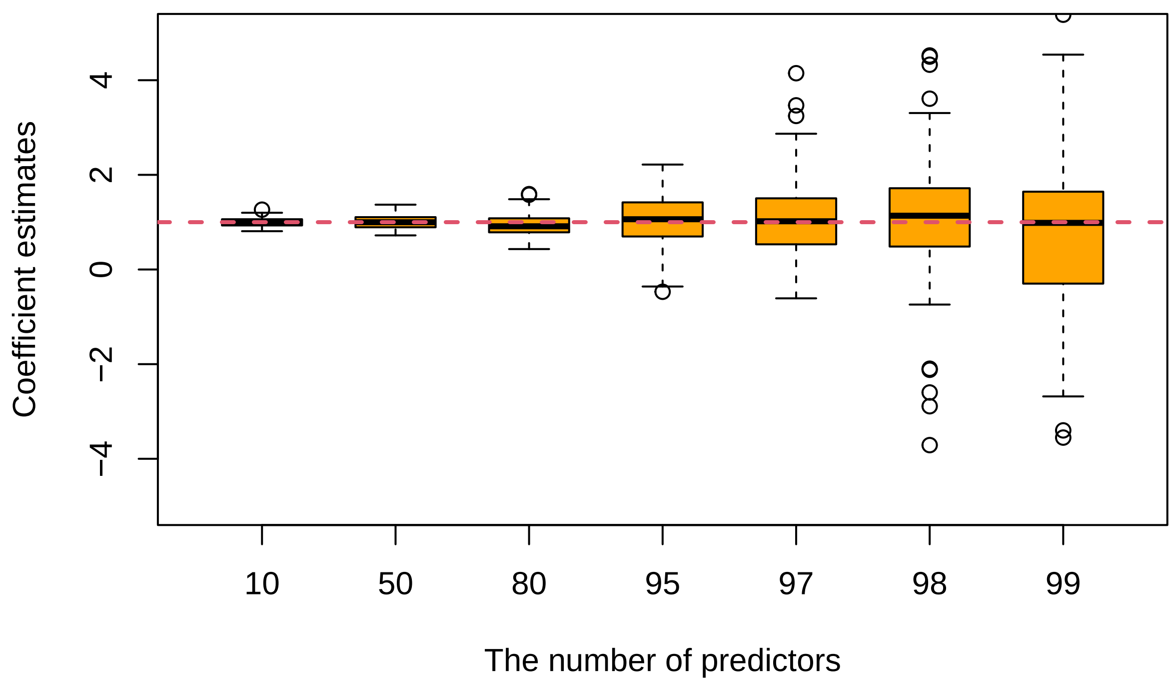
Lecture02

JongCheolLee

2024-09-23

```r
set.seed(123)
n <- 100
pp <- c(10, 50, 80, 95, 97, 98, 99) #
B <- matrix(0, 100, length(pp))

for (i in 1:100) { # 100
  for (j in 1:length(pp)) {
    beta <- rep(0, pp[j])
    beta[1] <- 1 # Beta1         0. Bias = E(Beta1_hat) - 1
    x <- matrix(rnorm(n*pp[j]), n, pp[j])
    y <- x %*% beta + rnorm(n) # True Linear Model
    g <- lm(y~x) # Estimation
    B[i,j] <- g$coef[2] #       , Beta1_hat
  }
}

boxplot(B, col="orange", boxwex=0.6, ylab="Coefficient estimates",
names=pp, xlab="The number of predictors", ylim=c(-5,5))
abline(h=1, col=2, lty=2, lwd=2)
```




```r
apply(B, 2, mean)
```

```
## [1] 1.0005277 1.0057437 0.9432175 1.0358624 1.0855411 1.0727110 0.9223116
```

- LSE의 unbiased 성질 덕분에 모두 평균은 1에 근접함

```r
apply(B, 2, var)
```

```
## [1]  0.008622572  0.022278710  0.054041523  0.301848895  0.685693276
## [6]  5.941321509 38.018106760
```

- 하지만 변수가 많아질수록 B1의 추정치의 분산이 매우 커진다.

Best Subset Selection

```r
library(ISLR)
```

```
## Warning:    'ISLR'  R   4.2.3
```

```r
names(Hitters)
```

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

```r
dim(Hitters)
```

```
## [1] 322  20
```

```r
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

```r
  #
```

```r
Hitters <- na.omit(Hitters)
```

```r
dim(Hitters)
```

```
## [1] 263  20
```

```r
sum(is.na(Hitters))
```

```
## [1] 0
```

```r
library(leaps)
```

```
## Warning:    'leaps'  R   4.2.3
```

```r
fit <- regsubsets(Salary ~ ., Hitters)
  # best regression subset
summary(fit)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., 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 CRBI
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*" 
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " " 
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " " 
##          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 ) " "    " "     " "       " "     " "     " "    " "       
## 2  ( 1 ) " "    " "     " "       " "     " "     " "    " "       
## 3  ( 1 ) " "    " "     " "       "*"     " "     " "    " "       
## 4  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "       
## 5  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "       
## 6  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "       
## 7  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "       
## 8  ( 1 ) "*"    " "     "*"       "*"     " "     " "    " "
```

```r
sg <- summary(fit)
names(sg)
```

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

```r
dim(sg$which)
```

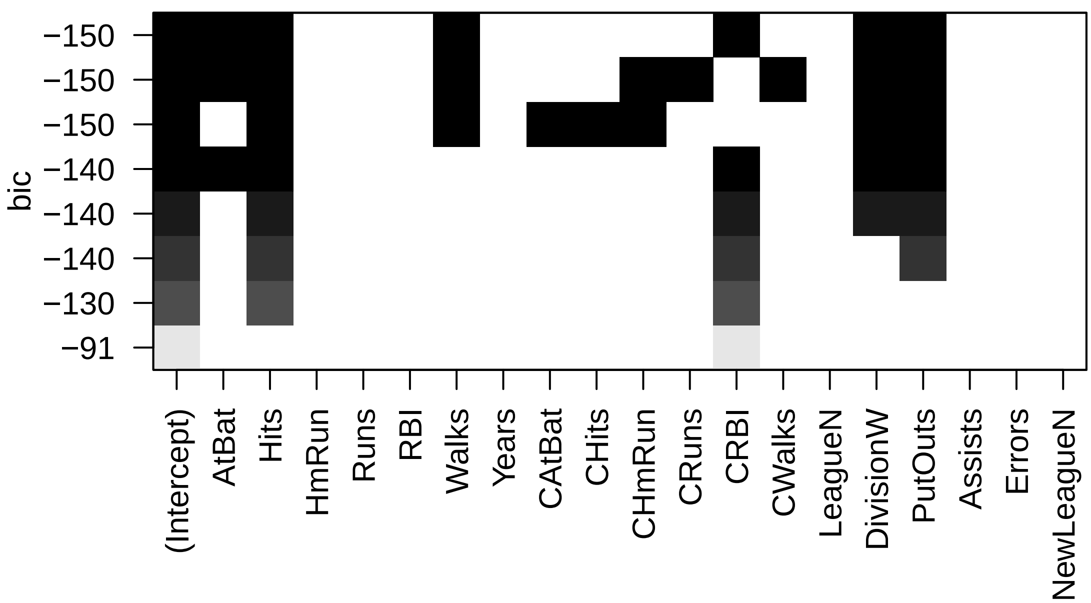
```
## [1]  8 20
```

```r
sg$which
```

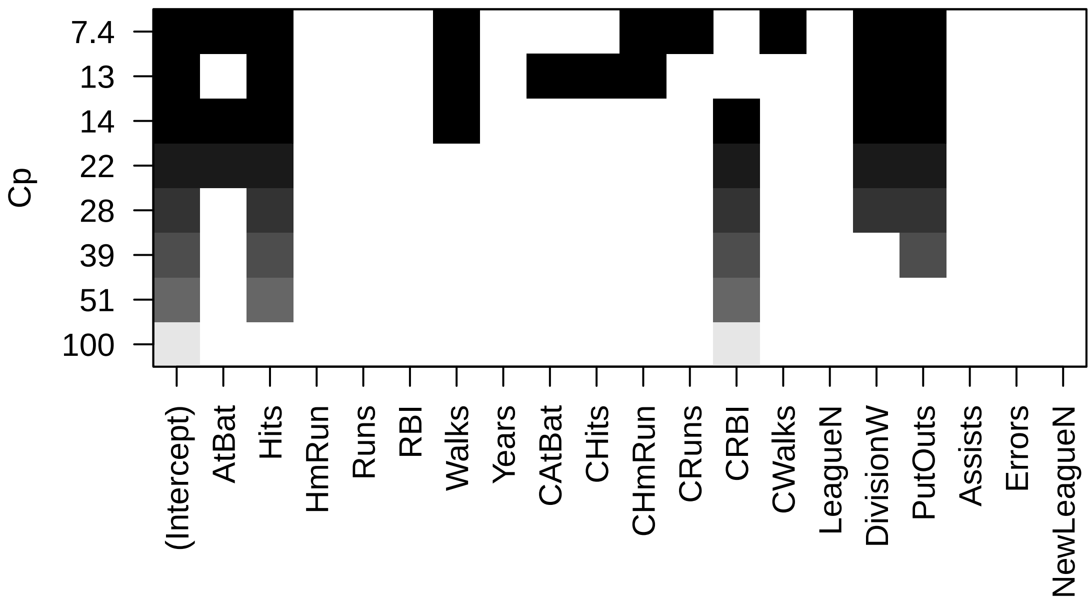
```
##   (Intercept) AtBat  Hits HmRun  Runs   RBI Walks Years CAtBat CHits CHmRun
## 1        TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 2        TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 3        TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 4        TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 5        TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 6        TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  FALSE FALSE  FALSE
## 7        TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE   TRUE  TRUE   TRUE
## 8        TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  FALSE FALSE   TRUE
##   CRuns  CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
```

```
## 1 FALSE  TRUE  FALSE   FALSE     FALSE   FALSE   FALSE  FALSE      FALSE
## 2 FALSE  TRUE  FALSE   FALSE     FALSE   FALSE   FALSE  FALSE      FALSE
## 3 FALSE  TRUE  FALSE   FALSE     FALSE    TRUE   FALSE  FALSE      FALSE
## 4 FALSE  TRUE  FALSE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 5 FALSE  TRUE  FALSE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 6 FALSE  TRUE  FALSE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 7 FALSE FALSE  FALSE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 8  TRUE FALSE   TRUE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
```

```r
plot(fit) # default: bic
```




```r
plot(fit, scale="Cp")
```




```
big <- regsubsets(Salary ~ ., data=Hitters, nvmax=19, nbest=10)
  #    8      , 19
  #     10
sg <- summary(big)
dim(sg$which)
```

```
## [1] 181  20
```

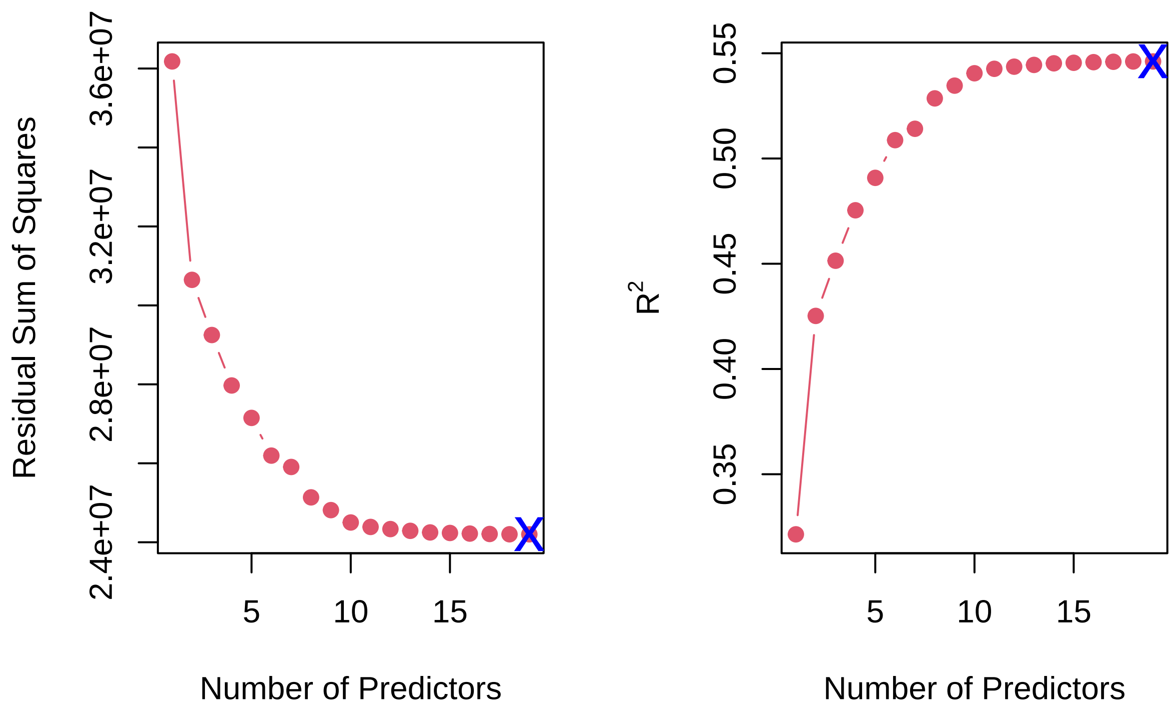
- best를 뽑는 방식: rss, rsq

```
sg.size <- as.numeric(rownames(sg$which))
table(sg.size)
```

```
## sg.size
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
## 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10  1
```

```
sg.rss <- tapply(sg$rss, sg.size, min)
w1 <- which.min(sg.rss)
sg.rsq <- tapply(sg$rsq, sg.size, max)
w2 <- which.max(sg.rsq)
par(mfrow=c(1,2))
plot(1:19, sg.rss, type="b", xlab="Number of Predictors",
ylab="Residual Sum of Squares", col=2, pch=19)
```

```r
points(w1, sg.rss[w1], pch="x", col="blue", cex=2)
plot(1:19, sg.rsq, type="b", xlab="Number of Predictors",
ylab=expression(R^2), col=2, pch=19)
points(w2, sg.rsq[w2], pch="x", col="blue", cex=2)
```



- RSS, R^2의 특징은 모델에 포함되는 변수를 늘릴수록 무조건 줄어든다.
- 따라서 서로 다른 모델 간의 비교에는 부적절하다.
- 하지만 같은 모델 내에서의 비교에는 적절한 방법

Forward, Backward selection

```r
g.full <- regsubsets(Salary ~., data=Hitters)
g.forw <- regsubsets(Salary ~., data=Hitters, method="forward")
g.back <- regsubsets(Salary ~., data=Hitters, method="backward")
full <- summary(g.full)$which[,-1]
full[full==TRUE] <- 1
forw <- summary(g.forw)$which[,-1]
forw[forw==TRUE] <- 1
back <- summary(g.back)$which[,-1]
back[back==TRUE] <- 1
```

```
full
```

```
##   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI CWalks
## 1     0    0     0    0   0     0     0      0     0      0     0    1      0
## 2     0    1     0    0   0     0     0      0     0      0     0    1      0
## 3     0    1     0    0   0     0     0      0     0      0     0    1      0
## 4     0    1     0    0   0     0     0      0     0      0     0    1      0
## 5     1    1     0    0   0     0     0      0     0      0     0    1      0
## 6     1    1     0    0   0     1     0      0     0      0     0    1      0
## 7     0    1     0    0   0     1     0      1     1      1     0    0      0
## 8     1    1     0    0   0     1     0      0     0      1     1    0      1
##   LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1       0         0       0       0      0          0
## 2       0         0       0       0      0          0
## 3       0         0       1       0      0          0
## 4       0         1       1       0      0          0
## 5       0         1       1       0      0          0
## 6       0         1       1       0      0          0
## 7       0         1       1       0      0          0
## 8       0         1       1       0      0          0
```

```
forw
```

```
##   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI CWalks
## 1     0    0     0    0   0     0     0      0     0      0     0    1      0
## 2     0    1     0    0   0     0     0      0     0      0     0    1      0
## 3     0    1     0    0   0     0     0      0     0      0     0    1      0
## 4     0    1     0    0   0     0     0      0     0      0     0    1      0
## 5     1    1     0    0   0     0     0      0     0      0     0    1      0
## 6     1    1     0    0   0     1     0      0     0      0     0    1      0
## 7     1    1     0    0   0     1     0      0     0      0     0    1      1
## 8     1    1     0    0   0     1     0      0     0      0     1    1      1
##   LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1       0         0       0       0      0          0
## 2       0         0       0       0      0          0
## 3       0         0       1       0      0          0
## 4       0         1       1       0      0          0
## 5       0         1       1       0      0          0
## 6       0         1       1       0      0          0
## 7       0         1       1       0      0          0
## 8       0         1       1       0      0          0
```

```
back
```

```
##   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI CWalks
## 1     0    0     0    0   0     0     0      0     0      0     1    0      0
## 2     0    1     0    0   0     0     0      0     0      0     1    0      0
## 3     0    1     0    0   0     0     0      0     0      0     1    0      0
## 4     1    1     0    0   0     0     0      0     0      0     1    0      0
## 5     1    1     0    0   0     1     0      0     0      0     1    0      0
## 6     1    1     0    0   0     1     0      0     0      0     1    0      0
## 7     1    1     0    0   0     1     0      0     0      0     1    0      1
```

```
## 8      1     1      0     0    0      1      0       0      0       0      1     1       1
##   LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1       0         0       0       0      0          0
## 2       0         0       0       0      0          0
## 3       0         0       1       0      0          0
## 4       0         0       1       0      0          0
## 5       0         0       1       0      0          0
## 6       0         1       1       0      0          0
## 7       0         1       1       0      0          0
## 8       0         1       1       0      0          0
```

```
coef(g.full, 1:5)
```

```
## [[1]]
## (Intercept)        CRBI
## 274.5803864   0.7909536
##
## [[2]]
## (Intercept)        Hits        CRBI
## -47.9559022   3.3008446   0.6898994
##
## [[3]]
## (Intercept)        Hits        CRBI     PutOuts
## -71.4592204   2.8038162   0.6825275   0.2735814
##
## [[4]]
##  (Intercept)         Hits         CRBI    DivisionW      PutOuts
##   13.9231044    2.6757978    0.6817790 -139.9538855    0.2735002
##
## [[5]]
##  (Intercept)        AtBat         Hits         CRBI    DivisionW      PutOuts
##   97.7684116   -1.4401428    7.1753197    0.6882079 -129.7319386    0.2905164
```

```
coef(g.forw, 1:5)
```

```
## [[1]]
## (Intercept)        CRBI
## 274.5803864   0.7909536
##
## [[2]]
## (Intercept)        Hits        CRBI
## -47.9559022   3.3008446   0.6898994
##
## [[3]]
## (Intercept)        Hits        CRBI     PutOuts
## -71.4592204   2.8038162   0.6825275   0.2735814
##
## [[4]]
##  (Intercept)         Hits         CRBI    DivisionW      PutOuts
##   13.9231044    2.6757978    0.6817790 -139.9538855    0.2735002
##
## [[5]]
##  (Intercept)        AtBat         Hits         CRBI    DivisionW      PutOuts
```

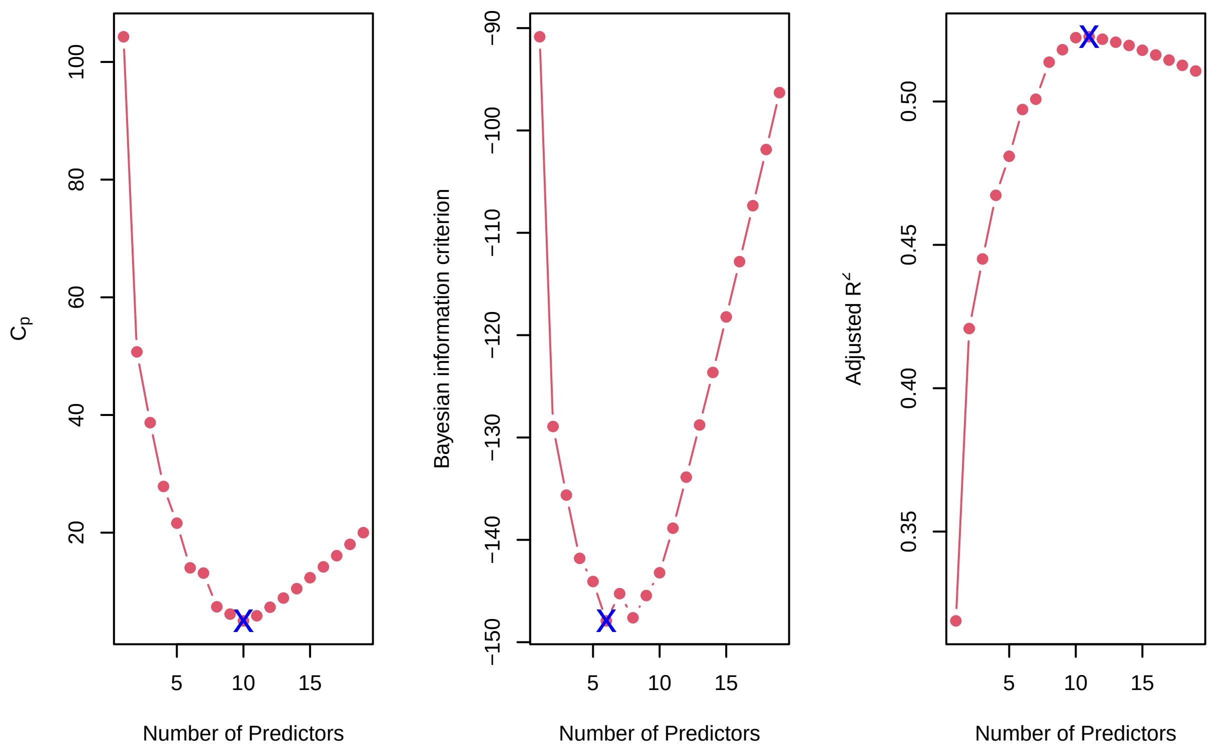
```
##   97.7684116   -1.4401428    7.1753197    0.6882079 -129.7319386    0.2905164
```

```r
coef(g.back, 1:5)
```

```
## [[1]]
## (Intercept)       CRuns
## 259.0822757   0.7664116
##
## [[2]]
## (Intercept)        Hits       CRuns
## -50.8174029   3.2257212   0.6614168
##
## [[3]]
## (Intercept)        Hits       CRuns     PutOuts
## -79.3969049   2.6431989   0.6648928   0.3100558
##
## [[4]]
## (Intercept)       AtBat        Hits       CRuns     PutOuts
##  17.5690044  -1.5476983   7.4695315   0.6702041   0.3286804
##
## [[5]]
## (Intercept)       AtBat        Hits       Walks       CRuns     PutOuts
##  16.3605866  -1.9686325   7.8904780   3.6851865   0.6218929   0.3000029
```

Cp(AIC), BIC, adj_R^2

```r
sg.cp <- tapply(sg$cp, sg.size, min)
w3 <- which.min(sg.cp)
sg.bic <- tapply(sg$bic, sg.size, min)
w4 <- which.min(sg.bic)
sg.adjr2 <- tapply(sg$adjr2, sg.size, max)
w5 <- which.max(sg.adjr2)
par(mfrow=c(1,3))
plot(1:19, sg.cp, type="b", xlab ="Number of Predictors",
ylab=expression(C[p]), col=2, pch=19)
points(w3, sg.cp[w3], pch="x", col="blue", cex=2)
plot(1:19, sg.bic, type="b", xlab ="Number of Predictors",
ylab="Bayesian information criterion", col=2, pch=19)
points(w4, sg.bic[w4], pch="x", col="blue", cex=2)
plot(1:19, sg.adjr2, type="b", xlab ="Number of Predictors",
ylab=expression(paste("Adjusted ", R^2)), col=2, pch=19)
points(w5, sg.adjr2[w5], pch="x", col="blue", cex=2)
```




```r
model1 <- coef(big, which.min(sg$rss))
model2 <- coef(big, which.max(sg$rsq))
model3 <- coef(big, which.max(sg$adjr2))
model4 <- coef(big, which.min(sg$cp))
model5 <- coef(big, which.min(sg$bic))
RES <- matrix(0, 20, 5)
rownames(RES) <- names(model1)
colnames(RES) <- c("rss", "rsq", "adjr2", "cp", "bic")
for (i in 1:5) {
  model <- get(paste("model", i, sep=""))
  w <- match(names(model), rownames(RES))
  RES[w, i] <- model
}
RES
```

```
##                    rss          rsq       adjr2           cp         bic
## (Intercept) 163.1035878  163.1035878 135.7512195 162.5354420  91.5117981
## AtBat        -1.9798729   -1.9798729  -2.1277482  -2.1686501  -1.8685892
## Hits          7.5007675    7.5007675   6.9236994   6.9180175   7.6043976
## HmRun         4.3308829    4.3308829   0.0000000   0.0000000   0.0000000
## Runs         -2.3762100   -2.3762100   0.0000000   0.0000000   0.0000000
## RBI          -1.0449620   -1.0449620   0.0000000   0.0000000   0.0000000
## Walks         6.2312863    6.2312863   5.6202755   5.7732246   3.6976468
## Years        -3.4890543   -3.4890543   0.0000000   0.0000000   0.0000000
## CAtBat       -0.1713405   -0.1713405  -0.1389914  -0.1300798   0.0000000
## CHits         0.1339910    0.1339910   0.0000000   0.0000000   0.0000000
```

```
## CHmRun        -0.1728611   -0.1728611    0.0000000    0.0000000    0.0000000
## CRuns          1.4543049    1.4543049    1.4553310    1.4082490    0.0000000
## CRBI           0.8077088    0.8077088    0.7852528    0.7743122    0.6430169
## CWalks        -0.8115709   -0.8115709   -0.8228559   -0.8308264    0.0000000
## LeagueN       62.5994230   62.5994230   43.1116152    0.0000000    0.0000000
## DivisionW   -116.8492456 -116.8492456 -111.1460252 -112.3800575 -122.9515338
## PutOuts        0.2818925    0.2818925    0.2894087    0.2973726    0.2643076
## Assists        0.3710692    0.3710692    0.2688277    0.2831680    0.0000000
## Errors        -3.3607605   -3.3607605    0.0000000    0.0000000    0.0000000
## NewLeagueN   -24.7623251  -24.7623251    0.0000000    0.0000000    0.0000000
```

```
apply(RES, 2, function(t) sum(t!=0)-1)
```

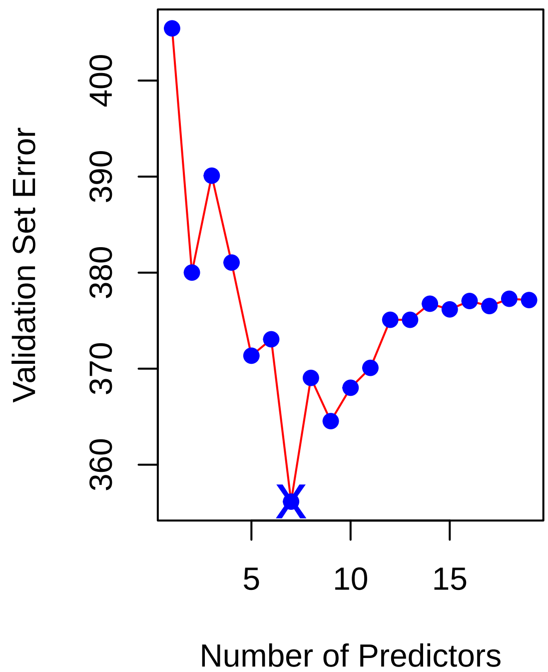
```
##   rss   rsq adjr2    cp   bic
##    19    19    11    10     6
```

Validation Set

```
set.seed(1)
train <- sample(c(TRUE, FALSE), nrow(Hitters), replace=TRUE)
test <- (!train)
g1 <- regsubsets(Salary ~ ., data=Hitters[train, ], nvmax=19)
test.mat <- model.matrix(Salary~., data=Hitters[test, ])
val.errors <- rep(NA, 19)

for (i in 1:19) {
  coefi <- coef(g1, id=i)
  pred <- test.mat[, names(coefi)] %*% coefi
  val.errors[i] <- sqrt(mean((Hitters$Salary[test]-pred)^2))
}
val.errors
```

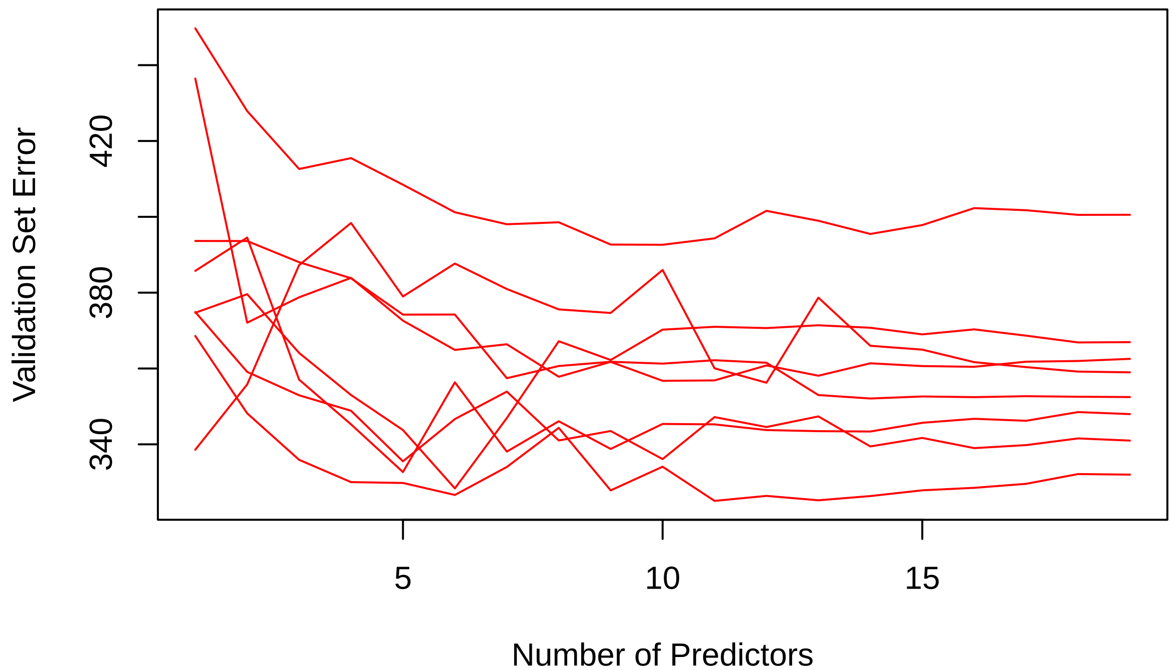
```
##  [1] 405.4347 380.0072 390.0970 381.0491 371.3517 373.0627 356.1586 369.0412
##  [9] 364.5403 368.0148 370.0855 375.0932 375.0879 376.7641 376.1757 377.0469
## [17] 376.5201 377.2791 377.1447
```

```
w <- which.min(val.errors)
par(mfrow=c(1,2))
plot(1:19, val.errors, type="l", col="red",
xlab="Number of Predictors", ylab="Validation Set Error")
points(1:19, val.errors, pch=19, col="blue")
points(w, val.errors[w], pch="x", col="blue", cex=2)
```



K-fold validation

```
set.seed(1234)
N <- 8
ERR <- matrix(0, 19, N)
for (k in 1:N) {
  tr <- sample(c(TRUE, FALSE), nrow(Hitters), replace=TRUE)
  tt <- (!tr)
  g <- regsubsets(Salary ~ ., data=Hitters[tr, ], nvmax=19)
  tt.mat <- model.matrix(Salary~., data=Hitters[tt, ])
  for (i in 1:19) {
    coefi <- coef(g, id=i)
    pred <- tt.mat[, names(coefi)] %*% coefi
    ERR[i,k] <- sqrt(mean((Hitters$Salary[tt]-pred)^2))
  }
}
matplot(ERR, type="l", col="red", xlab="Number of Predictors",
lty=1, ylab="Validation Set Error")
```



- Validation Set이 바뀔 때마다 매우 다른 결과
- 해결책은 교차검증!

```
apply(ERR, 2, which.min)
```

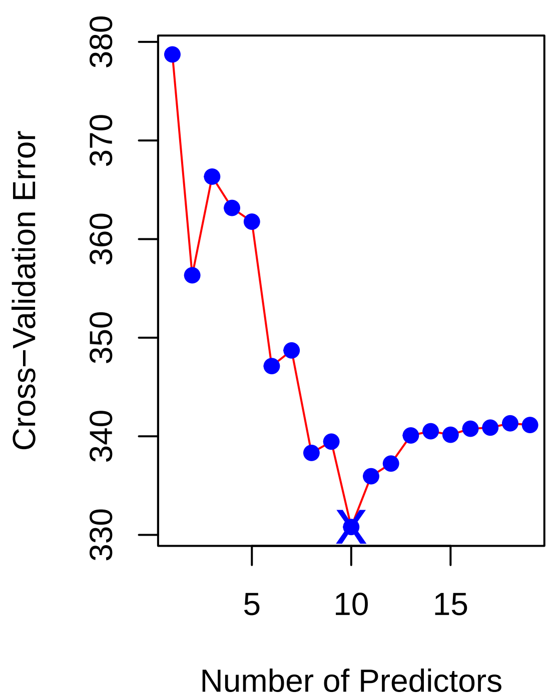
```
## [1] 11  5 14 10 10  6  1  5
```

```
## Define new "predict" function on regsubset
predict.regsubsets <- function(object, newdata, id, ...) {
form <- as.formula(object$call[[2]])
mat <- model.matrix(form, newdata)
coefi <- coef(object, id=id)
xvars <- names(coefi)
mat[, xvars] %*% coefi
}
set.seed(1)
K <- 10
n <- nrow(Hitters)
fd <- sample(rep(1:K, length=n))
cv.errors <- matrix(NA , n, 19, dimnames=list(NULL, paste(1:19)))

for (i in 1:K) {
  fit <- regsubsets(Salary~., Hitters[fd!=i, ], nvmax=19)
  for (j in 1:19) {
    pred <- predict(fit, Hitters[fd==i, ], id=j)
```

```r
    cv.errors[fd==i, j] <- (Hitters$Salary[fd==i]-pred)^2
  }
}

sqrt(apply(cv.errors, 2, mean))
```

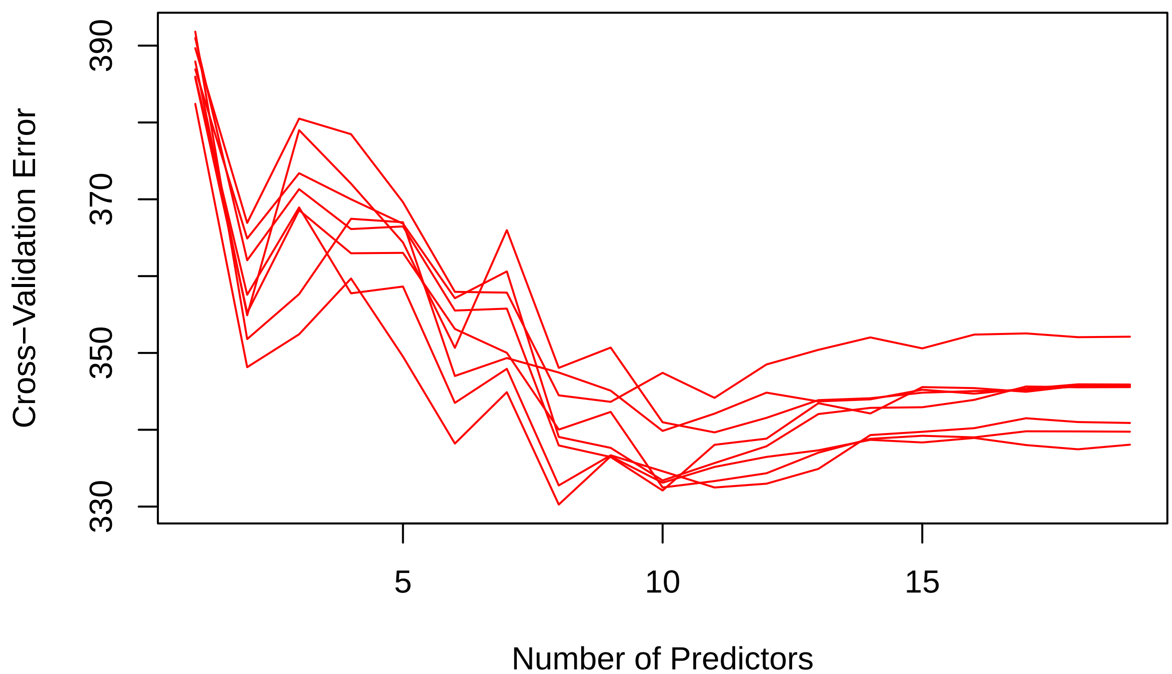
```
##        1        2        3        4        5        6        7        8
## 378.7273 356.3299 366.3377 363.1608 361.7680 347.1194 348.7096 338.3155
##        9       10       11       12       13       14       15       16
## 339.4583 330.8001 335.9470 337.2333 340.0865 340.5136 340.1557 340.7668
##       17       18       19
## 340.8814 341.3180 341.1447
```

```r
K.ERR <- sqrt(apply(cv.errors, 2, mean))
ww <- which.min(K.ERR)
par(mfrow=c(1,2))
plot(1:19, K.ERR, type="l", col="red",
xlab="Number of Predictors", ylab="Cross-Validation Error")
points(1:19, K.ERR, pch=19, col="blue")
points(ww, K.ERR[ww], pch="x", col="blue", cex=2)
```




```r
## 10-fold CV with 8 different splits
N <- 8
n <- nrow(Hitters)
```

```r
ERR <- matrix(0, 19, N)
set.seed(1234)
for (k in 1:N) {
  fd <- sample(rep(1:K, length=n))
  CVR <- matrix(NA , n, 19)
  for (i in 1:K) {
    f <- regsubsets(Salary~., data=Hitters[fd!=i, ], nvmax=19)
    for (j in 1:19) {
      pred <- predict(f, Hitters[fd==i, ], id=j)
      CVR[fd==i, j] <- (Hitters$Salary[fd==i]-pred)^2
    }
  }
  ERR[,k] <- sqrt(apply(CVR, 2, mean))
}
matplot(ERR, type="l", col="red", xlab="Number of Predictors",
lty=1, ylab="Cross-Validation Error")
```




```r
apply(ERR, 2, which.min)
```

```
## [1] 10 10 10  9 11 10 11  8
```

```r
set.seed(111)
fd <- sample(rep(1:K, length=n))
CVR.1se <- matrix(NA, n, 19)
```

```r
for (i in 1:K) {
  fit <- regsubsets(Salary~., Hitters[fd!=i, ], nvmax=19)
  for (j in 1:19) {
    pred <- predict(fit, Hitters[fd==i, ], id=j)
    CVR.1se[fd==i, j] <- Hitters$Salary[fd==i]-pred
  }
}
avg <- sqrt(apply(CVR.1se^2, 2, mean))
se <- apply(CVR.1se, 2, sd)/sqrt(n)
PE <- cbind(avg - se, avg, avg + se)
data.frame(lwr=PE[,1], mean=PE[,2], upp=PE[,3])
```

```
##         lwr     mean      upp
## 1  369.1514 393.4582 417.7651
## 2  352.7224 375.9457 399.1690
## 3  356.6445 380.1251 403.6058
## 4  348.5281 371.4749 394.4218
## 5  350.8331 373.9322 397.0313
## 6  339.3305 361.6738 384.0170
## 7  345.5366 368.2846 391.0326
## 8  330.4133 352.1664 373.9196
## 9  329.9209 351.6399 373.3590
## 10 317.9704 338.9042 359.8380
## 11 322.5224 343.7582 364.9941
## 12 324.6231 345.9976 367.3720
## 13 327.7008 349.2755 370.8502
## 14 327.2778 348.8254 370.3730
## 15 328.2571 349.8688 371.4806
## 16 328.8777 350.5306 372.1836
## 17 327.9254 349.5151 371.1048
## 18 328.4501 350.0743 371.6985
## 19 328.5078 350.1358 371.7638
```

```r
which.min(PE[,2])
```

```
## [1] 10
```

```r
w <- which.min(PE[,2])
which(PE[w, 1] < PE[,2] & PE[w, 3] > PE[,2])
```

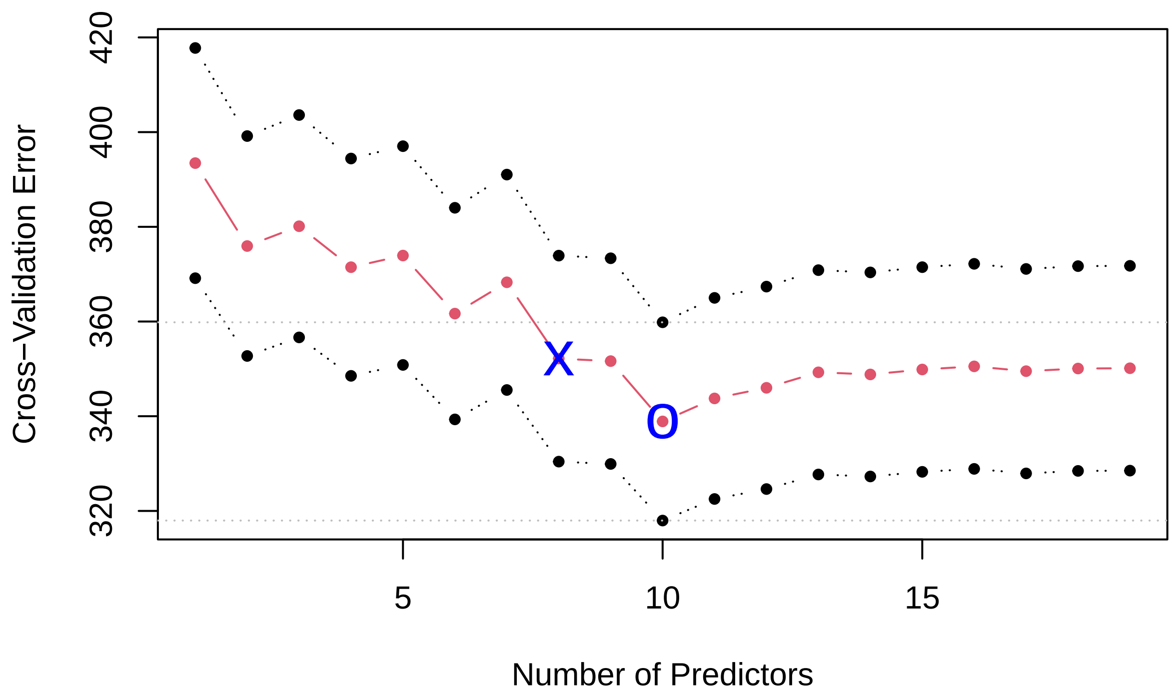
```
##  [1]  8  9 10 11 12 13 14 15 16 17 18 19
```

```r
min(which(PE[w, 1] < PE[,2] & PE[w, 3] > PE[,2]))
```

```
## [1] 8
```

```r
matplot(1:19, PE, type="b", col=c(1,2,1), lty=c(3,1,3), pch=20,
xlab="Number of Predictors", ylab="Cross-Validation Error")
abline(h=PE[w, 1], lty=3, col="gray")
abline(h=PE[w, 3], lty=3, col="gray")
```

```r
points(which.min(avg), PE[which.min(avg),2],
pch="o",col="blue",cex=2)
up <- which(PE[,2] < PE[which.min(PE[,2]),3])
points(min(up), PE[min(up),2], pch="x", col="blue", cex=2)
```



- CV Error가 가장 작은 10번째 값의 upper, lower 바운드 내에서 가장 가벼운 모델인 8번째 모델을 선택해야한다.

Shrinkage Methods

- **shrinkage** 뜻: 규제, 제약, 압축
- **Ridge**: 계수들의 L2-norm penalty를 제약식에 추가
    - 파라미터 lambda가 커질수록 제약이 강해짐.
- **Lasso**(Least absolute shrinkage and selection operator): L1-norm penalty

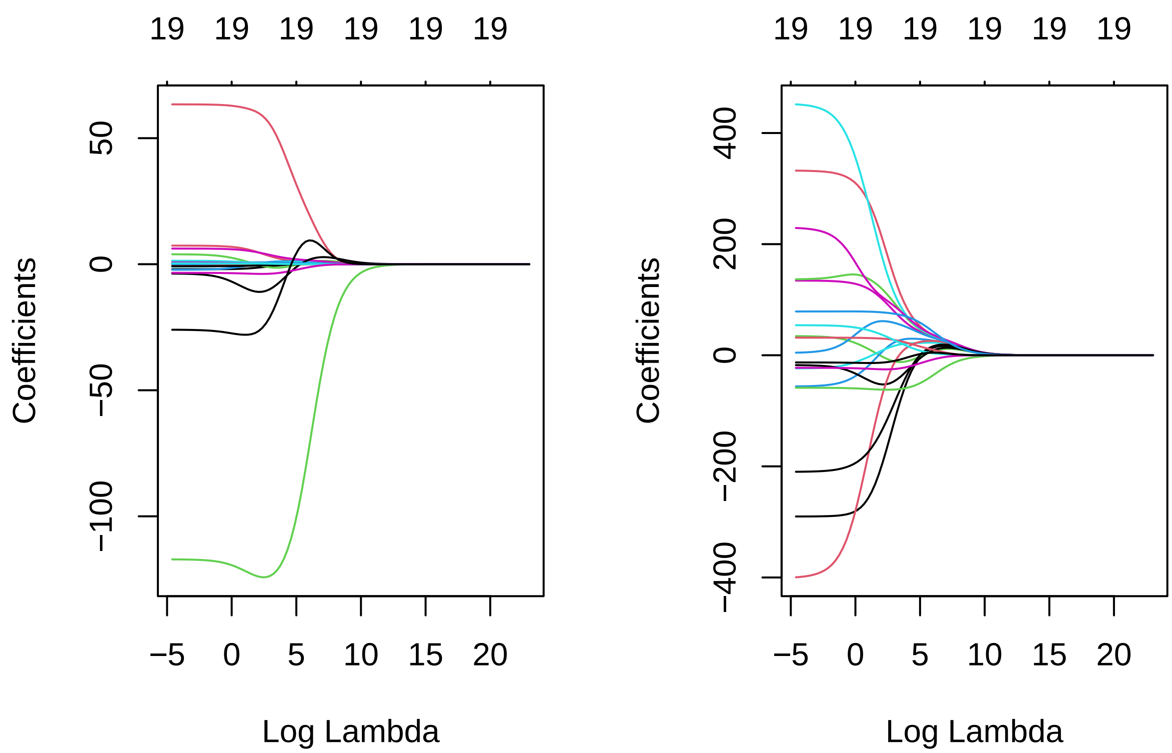
```r
library(glmnet)
```

1. Ridge regression

```
##            : Matrix
```

```
## Loaded glmnet 4.1-8
```

```r
x0 <- model.matrix(Salary~., Hitters)[, -1]
y <- Hitters$Salary

grid <- 10^seq(10, -2, length=100)
g1 <- glmnet(x0, y, alpha=0, lambda=grid)

par(mfrow=c(1,2))
plot(g1, "lambda", label=TRUE)

fun <- function(t) sqrt(var(t)*(length(t)-1)/length(t))
sdx <- matrix(apply(x0, 2, fun), dim(x0)[2], dim(x0)[1])

x <- x0/t(sdx)
g2 <- glmnet(x, y, alpha=0, lambda=grid)

plot(g2, "lambda", label=TRUE)
```



- glmnet 사용시 범주형 변수는 더미변수로 인코딩 필수
- 그래프 위의 19는 shrinkage method의 자유도(= 0이 아닌 계수의 개수)

```r
data.frame(sd_g1=apply(x0, 2, sd), sd_g2=apply(x, 2, sd))
```

```
##                       sd_g1      sd_g2
```

```
## AtBat       147.3072088 1.001907
## Hits         45.1253259 1.001907
## HmRun         8.7571077 1.001907
## Runs         25.5398156 1.001907
## RBI          25.8827143 1.001907
## Walks        21.7180561 1.001907
## Years         4.7936159 1.001907
## CAtBat     2286.5829295 1.001907
## CHits       648.1996437 1.001907
## CHmRun       82.1975815 1.001907
## CRuns       331.1985706 1.001907
## CRBI        323.3676682 1.001907
## CWalks      264.0558680 1.001907
## LeagueN       0.5001378 1.001907
## DivisionW     0.5008628 1.001907
## PutOuts     279.9345755 1.001907
## Assists     145.0805766 1.001907
## Errors        6.6065742 1.001907
## NewLeagueN    0.4996443 1.001907
```

```r
names(g2)
```

```
##  [1] "a0"        "beta"      "df"        "dim"       "lambda"    "dev.ratio"
##  [7] "nulldev"   "npasses"   "jerr"      "offset"    "call"      "nobs"
```

```r
data.frame(lambda=g2$lambda, df=g2$df)
```

```
##            lambda df
## 1    1.000000e+10 19
## 2    7.564633e+09 19
## 3    5.722368e+09 19
## 4    4.328761e+09 19
## 5    3.274549e+09 19
## 6    2.477076e+09 19
## 7    1.873817e+09 19
## 8    1.417474e+09 19
## 9    1.072267e+09 19
## 10   8.111308e+08 19
## 11   6.135907e+08 19
## 12   4.641589e+08 19
## 13   3.511192e+08 19
## 14   2.656088e+08 19
## 15   2.009233e+08 19
## 16   1.519911e+08 19
## 17   1.149757e+08 19
## 18   8.697490e+07 19
## 19   6.579332e+07 19
## 20   4.977024e+07 19
## 21   3.764936e+07 19
## 22   2.848036e+07 19
## 23   2.154435e+07 19
## 24   1.629751e+07 19
## 25   1.232847e+07 19
```

```
## 26  9.326033e+06 19
## 27  7.054802e+06 19
## 28  5.336699e+06 19
## 29  4.037017e+06 19
## 30  3.053856e+06 19
## 31  2.310130e+06 19
## 32  1.747528e+06 19
## 33  1.321941e+06 19
## 34  1.000000e+06 19
## 35  7.564633e+05 19
## 36  5.722368e+05 19
## 37  4.328761e+05 19
## 38  3.274549e+05 19
## 39  2.477076e+05 19
## 40  1.873817e+05 19
## 41  1.417474e+05 19
## 42  1.072267e+05 19
## 43  8.111308e+04 19
## 44  6.135907e+04 19
## 45  4.641589e+04 19
## 46  3.511192e+04 19
## 47  2.656088e+04 19
## 48  2.009233e+04 19
## 49  1.519911e+04 19
## 50  1.149757e+04 19
## 51  8.697490e+03 19
## 52  6.579332e+03 19
## 53  4.977024e+03 19
## 54  3.764936e+03 19
## 55  2.848036e+03 19
## 56  2.154435e+03 19
## 57  1.629751e+03 19
## 58  1.232847e+03 19
## 59  9.326033e+02 19
## 60  7.054802e+02 19
## 61  5.336699e+02 19
## 62  4.037017e+02 19
## 63  3.053856e+02 19
## 64  2.310130e+02 19
## 65  1.747528e+02 19
## 66  1.321941e+02 19
## 67  1.000000e+02 19
## 68  7.564633e+01 19
## 69  5.722368e+01 19
## 70  4.328761e+01 19
## 71  3.274549e+01 19
## 72  2.477076e+01 19
## 73  1.873817e+01 19
## 74  1.417474e+01 19
## 75  1.072267e+01 19
## 76  8.111308e+00 19
## 77  6.135907e+00 19
## 78  4.641589e+00 19
## 79  3.511192e+00 19
```

```
## 80  2.656088e+00 19
## 81  2.009233e+00 19
## 82  1.519911e+00 19
## 83  1.149757e+00 19
## 84  8.697490e-01 19
## 85  6.579332e-01 19
## 86  4.977024e-01 19
## 87  3.764936e-01 19
## 88  2.848036e-01 19
## 89  2.154435e-01 19
## 90  1.629751e-01 19
## 91  1.232847e-01 19
## 92  9.326033e-02 19
## 93  7.054802e-02 19
## 94  5.336699e-02 19
## 95  4.037017e-02 19
## 96  3.053856e-02 19
## 97  2.310130e-02 19
## 98  1.747528e-02 19
## 99  1.321941e-02 19
## 100 1.000000e-02 19
```

```r
data.frame(log.lambda=round(log(g2$lambda), 4), df=g2$df)
```

```
##     log.lambda df
## 1      23.0259 19
## 2      22.7467 19
## 3      22.4676 19
## 4      22.1885 19
## 5      21.9094 19
## 6      21.6303 19
## 7      21.3512 19
## 8      21.0721 19
## 9      20.7930 19
## 10     20.5139 19
## 11     20.2348 19
## 12     19.9557 19
## 13     19.6766 19
## 14     19.3975 19
## 15     19.1184 19
## 16     18.8393 19
## 17     18.5602 19
## 18     18.2811 19
## 19     18.0020 19
## 20     17.7229 19
## 21     17.4438 19
## 22     17.1647 19
## 23     16.8856 19
## 24     16.6065 19
## 25     16.3274 19
## 26     16.0483 19
## 27     15.7692 19
## 28     15.4901 19
## 29     15.2110 19
```

```
## 30     14.9319 19
## 31     14.6528 19
## 32     14.3737 19
## 33     14.0946 19
## 34     13.8155 19
## 35     13.5364 19
## 36     13.2573 19
## 37     12.9782 19
## 38     12.6991 19
## 39     12.4200 19
## 40     12.1409 19
## 41     11.8618 19
## 42     11.5827 19
## 43     11.3036 19
## 44     11.0245 19
## 45     10.7454 19
## 46     10.4663 19
## 47     10.1872 19
## 48      9.9081 19
## 49      9.6290 19
## 50      9.3499 19
## 51      9.0708 19
## 52      8.7917 19
## 53      8.5126 19
## 54      8.2335 19
## 55      7.9544 19
## 56      7.6753 19
## 57      7.3962 19
## 58      7.1171 19
## 59      6.8380 19
## 60      6.5589 19
## 61      6.2798 19
## 62      6.0007 19
## 63      5.7216 19
## 64      5.4425 19
## 65      5.1634 19
## 66      4.8843 19
## 67      4.6052 19
## 68      4.3261 19
## 69      4.0470 19
## 70      3.7679 19
## 71      3.4888 19
## 72      3.2097 19
## 73      2.9306 19
## 74      2.6515 19
## 75      2.3724 19
## 76      2.0933 19
## 77      1.8142 19
## 78      1.5351 19
## 79      1.2560 19
## 80      0.9769 19
## 81      0.6978 19
## 82      0.4187 19
## 83      0.1396 19
```

```
## 84     -0.1396 19
## 85     -0.4187 19
## 86     -0.6978 19
## 87     -0.9769 19
## 88     -1.2560 19
## 89     -1.5351 19
## 90     -1.8142 19
## 91     -2.0933 19
## 92     -2.3724 19
## 93     -2.6515 19
## 94     -2.9306 19
## 95     -3.2097 19
## 96     -3.4888 19
## 97     -3.7679 19
## 98     -4.0470 19
## 99     -4.3261 19
## 100    -4.6052 19
```

```
dim(coef(g2))
```

```
## [1]  20 100
```

```
coef(g2)[, c("s0", "s10", "s20", "s30")]
```

```
## 20 x 4 sparse Matrix of class "dgCMatrix"
##                        s0           s10           s20           s30
## (Intercept)  5.359257e+02  5.359228e+02  5.358758e+02  5.351104e+02
## AtBat        8.003360e-06  1.304343e-04  2.125620e-03  3.460670e-02
## Hits         8.893439e-06  1.449404e-04  2.362028e-03  3.845827e-02
## HmRun        6.954354e-06  1.133381e-04  1.846993e-03  3.006538e-02
## Runs         8.511971e-06  1.387233e-04  2.260696e-03  3.680432e-02
## RBI          9.112034e-06  1.485028e-04  2.420054e-03  3.939536e-02
## Walks        8.998709e-06  1.466559e-04  2.389972e-03  3.890955e-02
## Years        8.122690e-06  1.323790e-04  2.157288e-03  3.511605e-02
## CAtBat       1.066656e-05  1.738376e-04  2.832917e-03  4.611551e-02
## CHits        1.112828e-05  1.813623e-04  2.955542e-03  4.811156e-02
## CHmRun       1.064214e-05  1.734395e-04  2.826419e-03  4.600702e-02
## CRuns        1.140740e-05  1.859114e-04  3.029659e-03  4.931420e-02
## CRBI         1.149433e-05  1.873281e-04  3.052739e-03  4.968778e-02
## CWalks       9.930368e-06  1.618394e-04  2.637348e-03  4.292109e-02
## LeagueN     -2.895411e-07 -4.718666e-06 -7.686756e-05 -1.243483e-03
## DivisionW   -3.902926e-06 -6.360786e-05 -1.036620e-03 -1.688637e-02
## PutOuts      6.091765e-06  9.928031e-05  1.617947e-03  2.634854e-02
## Assists      5.156775e-07  8.404231e-06  1.369608e-04  2.230188e-03
## Errors      -1.094908e-07 -1.784441e-06 -2.908560e-05 -4.749957e-04
## NewLeagueN  -5.746384e-08 -9.364259e-07 -1.523710e-05 -2.418792e-04
```

```
g2$lambda[50]
```

```
## [1] 11497.57
```

```r
coef(g2)[,50]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI 
## 407.35605020   5.43369951   6.22356733   4.58549860   5.88086202   6.19593490 
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns 
##   6.27798659   5.29979792   7.14752789   7.53950891   7.18234860   7.72864834 
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists 
##   7.79069879   6.59289871   0.04244485  -3.10715882   4.60526283   0.37837242 
##       Errors   NewLeagueN 
##  -0.13519479   0.15032295
```

```r
sqrt(sum(coef(g2)[-1, 50]^2))
```

```
## [1] 24.17063
```

```r
g2$lambda[60]
```

```
## [1] 705.4802
```

```r
coef(g2)[,60]
```

```
## (Intercept)       AtBat        Hits       HmRun        Runs         RBI 
##   54.325200   16.483353   29.555975   10.312055   23.903039   21.885732 
##       Walks       Years      CAtBat       CHits      CHmRun       CRuns 
##   28.610668   12.422480   24.725984   30.242802   27.711305   30.926412 
##        CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists 
##   31.566475   18.948465    6.830714  -27.324449   33.115519    2.325614 
##      Errors  NewLeagueN 
##   -4.639451    4.294655
```

```r
sqrt(sum(coef(g2)[-1, 60]^2))
```

```
## [1] 99.07558
```

```r
l2norm <- function(t) sqrt(sum(t^2))
l2 <- apply(coef(g2)[-1,], 2, l2norm)
data.frame(log_lambda=round(log(g2$lambda), 4),
l2norm=round(l2, 4))
```

```
##     log_lambda  l2norm
## s0     23.0259  0.0000
## s1     22.7467  0.0000
## s2     22.4676  0.0001
## s3     22.1885  0.0001
## s4     21.9094  0.0001
## s5     21.6303  0.0001
## s6     21.3512  0.0002
## s7     21.0721  0.0003
## s8     20.7930  0.0003
```

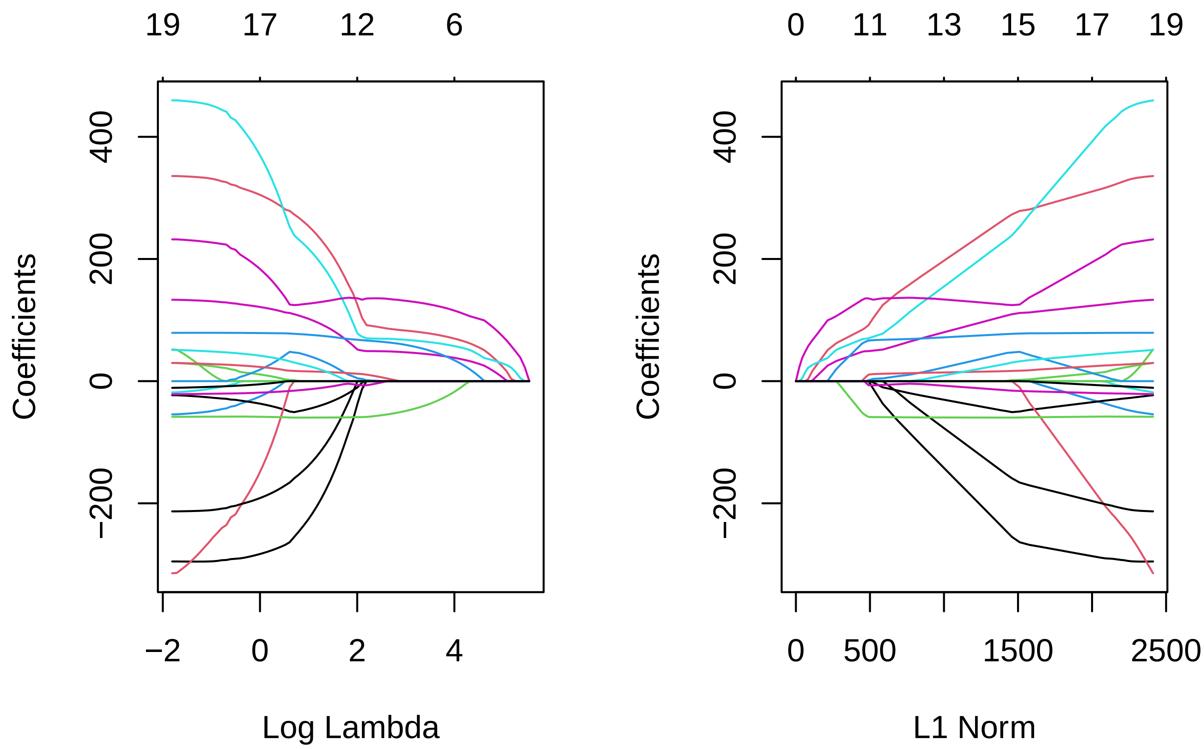
```
## s9     20.5139   0.0004
## s10    20.2348   0.0006
## s11    19.9557   0.0008
## s12    19.6766   0.0010
## s13    19.3975   0.0013
## s14    19.1184   0.0018
## s15    18.8393   0.0023
## s16    18.5602   0.0031
## s17    18.2811   0.0041
## s18    18.0020   0.0054
## s19    17.7229   0.0071
## s20    17.4438   0.0094
## s21    17.1647   0.0125
## s22    16.8856   0.0165
## s23    16.6065   0.0218
## s24    16.3274   0.0288
## s25    16.0483   0.0380
## s26    15.7692   0.0503
## s27    15.4901   0.0664
## s28    15.2110   0.0878
## s29    14.9319   0.1161
## s30    14.6528   0.1534
## s31    14.3737   0.2027
## s32    14.0946   0.2678
## s33    13.8155   0.3537
## s34    13.5364   0.4672
## s35    13.2573   0.6168
## s36    12.9782   0.8141
## s37    12.6991   1.0739
## s38    12.4200   1.4157
## s39    12.1409   1.8613
## s40    11.8618   2.4473
## s41    11.5827   3.2124
## s42    11.3036   4.2074
## s43    11.0245   5.4950
## s44    10.7454   7.1502
## s45    10.4663   9.2606
## s46    10.1872  11.9229
## s47     9.9081  15.2373
## s48     9.6290  19.2964
## s49     9.3499  24.1706
## s50     9.0708  29.8882
## s51     8.7917  36.4179
## s52     8.5126  43.6571
## s53     8.2335  51.4379
## s54     7.9544  59.5465
## s55     7.6753  67.7614
## s56     7.3962  75.8939
## s57     7.1171  83.8348
## s58     6.8380  91.5394
## s59     6.5589  99.0756
## s60     6.2798 106.5535
## s61     6.0007 114.1538
## s62     5.7216 122.1068
```

```
## s63     5.4425 130.7626
## s64     5.1634 140.4553
## s65     4.8843 151.7662
## s66     4.6052 165.3257
## s67     4.3261 181.6929
## s68     4.0470 201.4140
## s69     3.7679 224.9535
## s70     3.4888 252.3551
## s71     3.2097 283.4422
## s72     2.9306 317.9353
## s73     2.6515 355.1502
## s74     2.3724 394.3546
## s75     2.0933 434.7652
## s76     1.8142 475.6017
## s77     1.5351 516.2080
## s78     1.2560 555.8559
## s79     0.9769 594.1277
## s80     0.6978 629.9067
## s81     0.4187 663.6317
## s82     0.1396 694.1000
## s83    -0.1396 721.6804
## s84    -0.4187 745.5967
## s85    -0.6978 766.0221
## s86    -0.9769 782.8521
## s87    -1.2560 796.5702
## s88    -1.5351 807.6860
## s89    -1.8142 816.3247
## s90    -2.0933 823.0769
## s91    -2.3724 828.2603
## s92    -2.6515 832.1470
## s93    -2.9306 835.2884
## s94    -3.2097 837.5397
## s95    -3.4888 839.4284
## s96    -3.7679 840.7913
## s97    -4.0470 841.8099
## s98    -4.3261 842.6548
## s99    -4.6052 843.1611
```

2. Lasso

- Ridge regression은 중요한 변수를 선택하는 subset selection이 어렵다.
- Ridge와는 달리 계수가 정확히 0이 될 수 있다.
    - Lasso에서 작은 모델은 0인 계수가 더 많은 모델
- 필요없는 계수를 0으로 보냄으로써 변수 선택을 한다.
- 이 방법에선 optimal model을 찾는 게 optimal lambda를 찾는 문제임.

```r
g3 <- glmnet(x, y, alpha=1)
par(mfrow=c(1,2))
plot(g3, "lambda", label=TRUE)
plot(g3, "norm", label=TRUE)
```



- 왼쪽 그래프를 보면 lambda가 아주 큰 값에서 작아지면서 0이 아닌 계수가 하나씩 등장함을 알 수 있다.

```r
dim(coef(g3))
```

```
## [1] 20 80
```

```r
coef(g3)[, c("s0", "s10", "s40", "s60")]
```

```
## 20 x 4 sparse Matrix of class "dgCMatrix"
##                   s0          s10         s40          s60
## (Intercept) 535.9259 222.0974180  40.652951   153.262451
## AtBat              .           .  -84.358820 -283.848789
## Hits               .  50.8499527 158.005411   306.337268
## HmRun              .           .           .   11.386077
## Runs               .           .           .  -26.445699
## RBI                .           .           .           .
## Walks              .  25.4064700  64.595375  121.943910
## Years              .           .  -19.691966  -36.282468
## CAtBat             .           .           . -155.199102
## CHits              .           .           .           .
## CHmRun             .           .   10.478327   17.456030
## CRuns              .  37.9218169 112.825764  375.328259
## CRBI               .  99.5844116 136.599052  186.323253
## CWalks             .           .  -34.517519 -192.606975
```

```
## LeagueN       .         .          13.103378   23.517927
## DivisionW     .         .         -59.253067  -58.306307
## PutOuts       .         0.4926175  68.973044   78.789863
## Assists       .         .           .          42.115015
## Errors        .         .          -4.200133  -18.955204
## NewLeagueN    .         .           .          -5.314774
```

```r
data.frame(lambda=g3$lambda, df=g3$df)
```

```
##         lambda df
## 1  255.2820965  0
## 2  232.6035386  1
## 3  211.9396813  2
## 4  193.1115442  2
## 5  175.9560468  3
## 6  160.3245966  4
## 7  146.0818013  4
## 8  133.1042967  4
## 9  121.2796778  4
## 10 110.5055255  4
## 11 100.6885192  5
## 12  91.7436287  5
## 13  83.5933775  5
## 14  76.1671723  5
## 15  69.4006906  6
## 16  63.2353245  6
## 17  57.6176726  6
## 18  52.4990774  6
## 19  47.8352040  6
## 20  43.5856563  6
## 21  39.7136268  6
## 22  36.1855776  6
## 23  32.9709506  6
## 24  30.0419022  6
## 25  27.3730624  6
## 26  24.9413150  6
## 27  22.7255973  6
## 28  20.7067179  6
## 29  18.8671902  6
## 30  17.1910810  7
## 31  15.6638727  7
## 32  14.2723374  7
## 33  13.0044223  9
## 34  11.8491453  9
## 35  10.7964999  9
## 36   9.8373686  9
## 37   8.9634439  9
## 38   8.1671562 11
## 39   7.4416086 11
## 40   6.7805166 12
## 41   6.1781542 12
## 42   5.6293040 13
## 43   5.1292121 13
## 44   4.6735471 13
```

```
## 45   4.2583620 13
## 46   3.8800609 13
## 47   3.5353670 13
## 48   3.2212947 13
## 49   2.9351238 13
## 50   2.6743755 13
## 51   2.4367913 13
## 52   2.2203135 14
## 53   2.0230670 15
## 54   1.8433433 15
## 55   1.6795857 17
## 56   1.5303760 17
## 57   1.3944216 17
## 58   1.2705450 17
## 59   1.1576733 17
## 60   1.0548288 17
## 61   0.9611207 17
## 62   0.8757374 17
## 63   0.7979393 17
## 64   0.7270526 17
## 65   0.6624632 18
## 66   0.6036118 18
## 67   0.5499886 18
## 68   0.5011291 17
## 69   0.4566102 18
## 70   0.4160462 18
## 71   0.3790858 18
## 72   0.3454089 18
## 73   0.3147237 18
## 74   0.2867645 18
## 75   0.2612891 18
## 76   0.2380769 18
## 77   0.2169268 18
## 78   0.1976557 18
## 79   0.1800965 18
## 80   0.1640972 19
```

```r
dim(g3$beta)
```

```
## [1] 19 80
```

```r
df2 <- apply(g3$beta, 2, function(t) sum(t!=0))
```

```r
data.frame(df1=g3$df, df2=df2)
```

```
##     df1 df2
## s0    0   0
## s1    1   1
## s2    2   2
## s3    2   2
## s4    3   3
## s5    4   4
```

```
## s6     4   4
## s7     4   4
## s8     4   4
## s9     4   4
## s10    5   5
## s11    5   5
## s12    5   5
## s13    5   5
## s14    6   6
## s15    6   6
## s16    6   6
## s17    6   6
## s18    6   6
## s19    6   6
## s20    6   6
## s21    6   6
## s22    6   6
## s23    6   6
## s24    6   6
## s25    6   6
## s26    6   6
## s27    6   6
## s28    6   6
## s29    7   7
## s30    7   7
## s31    7   7
## s32    9   9
## s33    9   9
## s34    9   9
## s35    9   9
## s36    9   9
## s37   11  11
## s38   11  11
## s39   12  12
## s40   12  12
## s41   13  13
## s42   13  13
## s43   13  13
## s44   13  13
## s45   13  13
## s46   13  13
## s47   13  13
## s48   13  13
## s49   13  13
## s50   13  13
## s51   14  14
## s52   15  15
## s53   15  15
## s54   17  17
## s55   17  17
## s56   17  17
## s57   17  17
## s58   17  17
## s59   17  17
```

```
## s60   17   17
## s61   17   17
## s62   17   17
## s63   17   17
## s64   18   18
## s65   18   18
## s66   18   18
## s67   17   17
## s68   18   18
## s69   18   18
## s70   18   18
## s71   18   18
## s72   18   18
## s73   18   18
## s74   18   18
## s75   18   18
## s76   18   18
## s77   18   18
## s78   18   18
## s79   19   19
```

Selecting the Tuning Parameter in shrinkage methods

```r
set.seed(123)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.test <- y[test]
grid <- 10^seq(10, -2, length=100)
r1 <- glmnet(x[train, ], y[train], alpha=0, lambda=grid)
ss <- 0:(length(r1$lambda)-1)
Err <- NULL
for (i in 1:length(r1$lambda)) {
  r1.pred <- predict(r1, s=ss[i], newx=x[test, ])
  Err[i] <- mean((r1.pred - y.test)^2)
}
wh <- which.min(Err)
lam.opt <- r1$lambda[wh]
```

```r
r.full <- glmnet(x, y, alpha=0, lambda=grid)
r.full$beta[,wh]
```

One standard error rule

```
##       AtBat        Hits       HmRun        Runs         RBI       Walks       Years
## -290.21072  332.28861   34.40765  -56.04846  -23.66939  134.40389  -17.76977
##      CAtBat       CHits      CHmRun       CRuns        CRBI      CWalks     LeagueN
## -399.62021  136.72188    4.61810  451.57447  229.04284 -209.74151   31.65086
##   DivisionW     PutOuts     Assists      Errors NewLeagueN
##   -58.53064    78.79843    54.03884  -22.57788  -12.96145
```

```r
predict(r.full, type="coefficients", s=lam.opt)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  164.11322
## AtBat       -290.21072
## Hits         332.28861
## HmRun         34.40765
## Runs         -56.04846
## RBI          -23.66939
## Walks        134.40389
## Years        -17.76977
## CAtBat      -399.62021
## CHits        136.72188
## CHmRun         4.61810
## CRuns        451.57447
## CRBI         229.04284
## CWalks      -209.74151
## LeagueN       31.65086
## DivisionW    -58.53064
## PutOuts       78.79843
## Assists       54.03884
## Errors       -22.57788
## NewLeagueN   -12.96145
```

- 누가 validation set으로 할당되냐에 따라 결과가 크게 차이날 수 있어서 실제론 CV가 훨씬 안정적이고 좋음.

```r
set.seed(1)
cv.r <- cv.glmnet(x, y, alpha=0, nfolds=10)
names(cv.r)
```

Cross-Validation(One standard error rule)

```
##  [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"
##  [6] "nzero"      "call"       "name"       "glmnet.fit" "lambda.min"
## [11] "lambda.1se" "index"
```

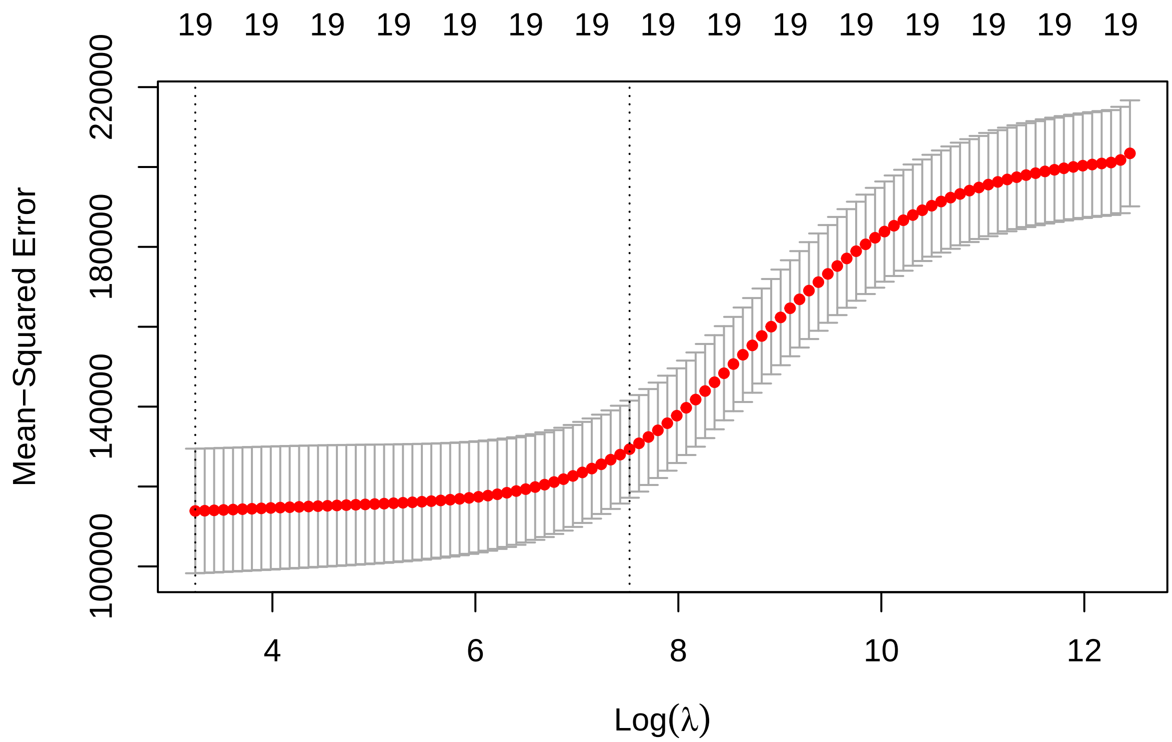
```r
cbind(cv.r$cvlo, cv.r$cvm, cv.r$cvup)
```

```
##              [,1]     [,2]     [,3]
##    [1,] 190140.46 203414.3 216688.2
##    [2,] 188435.90 201753.4 215070.9
##    [3,] 187999.83 201122.4 214244.9
##    [4,] 187767.54 200881.5 213995.6
##    [5,] 187513.77 200618.5 213723.2
##    [6,] 187236.65 200331.2 213425.7
##    [7,] 186934.18 200017.6 213101.1
##    [8,] 186604.21 199675.6 212747.0
##    [9,] 186244.44 199302.8 212361.1
```

```
## [10,] 185852.45 198896.5 211940.6
## [11,] 185425.62 198454.2 211482.8
## [12,] 184961.20 197973.0 210984.8
## [13,] 184456.31 197449.9 210443.5
## [14,] 183907.89 196881.8 209855.6
## [15,] 183312.74 196265.3 209217.9
## [16,] 182667.47 195597.1 208526.6
## [17,] 181968.87 194873.7 207778.5
## [18,] 181213.39 194091.5 206969.6
## [19,] 180397.47 193247.0 206096.5
## [20,] 179517.56 192336.4 205155.2
## [21,] 178570.15 191356.2 204142.2
## [22,] 177551.80 190302.9 203053.9
## [23,] 176459.22 189173.1 201887.0
## [24,] 175289.34 187963.8 200638.3
## [25,] 174039.38 186672.2 199305.0
## [26,] 172706.96 185296.0 197885.0
## [27,] 171290.15 183833.2 196376.3
## [28,] 169787.63 182282.8 194778.0
## [29,] 168198.74 180644.2 193089.7
## [30,] 166523.63 178917.8 191311.9
## [31,] 164763.34 177104.9 189446.4
## [32,] 162919.90 175207.9 187495.8
## [33,] 160996.41 173230.3 185464.1
## [34,] 158997.12 171176.8 183356.5
## [35,] 156927.45 169053.5 181179.5
## [36,] 154794.08 166867.6 178941.2
## [37,] 152605.00 164627.9 176650.8
## [38,] 150368.67 162343.6 174318.5
## [39,] 148095.28 160025.6 171956.0
## [40,] 145795.70 157685.8 169575.8
## [41,] 143481.49 155336.5 167191.5
## [42,] 141164.76 152990.7 164816.6
## [43,] 138857.85 150661.6 162465.4
## [44,] 136573.14 148362.4 160151.7
## [45,] 134322.76 146105.9 157889.1
## [46,] 132118.32 143904.4 155690.5
## [47,] 129970.65 141769.1 153567.6
## [48,] 127889.62 139710.3 151531.1
## [49,] 125883.88 137736.8 149589.8
## [50,] 123960.66 135855.8 147751.0
## [51,] 122125.72 134072.9 146020.1
## [52,] 120383.59 132392.3 144401.0
## [53,] 118737.40 130816.7 142896.1
## [54,] 117188.65 129347.0 141505.4
## [55,] 115737.57 127982.7 140227.7
## [56,] 114383.22 126721.9 139060.5
## [57,] 113123.58 125561.8 138000.0
## [58,] 111955.75 124498.6 137041.4
## [59,] 110876.09 123527.7 136179.3
## [60,] 109880.40 122644.0 135407.7
## [61,] 108964.11 121842.2 134720.2
## [62,] 108122.40 121116.4 134110.5
## [63,] 107350.29 120461.2 133572.1
```

```
##  [64,] 106642.06 119870.2 133098.3
##  [65,] 105996.08 119341.1 132686.2
##  [66,] 105406.42 118867.7 132328.9
##  [67,] 104866.59 118442.3 132018.0
##  [68,] 104373.47 118061.9 131750.3
##  [69,] 103922.08 117721.3 131520.5
##  [70,] 103511.49 117419.3 131327.1
##  [71,] 103139.05 117152.6 131166.1
##  [72,] 102796.32 116912.4 131028.5
##  [73,] 102482.09 116697.6 130913.1
##  [74,] 102197.31 116509.3 130821.2
##  [75,] 101934.73 116339.9 130745.0
##  [76,] 101693.90 116187.9 130681.8
##  [77,] 101472.07 116051.9 130631.6
##  [78,] 101265.93 115927.2 130588.5
##  [79,] 101074.12 115813.4 130552.8
##  [80,] 100895.41 115708.4 130521.5
##  [81,] 100727.89 115610.7 130493.5
##  [82,] 100567.47 115516.2 130464.9
##  [83,] 100417.38 115427.5 130437.6
##  [84,] 100272.99 115340.7 130408.5
##  [85,] 100132.01 115253.6 130375.1
##  [86,]  99995.72 115167.1 130338.4
##  [87,]  99864.31 115081.5 130298.8
##  [88,]  99734.43 114994.0 130253.6
##  [89,]  99605.68 114904.3 130202.9
##  [90,]  99478.24 114812.8 130147.3
##  [91,]  99352.11 114719.7 130087.3
##  [92,]  99227.07 114625.1 130023.0
##  [93,]  99100.68 114527.2 129953.6
##  [94,]  98976.99 114429.7 129882.4
##  [95,]  98851.93 114329.8 129807.7
##  [96,]  98728.10 114230.0 129731.9
##  [97,]  98605.97 114131.2 129656.4
##  [98,]  98484.38 114032.8 129581.3
##  [99,]  98364.96 113936.9 129508.8
## [100,]  98285.04 113880.2 129475.3
```

```r
plot(cv.r)
```




```r
log(cv.r$lambda.min)
```

```
## [1] 3.239784
```

```r
log(cv.r$lambda.1se)
```

```
## [1] 7.519336
```

```r
which.min(cv.r$lambda)
```

```
## [1] 100
```

```r
which(cv.r$lambda==cv.r$lambda.1se)
```

```
## [1] 54
```

```r
b.min <- predict(cv.r, type="coefficients", s=cv.r$lambda.min)
b.1se <- predict(cv.r, type="coefficients", s=cv.r$lambda.1se)
cbind(b.min, b.1se)
```

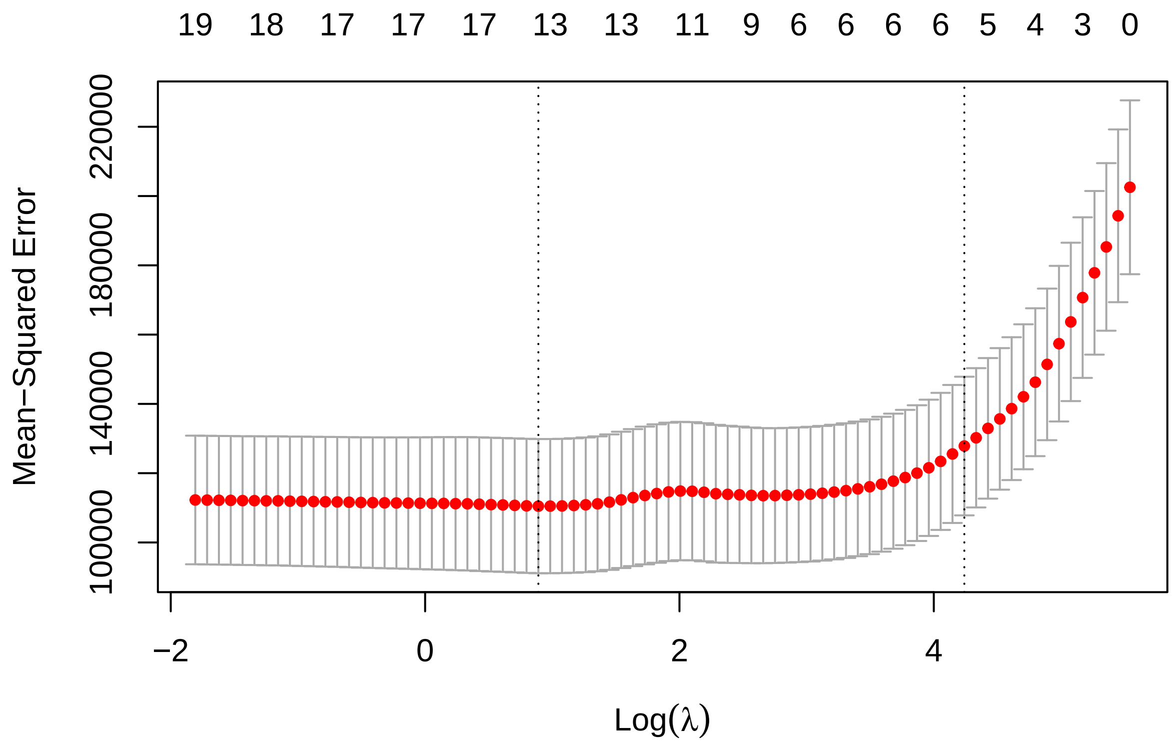
```
## 20 x 2 sparse Matrix of class "dgCMatrix"
##                      s1          s1
```

```
## (Intercept)   81.126932 159.796625
## AtBat       -100.212924  15.067887
## Hits         124.863402  20.125453
## HmRun        -11.936650  11.266961
## Runs          25.869160  17.918165
## RBI           18.419837  17.744128
## Walks         73.236073  20.071835
## Years        -43.380051  12.988099
## CAtBat        -2.737486  19.961033
## CHits         88.053957  22.229483
## CHmRun        57.264389  20.805218
## CRuns         97.811727  22.767565
## CRBI          82.970281  23.023410
## CWalks       -73.522590  17.424817
## LeagueN       26.563051   2.693852
## DivisionW    -61.406126 -14.545703
## PutOuts       73.730681  18.945085
## Assists       24.599326   1.332491
## Errors       -24.303154  -1.556113
## NewLeagueN    -9.028894   2.222950
```

```r
c(sqrt(sum(b.min[-1]^2)), sqrt(sum(b.1se[-1]^2)))
```

```
## [1] 279.4935  72.3250
```

```r
set.seed(2)
cv.l <- cv.glmnet(x, y, alpha=1, nfolds=10)
plot(cv.l)
```




```r
log(cv.l$lambda.min)
```

```
## [1] 0.8906821
```

```r
log(cv.l$lambda.1se)
```

```
## [1] 4.239897
```

```r
which(cv.l$lambda==cv.l$lambda.min)
```

```
## [1] 51
```

```r
which(cv.l$lambda==cv.l$lambda.1se)
```

```
## [1] 15
```

```r
b.min <- predict(cv.l, type="coefficients", s=cv.l$lambda.min)
b.1se <- predict(cv.l, type="coefficients", s=cv.l$lambda.1se)
cbind(b.min, b.1se)
```

```
## 20 x 2 sparse Matrix of class "dgCMatrix"
##                       s1          s1
```

```
## (Intercept)  129.41556 127.956948
## AtBat       -237.15665   .
## Hits         261.49419  64.110315
## HmRun          .          .
## Runs           .          .
## RBI            .          .
## Walks        105.06567  34.295642
## Years        -47.71291   .
## CAtBat         .          .
## CHits          .          .
## CHmRun        44.09343   .
## CRuns        225.18108  52.983406
## CRBI         125.98840 108.663331
## CWalks      -146.53946   .
## LeagueN       16.20588   .
## DivisionW    -59.66326  -4.030128
## PutOuts       76.60907  23.451888
## Assists       26.87539   .
## Errors       -14.27657   .
## NewLeagueN     .          .
```

```
c(sum(abs(b.min[-1])), sum(abs(b.1se[-1])))
```

```
## [1] 1386.8620  287.5347
```

Ridge: The Bias-Variance tradeoff

```
set.seed(1234)
K <- 100
p <- 40
n <- 50
beta <- runif(p, -1, 1)
lam <- 10^seq(3, -3, length.out=50)
x <- matrix(rnorm(n * p), n, p)
bhat <- array(0, c(p, length(lam), K))
```

```
for (i in 1:K) {
  y <- x %*% beta + rnorm(n)
  fit <- glmnet(x, y, alpha=0, lambda=lam)
  bhat[,,i] <- as.matrix(fit$beta)
}

MSE0 <- Bias0 <- Vars0 <- matrix(0, p, length(lam))

for (k in 1:length(lam)) {
  MS <- (bhat[,k,] - matrix(beta, p, K))^2
  MSE0[,k] <- apply(MS, 1, mean)
  Bias0[,k] <- abs(apply(bhat[,k,], 1, mean) - beta)
  Vars0[,k] <- apply(bhat[,k,], 1, var)
}
```

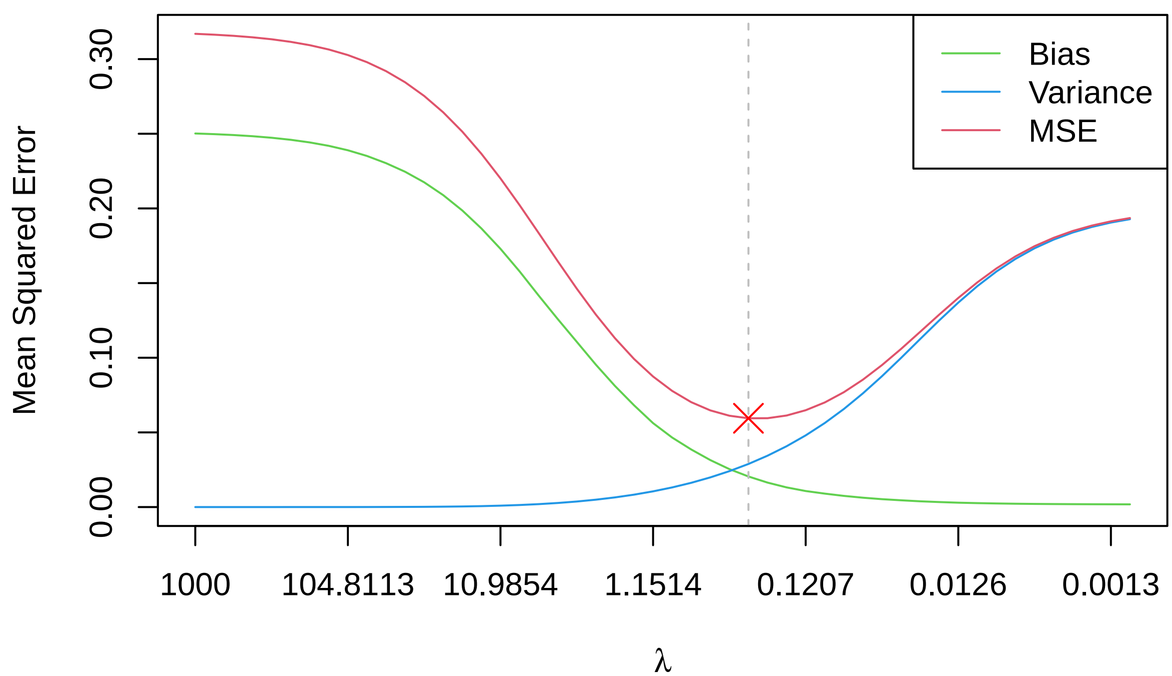
```r
MSE <- apply(MSE0, 2, mean)
Bias <- apply(Bias0, 2, mean)
Vars <- apply(Vars0, 2, mean)
```

```r
MAT <- cbind(Bias^2, Vars, MSE)
data.frame(lambda=round(lam, 3), MSE=MAT[,3])
```

```
##      lambda        MSE
## 1  1000.000 0.31693101
## 2   754.312 0.31635426
## 3   568.987 0.31559416
## 4   429.193 0.31459428
## 5   323.746 0.31328221
## 6   244.205 0.31156599
## 7   184.207 0.30933045
## 8   138.950 0.30643403
## 9   104.811 0.30270702
## 10   79.060 0.29795273
## 11   59.636 0.29195354
## 12   44.984 0.28448407
## 13   33.932 0.27533318
## 14   25.595 0.26433534
## 15   19.307 0.25140859
## 16   14.563 0.23659337
## 17   10.985 0.22008324
## 18    8.286 0.20223746
## 19    6.251 0.18356741
## 20    4.715 0.16469051
## 21    3.556 0.14626235
## 22    2.683 0.12889911
## 23    2.024 0.11309967
## 24    1.526 0.09920968
## 25    1.151 0.08742132
## 26    0.869 0.07778465
## 27    0.655 0.07025636
## 28    0.494 0.06475055
## 29    0.373 0.06117498
## 30    0.281 0.05945335
## 31    0.212 0.05950986
## 32    0.160 0.06131919
## 33    0.121 0.06484610
## 34    0.091 0.07005856
## 35    0.069 0.07691802
## 36    0.052 0.08531573
## 37    0.039 0.09508516
## 38    0.029 0.10590440
## 39    0.022 0.11733858
## 40    0.017 0.12888429
## 41    0.013 0.14005615
## 42    0.010 0.15047956
## 43    0.007 0.15980298
## 44    0.005 0.16789986
## 45    0.004 0.17472040
```

```
## 46     0.003 0.18032160
## 47     0.002 0.18489332
## 48     0.002 0.18848420
## 49     0.001 0.19134091
## 50     0.001 0.19354913
```

```
matplot(MAT, type="l", col=c(3,4,2), lty=1, xaxt="n",
xlab=expression(lambda), ylab="Mean Squared Error")
legend("topright", c("Bias", "Variance", "MSE"), col=c(3,4,2),
lty=1)

w <- which.min(MAT[,3])
abline(v=w, col="gray", lty=2)
points(w, MAT[w, 3], pch=4, col="red", cex=2)
cc <- seq(1, 50, 8)
axis(1, at=cc, labels=round(lam[cc],4))
```



Lasso: The Bias-Variance Trade Off

```
set.seed(1234)
K <- 100
p <- 40
n <- 50
```

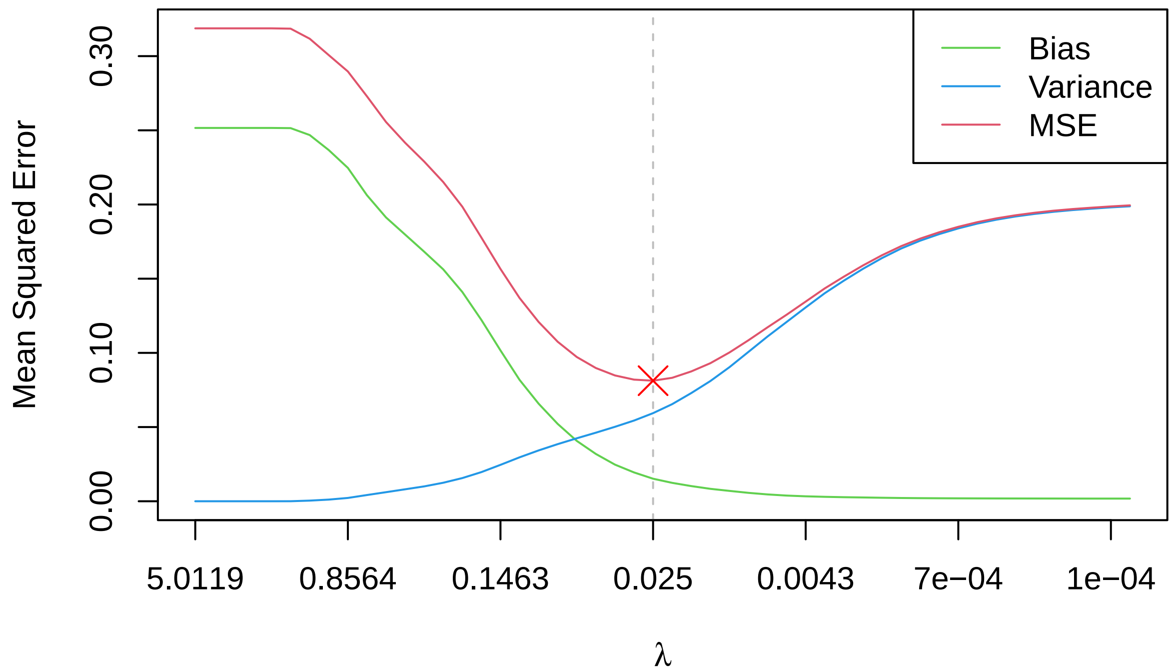
```r
beta <- runif(p, -1, 1)
lam <- 10^seq(0.7, -4, length.out=50)
x <- matrix(rnorm(n * p), n, p)
bhat <- array(0, c(p, length(lam), K))
for (i in 1:K) {
  y <- x %*% beta + rnorm(n)
  fit <- glmnet(x, y, alpha=1, lambda=lam)
  bhat[,,i] <- as.matrix(fit$beta)
}
MSE0 <- Bias0 <- Vars0 <- matrix(0, p, length(lam))
for (k in 1:length(lam)) {
  MS <- (bhat[,k,] - matrix(beta, p, K))^2
  MSE0[,k] <- apply(MS, 1, mean)
  Bias0[,k] <- abs(apply(bhat[,k,], 1, mean) - beta)
  Vars0[,k] <- apply(bhat[,k,], 1, var)
}
```

```r
MSE <- apply(MSE0, 2, mean)
Bias <- apply(Bias0, 2, mean)
Vars <- apply(Vars0, 2, mean)
MAT <- cbind(Bias^2, Vars, MSE)
data.frame(lambda=round(lam, 3), MSE=MAT[,3])
```

```
##    lambda        MSE
## 1   5.012 0.31872022
## 2   4.019 0.31872022
## 3   3.222 0.31872022
## 4   2.584 0.31872022
## 5   2.072 0.31872022
## 6   1.661 0.31850322
## 7   1.332 0.31171991
## 8   1.068 0.30066269
## 9   0.856 0.28965552
## 10  0.687 0.27292256
## 11  0.551 0.25560874
## 12  0.441 0.24162140
## 13  0.354 0.22897777
## 14  0.284 0.21512478
## 15  0.228 0.19851600
## 16  0.182 0.17771447
## 17  0.146 0.15656136
## 18  0.117 0.13699184
## 19  0.094 0.12084441
## 20  0.075 0.10749690
## 21  0.060 0.09720984
## 22  0.048 0.08979449
## 23  0.039 0.08479431
## 24  0.031 0.08197614
## 25  0.025 0.08125365
## 26  0.020 0.08319555
## 27  0.016 0.08748459
## 28  0.013 0.09301937
## 29  0.010 0.10024209
```

```
## 30  0.008 0.10850368
## 31  0.007 0.11722004
## 32  0.005 0.12570653
## 33  0.004 0.13455070
## 34  0.003 0.14345889
## 35  0.003 0.15135923
## 36  0.002 0.15890115
## 37  0.002 0.16579777
## 38  0.001 0.17190427
## 39  0.001 0.17699509
## 40  0.001 0.18128398
## 41  0.001 0.18497898
## 42  0.001 0.18810408
## 43  0.000 0.19067300
## 44  0.000 0.19273950
## 45  0.000 0.19442198
## 46  0.000 0.19582081
## 47  0.000 0.19694150
## 48  0.000 0.19786052
## 49  0.000 0.19871040
## 50  0.000 0.19941652
```

```r
matplot(MAT, type="l", col=c(3,4,2), lty=1, xaxt="n",
xlab=expression(lambda), ylab="Mean Squared Error")
legend("topright", c("Bias", "Variance", "MSE"), col=c(3,4,2),
lty=1)
w <- which.min(MAT[,3])
abline(v=w, col="gray", lty=2)
points(w, MAT[w, 3], pch=4, col="red", cex=2)
cc <- seq(1, 50, 8)
axis(1, at=cc, labels=round(lam[cc],4))
```



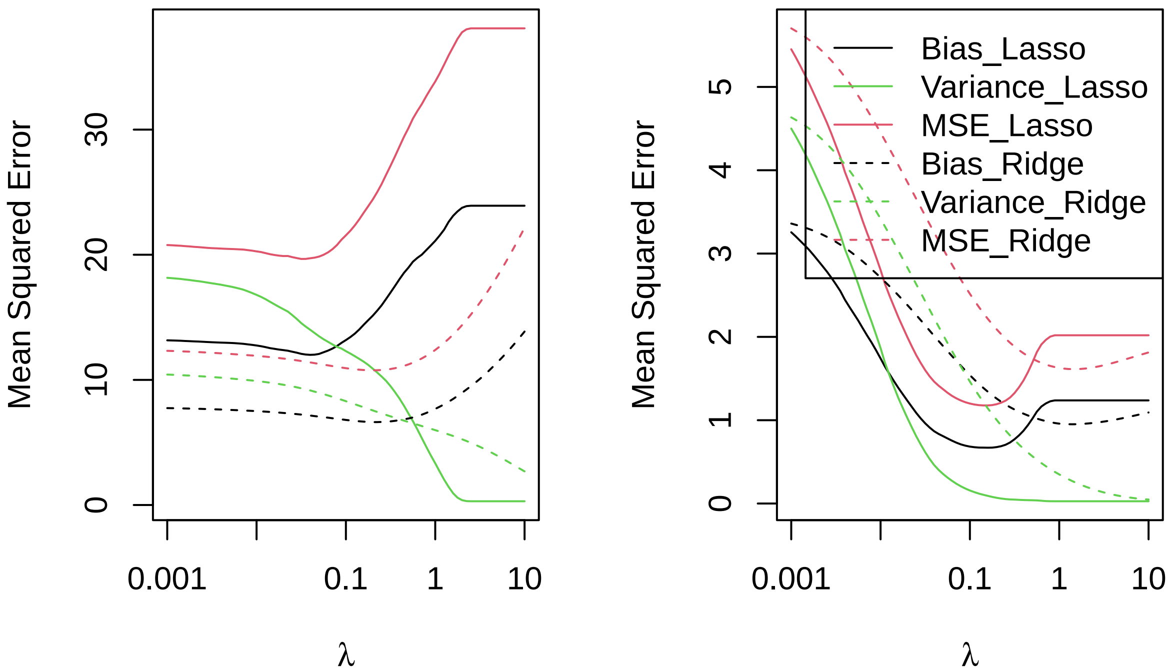

```
MSE.fun <- function(n, p, K, beta, lam, xtest, ytest) {
  yhat0 <- yhat1 <- array(0, c(n, length(lam), K))
  for (i in 1:K) {
    x <- matrix(rnorm(n * p), n, p)
    y <- x %*% beta + rnorm(n)
    g0 <- glmnet(x, y, alpha=0, lambda=lam)
    g1 <- glmnet(x, y, alpha=1, lambda=lam)
    yhat0[1:n, 1:length(lam), i] <- predict(g0, x.test)
    yhat1[1:n, 1:length(lam), i] <- predict(g1, x.test)
  }
  MSE0 <- Bias0 <- Vars0 <- array(0, c(n,length(lam)))
  MSE1 <- Bias1 <- Vars1 <- array(0, c(n,length(lam)))
  for (j in 1:length(lam)) {
    PE0 <-(yhat0[,j,] - matrix(ytest, n, K))^2
    PE1 <-(yhat1[,j,] - matrix(ytest, n, K))^2
    MSE0[ ,j] <- apply(PE0, 1, mean)
    MSE1[ ,j] <- apply(PE1, 1, mean)
    BS0 <- abs(yhat0[,j,] - matrix(ytest, n, K))
    BS1 <- abs(yhat1[,j,] - matrix(ytest, n, K))
    Bias0[,j] <- apply(BS0, 1, mean)
    Bias1[,j] <- apply(BS1, 1, mean)
    Vars0[,j] <- apply(yhat0[,j,], 1, var)
    Vars1[,j] <- apply(yhat1[,j,], 1, var)
  }
  MSE.r <- apply(MSE0, 2, mean)
  MSE.l <- apply(MSE1, 2, mean)
```

```r
  Bia.r <- apply(Bias0, 2, mean)
  Bia.l <- apply(Bias1, 2, mean)
  Var.r <- apply(Vars0, 2, mean)
  Var.l <- apply(Vars1, 2, mean)
  ridge <- apply(cbind(Bia.r^2, Var.r, MSE.r), 2, rev)
  lasso <- apply(cbind(Bia.l^2, Var.l, MSE.l), 2, rev)
  newlam <- rev(lam)
  return(list(ridge=ridge,lasso=lasso, lambda=newlam))
}
```

```r
set.seed(111000)
K <- 10
p <- 120
n <- 100
lam <- 10^seq(1, -3, -0.05)
x.test <- matrix(rnorm(n * p), n, p)
```

```r
## The case that all predictors have non-zero coefficients
beta1 <- beta2 <- runif(p, -1, 1)
ytest1 <- x.test %*% beta1 + rnorm(n)
g1 <- MSE.fun(n, p, K, beta1, lam, xtest, ytest1)
RES1 <- cbind(g1$lasso, g1$ridge)

## The case that only 5 predictors have non-zero coefficients
beta2[6:p] <- 0
ytest2 <- x.test %*% beta2 + rnorm(n)
g2 <- MSE.fun(n, p, K, beta2, lam, xtest, ytest2)
RES2 <- cbind(g2$lasso, g2$ridge)
```

```r
par(mfrow=c(1,2))
matplot(RES1, type="l", col=c(1,3,2), lty=rep(1:2,each=3),
xlab=expression(lambda), xaxt="n",
ylab="Mean Squared Error")
cc <- c(1, seq(21, 81, 20))
axis(1, at=cc, labels=g1$lambda[cc])
matplot(RES2, type="l", col=c(1,3,2), lty=rep(1:2,each=3),
xlab=expression(lambda), xaxt="n",
ylab="Mean Squared Error")
legend("topright",c("Bias_Lasso", "Variance_Lasso", "MSE_Lasso",
"Bias_Ridge", "Variance_Ridge", "MSE_Ridge"),
col=c(1,3,2), lty=rep(1:2, each=3))
axis(1, at=cc, labels=g2$lambda[cc])
```


Mean Squared Error
0
10
20
30
0.001
0.1
1
10
λ
Mean Squared Error
0
1
2
3
4
5
0.001
0.1
1
10
λ
Bias_Lasso
Variance_Lasso
MSE_Lasso
Bias_Ridge
Variance_Ridge
MSE_Ridge