TMA4268 Statistical Learning

Module 6: Linear Model Selection and Regularization

Thiago G. Martins, Department of Mathematical Sciences, NTNU

Spring 2021

Lab 1: Subset Selection Methods

Best Subset Selection

Goal: We wish to predict a baseball player's Salary on the basis of various statistics associated with performance in the previous year.

```
library(ISLR) # Package with data for an Introduction to Statistical
              # Learning with Applications in R
```

The dataset contains the response variable (or target variable) `Salary` and 19 covariates (or features), totaling 20 columns. The original dataset has 322 rows.

```
data(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"
```

```
dim(Hitters)
```

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

Some rows contain missing values for the `Salary` variable.

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

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

We use `na.omit` to remove all the rows with missing values in any variable.

```
Hitters=na.omit(Hitters)
dim(Hitters)
```

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

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

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

We now have zero missing values in our `Hitters` dataset.

```
Hitters=na.omit(Hitters)
dim(Hitters)
```

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

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

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

The `regsubsets()` function (part of the `leaps` library) performs best subset selection by identifying the best model that contains a given number of predictors, where best is quantified using RSS. The syntax is the same as for `lm()`. The `summary()` command outputs the best set of variables for each model size.

```
library(leaps)
regfit.full=regsubsets(Salary~.,Hitters)
summary(regfit.full)
```

```
## 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
## 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 ) " "  "*"    " "     "*"       "*"      " "      " "     " "
```

By default, `regsubsets()` only reports results up to the best eight-variable model. But the `nvmax` option can be used in order to return as many variables as are desired.

```
regfit.full=regsubsets(Salary~.,data=Hitters,nvmax=19)
reg.summary=summary(regfit.full)
reg.summary
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19)
## 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 19
## 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 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"
## 19  ( 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 )  " "  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  "*"  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"  "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
## 13  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

We can check what else we have available in the `summary` object.

```
names(reg.summary)
```

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

```r
reg.summary$rsq
```

```
##  [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227
##  [8] 0.5285569 0.5346124 0.5404950 0.5426153 0.5436302 0.5444570 0.5452164
## [15] 0.5454692 0.5457656 0.5459518 0.5460945 0.5461159
```

```r
par(mfrow=c(2,2))
plot(reg.summary$rss,xlab="Number of Variables",ylab="RSS",type="l")
plot(reg.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
which.max(reg.summary$adjr2)
```

```
## [1] 11
```

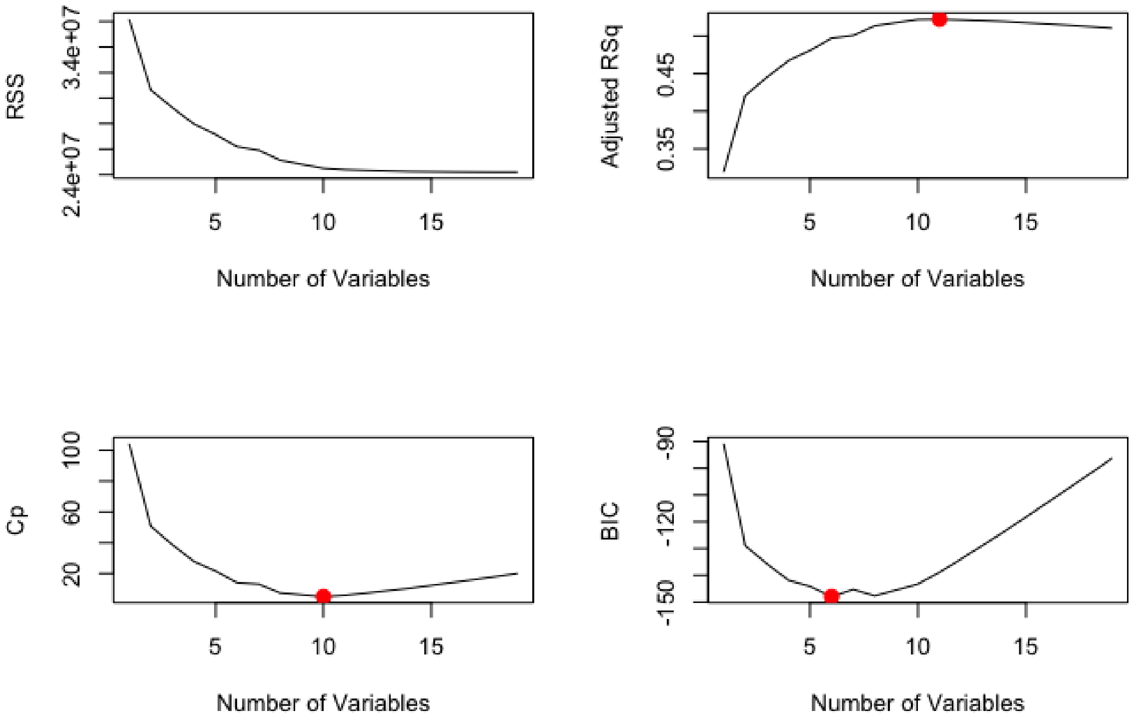
```r
points(11,reg.summary$adjr2[11], col="red",cex=2,pch=20)
plot(reg.summary$cp,xlab="Number of Variables",ylab="Cp",type='l')
which.min(reg.summary$cp)
```

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

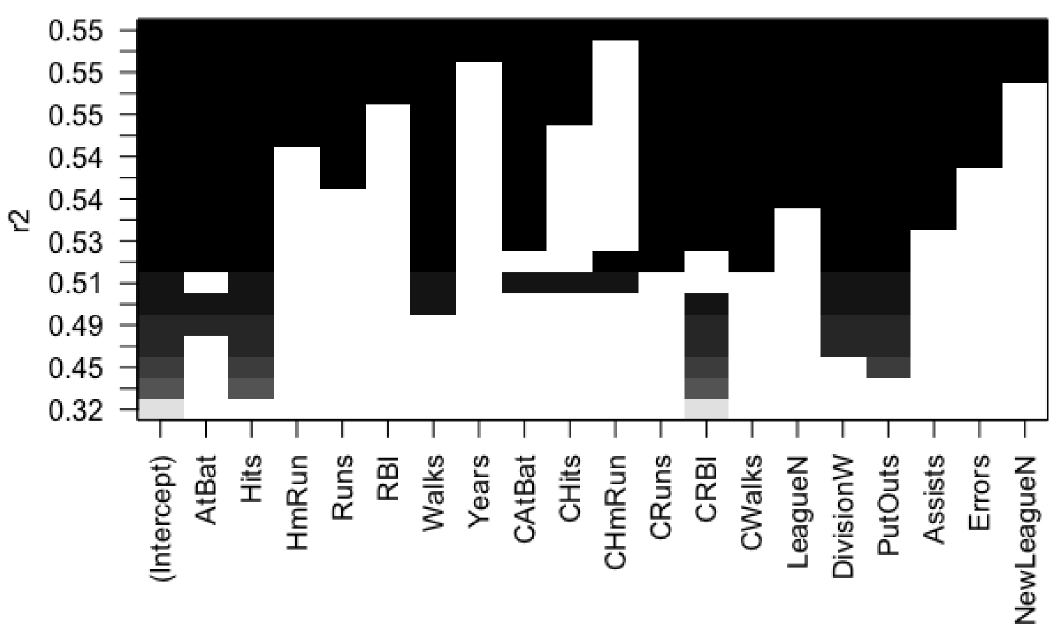
```r
points(10,reg.summary$cp[10],col="red",cex=2,pch=20)
which.min(reg.summary$bic)
```

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

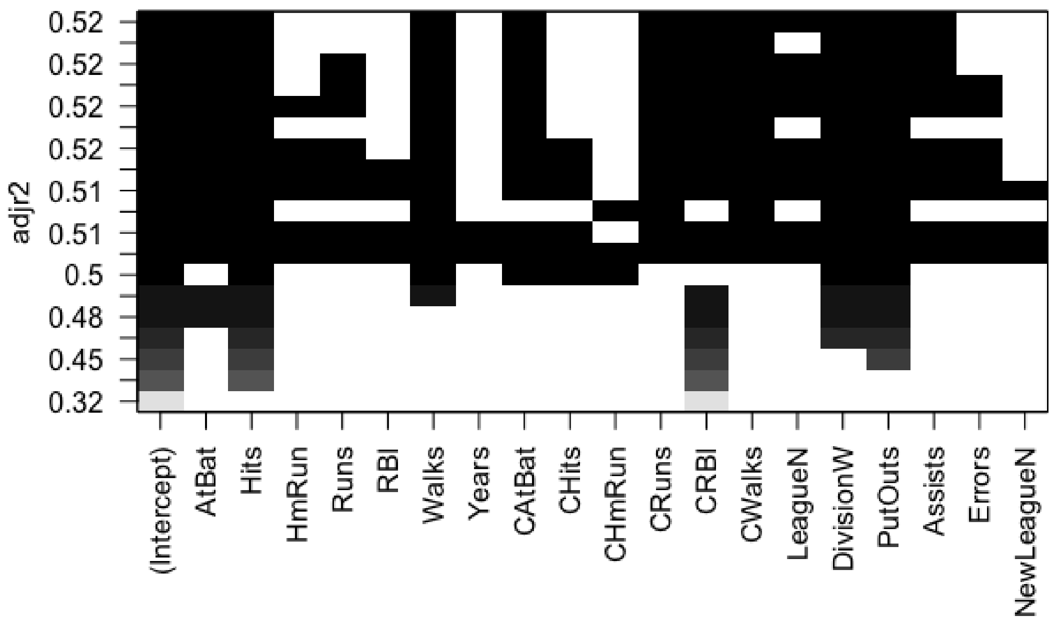
```r
plot(reg.summary$bic,xlab="Number of Variables",ylab="BIC",type='l')
points(6,reg.summary$bic[6],col="red",cex=2,pch=20)
```



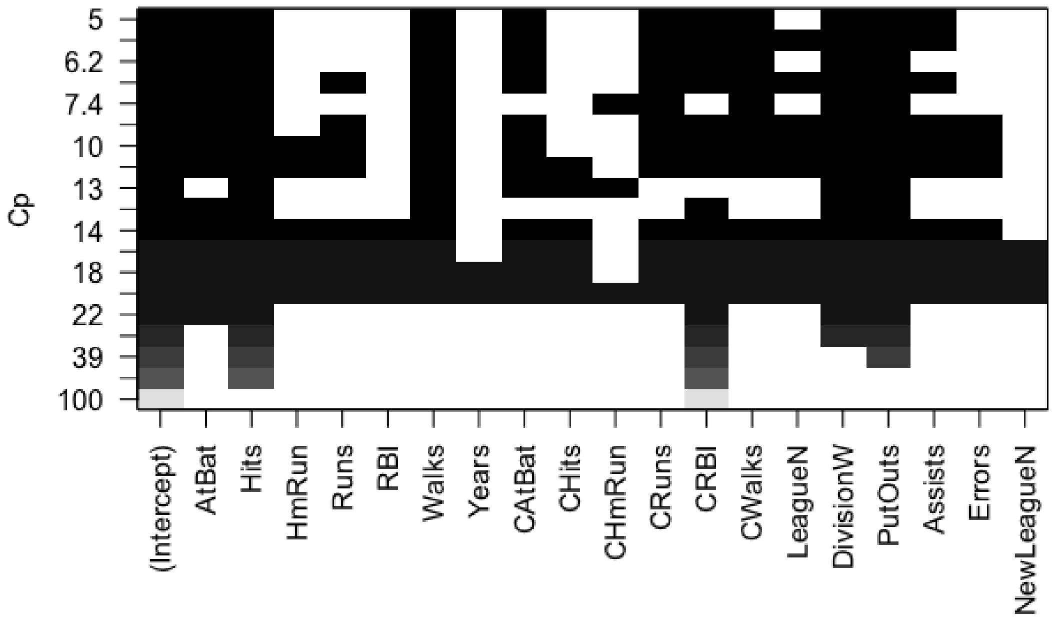

```r
plot(regfit.full,scale="r2")
```



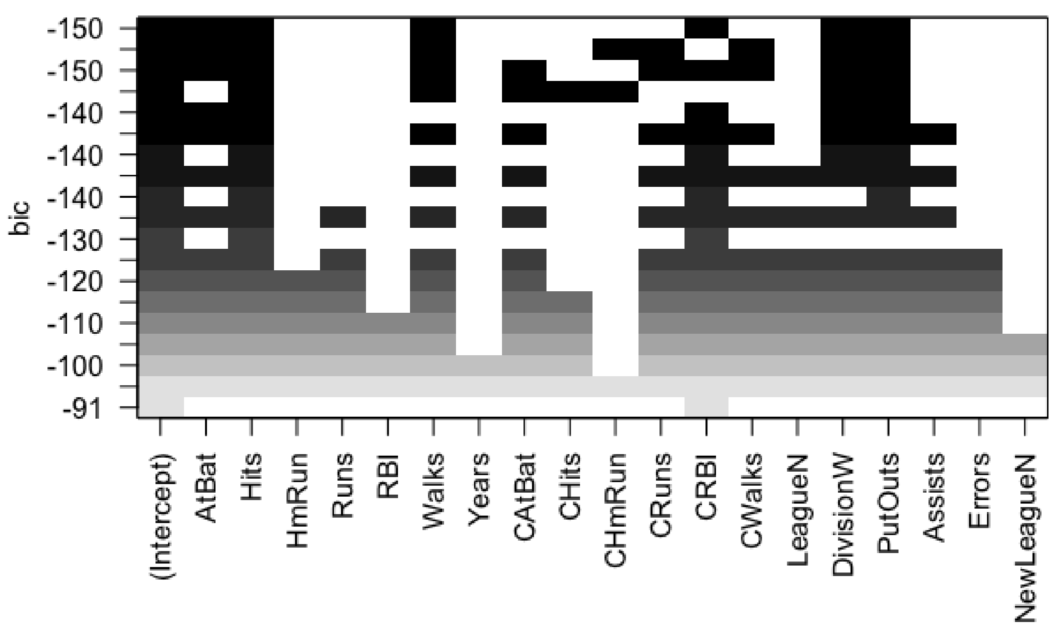
```r
plot(regfit.full,scale="adjr2")
```




```r
plot(regfit.full,scale="Cp")
```




```r
plot(regfit.full,scale="bic")
```



```r
coef(regfit.full,6)
```

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

Forward and Backward Stepwise Selection

We can also use the `regsubsets()` function to perform forward stepwise or backward stepwise selection.

Use the argument method="forward" to perform forward selection:

```r
regfit.fwd=regsubsets(Salary~.,data=Hitters,nvmax=19,method="forward")
summary(regfit.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "forward"
## 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 19
```

```
## Selection Algorithm: forward
##           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 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"
## 19  ( 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 )  "*"  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  "*"  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"  "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
## 13  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

Use the argument method="backward" to perform backward selection:

```
regfit.bwd=regsubsets(Salary~.,data=Hitters,nvmax=19,method="backward")
summary(regfit.bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "backward
## 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 19
## Selection Algorithm: backward
##           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 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"
## 19  ( 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 ) "*"  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 ) "*"  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"  "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
## 13  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

For this data, the best one-variable through six-variable models are each identical for best subset and forward selection. However, the best seven-variable models identified by forward stepwise selection, backward stepwise selection, and best subset selection are different.

```
coef(regfit.full,7)
```

```
##  (Intercept)         Hits        Walks       CAtBat        CHits
##   79.4509472    1.2833513    3.2274264   -0.3752350    1.4957073
##       CHmRun    DivisionW      PutOuts
##    1.4420538 -129.9866432    0.2366813
```

```
coef(regfit.fwd,7)
```

```
##  (Intercept)        AtBat         Hits        Walks         CRBI
##  109.7873062   -1.9588851    7.4498772    4.9131401    0.8537622
##       CWalks    DivisionW      PutOuts
##   -0.3053070 -127.1223928    0.2533404
```

```
coef(regfit.bwd,7)
```

```
##  (Intercept)        AtBat         Hits        Walks        CRuns
##  105.6487488   -1.9762838    6.7574914    6.0558691    1.1293095
##       CWalks    DivisionW      PutOuts
##   -0.7163346 -116.1692169    0.3028847
```

Validation Set Approach

In order for these approaches to yield accurate estimates of the test error, we must use only the training observations to perform all aspects of model-fitting, including variable selection.

In order to use the validation set approach, we begin by splitting the observations into a training set and a test set.

```
set.seed(1)
train=sample(c(TRUE,FALSE), nrow(Hitters),rep=TRUE)
test=(!train)
```

Now, we apply `regsubsets()` to the training set in order to perform best subset selection.

```
regfit.best=regsubsets(Salary~.,data=Hitters[train,],nvmax=19)
```

We now compute the validation set error for the best model of each model size. We first make a model matrix from the test data.

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

Now we run a loop, and for each size `i`, we extract the coefficients from `regfit.best` for the best model of that size, multiply them into the appropriate columns of the test model matrix to form the predictions, and compute the test MSE.

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

```
}
val.errors
```

```
##  [1] 164377.3 144405.5 152175.7 145198.4 137902.1 139175.7 126849.0
##  [8] 136191.4 132889.6 135434.9 136963.3 140694.9 140690.9 141951.2
## [15] 141508.2 142164.4 141767.4 142339.6 142238.2
```

We find that the best model is the one that contains ten variables.

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

```
## [1] 7
```

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

```
##  (Intercept)        AtBat         Hits        HmRun        Walks
##   71.8074075   -1.5038124    5.9130470  -11.5241809    8.4349759
##       CAtBat        CRuns         CRBI       CWalks    DivisionW
##   -0.1654850    1.7064330    0.7903694   -0.9107515 -109.5616997
##      PutOuts
##    0.2426078
```

Since there is no `predict()` method for `regsubsets()`, we can capture our steps above and write our own predict method.

```
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
}
```

Finally, we perform best subset selection on the full data set, and select the best ten-variable model. It is important that we make use of the full data set in order to obtain more accurate coefficient estimates. Note that we perform best subset selection on the full data set and select the best tenvariable model, rather than simply using the variables that were obtained from the training set, because the best ten-variable model on the full data set may differ from the corresponding model on the training set.

```
regfit.best=regsubsets(Salary~.,data=Hitters,nvmax=19)
coef(regfit.best,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
```

In fact, we see that the best ten-variable model on the full data set has a different set of variables than the best ten-variable model on the training set.

Cross-Validation Approach

This approach is somewhat involved, as we must perform best subset selection within each of the k training sets.

First, we create a vector that allocates each observation to one of $k = 10$ folds, and we create a matrix in which we will store the results.

```
k=10
set.seed(1)
folds=sample(1:k,nrow(Hitters),replace=TRUE)
cv.errors=matrix(NA,k,19, dimnames=list(NULL, paste(1:19)))
```

Now we write a for loop that performs cross-validation. In the jth fold, the elements of folds that equal j are in the test set, and the remainder are in the training set. We make our predictions for each model size (using our new `predict()` method), compute the test errors on the appropriate subset, and store them in the appropriate slot in the matrix `cv.errors`.

```
for(j in 1:k){
  best.fit=regsubsets(Salary~.,data=Hitters[folds!=j,],nvmax=19)
  for(i in 1:19){
    pred=predict(best.fit,Hitters[folds==j,],id=i)
```

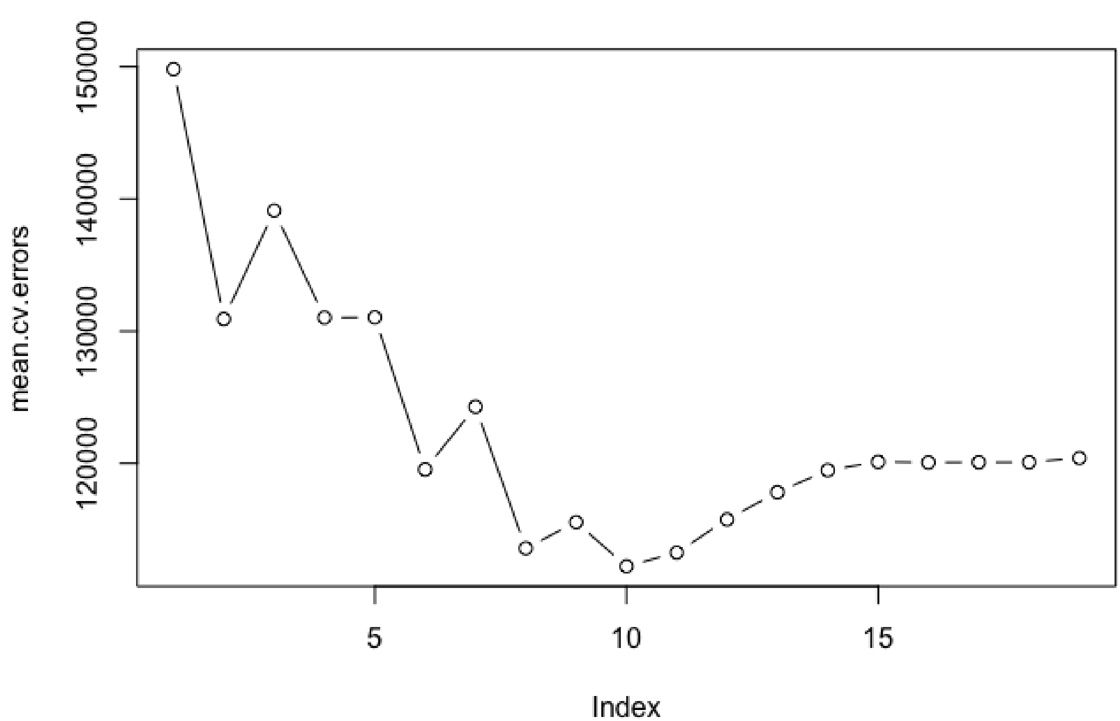
```
    cv.errors[j,i]=mean( (Hitters$Salary[folds==j]-pred)^2)
    }
}
```

We use the `apply()` function to average over the columns of this matrix in order to obtain a vector for which the jth element is the cross-validation error for the j-variable model.

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

```
##        1        2        3        4        5        6        7        8
## 149821.1 130922.0 139127.0 131028.8 131050.2 119538.6 124286.1 113580.0
##        9       10       11       12       13       14       15       16
## 115556.5 112216.7 113251.2 115755.9 117820.8 119481.2 120121.6 120074.3
##       17       18       19
## 120084.8 120085.8 120403.5
```

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



We see that cross-validation selects an 11-variable model. We now perform best subset selection on the full data set in order to obtain the 11-variable model.

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

Lab 2: Ridge Regression and the Lasso

```
library(glmnet) # Package Lasso and Elastic-Net Regularized
                # Generalized Linear Models
```

We will use the `glmnet` package in order to perform ridge regression and the lasso. The main function in this package is `glmnet()`, which can be used to fit ridge regression models, lasso models, and more.

This function has slightly different syntax from other model-fitting functions that we have encountered thus far in this book. In particular, we must pass in an `x` matrix as well as a `y` vector, and we do not use the `y ~ x syntax.

Lets then create the `y` vector and `x matrix.

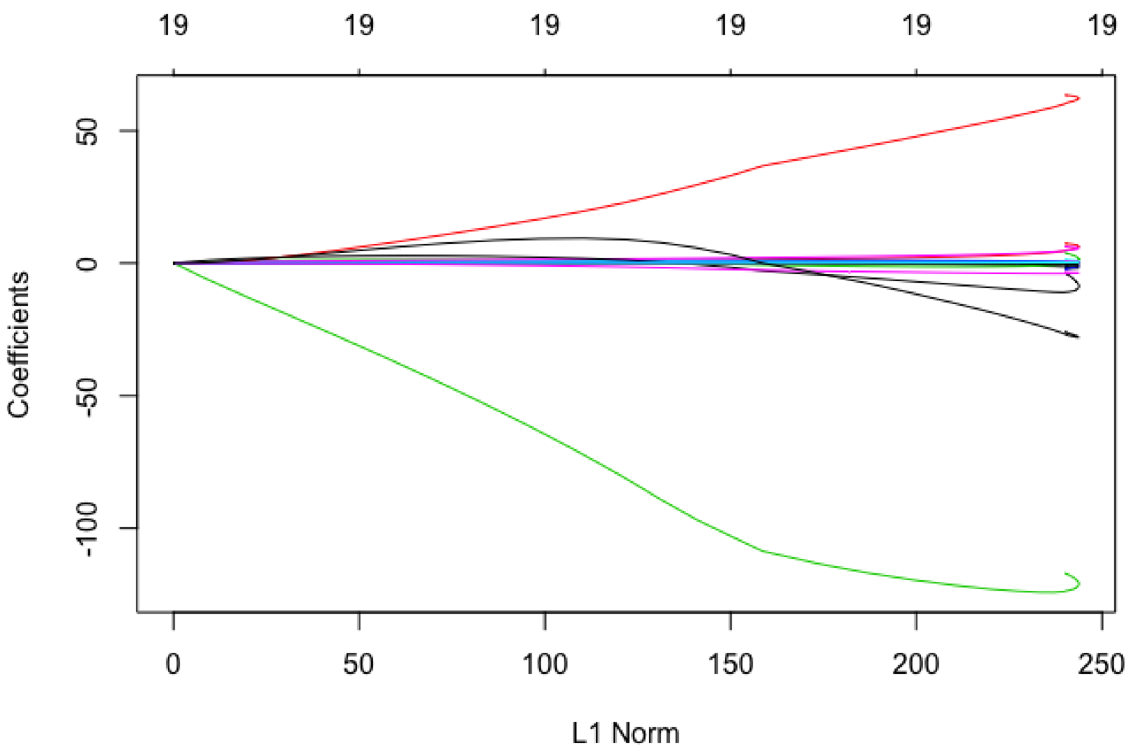
```
x=model.matrix(Salary~.,Hitters)[,-1]
y=Hitters$Salary
```

Ridge Regression

The `glmnet()` function has an `alpha` argument that determines what type of model is fit. If `alpha=0` then a ridge regression model is fit, and if `alpha=1` then a lasso model is fit.

We first fit a ridge regression model.

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



By default the `glmnet()` function performs ridge regression for an automatically selected range of λ values. However, here we have chosen to implement the function over a grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit.

Note that by default, the `glmnet()` function standardizes the variables so that they are on the same scale. To turn off this default setting, use the argument `standardize=FALSE`.

Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by `coef()`. In this case, it is a 20×100 matrix, with 20 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of λ).

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

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

These are the coefficients when $\lambda = 11,498$, along with their l_2 norm:

```
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
```

```
sqrt(sum(coef(ridge.mod)[-1,50]^2))
```

```
## [1] 6.360612
```

In contrast, here are the coefficients when $\lambda = 705$, along with their l_2 norm. Note the much larger l_2 norm of the coefficients associated with this smaller value of λ.

```
ridge.mod$lambda[60]
```

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

```
coef(ridge.mod)[,60]
```

```
##  (Intercept)       AtBat        Hits       HmRun        Runs 
##  54.32519950  0.11211115  0.65622409  1.17980910  0.93769713 
##          RBI       Walks       Years      CAtBat       CHits 
##   0.84718546  1.31987948  2.59640425  0.01083413  0.04674557 
##       CHmRun       CRuns        CRBI      CWalks     LeagueN 
##   0.33777318  0.09355528  0.09780402  0.07189612 13.68370191 
##    DivisionW     PutOuts     Assists      Errors  NewLeagueN 
## -54.65877750  0.11852289  0.01606037 -0.70358655  8.61181213
```

```
sqrt(sum(coef(ridge.mod)[-1,60]^2))
```

```
## [1] 57.11001
```

We can use the `predict()` function for a number of purposes. For instance, we can obtain the ridge regression coefficients for a new value of λ, say 50:

```
predict(ridge.mod,s=50,type="coefficients")[1:20,]
```

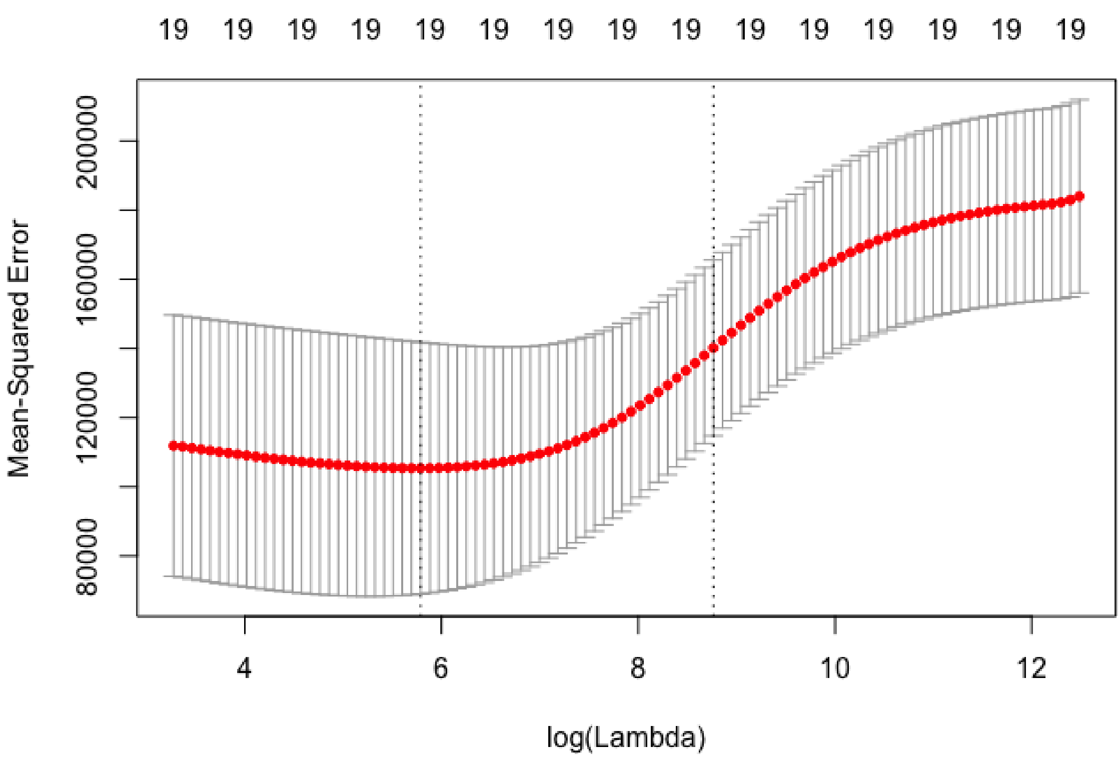
```
##   (Intercept)         AtBat          Hits         HmRun          Runs 
##  4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00  1.145892e+00 
##           RBI         Walks         Years        CAtBat         CHits 
##  8.038292e-01  2.716186e+00 -6.218319e+00  5.447837e-03  1.064895e-01 
##        CHmRun         CRuns          CRBI        CWalks       LeagueN 
##  6.244860e-01  2.214985e-01  2.186914e-01 -1.500245e-01  4.592589e+01 
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN 
## -1.182011e+02  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00
```

We now split the samples into a training set and a test set in order to estimate the test error of ridge regression and the lasso. This time we will randomly choose a subset of numbers between 1 and n and use it as the indices for the training observations.

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

In general, instead of arbitrarily choosing a specific λ, it would be better to use cross-validation to choose the tuning parameter λ. We can do this by using the built-in cross-validation function, `cv.glmnet()`. By default, the function performs ten-fold cross-validation, though this can be changed using the argument `nfolds`. Note that we set a random seed first so our results will be reproducible, since the choice of the cross-validation folds is random.

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




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

```
## [1] 326.0828
```

Therefore, we see that the value of λ that results in the smallest cross-validation error is 212. What is the test MSE associated with this value of λ?

```
ridge.pred=predict(ridge.mod,s=bestlam,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 119114.4
```

Finally, we refit our ridge regression model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates.

```
out=glmnet(x,y,alpha=0)
predict(out,type="coefficients",s=bestlam)[1:20,]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs 
##  15.44383135   0.07715547   0.85911581   0.60103107   1.06369007 
##          RBI        Walks        Years       CAtBat        CHits 
##   0.87936105   1.62444616   1.35254780   0.01134999   0.05746654 
##       CHmRun        CRuns         CRBI       CWalks      LeagueN 
##   0.40680157   0.11456224   0.12116504   0.05299202  22.09143189 
##    DivisionW      PutOuts      Assists       Errors   NewLeagueN 
## -79.04032637   0.16619903   0.02941950  -1.36092945   9.12487767
```

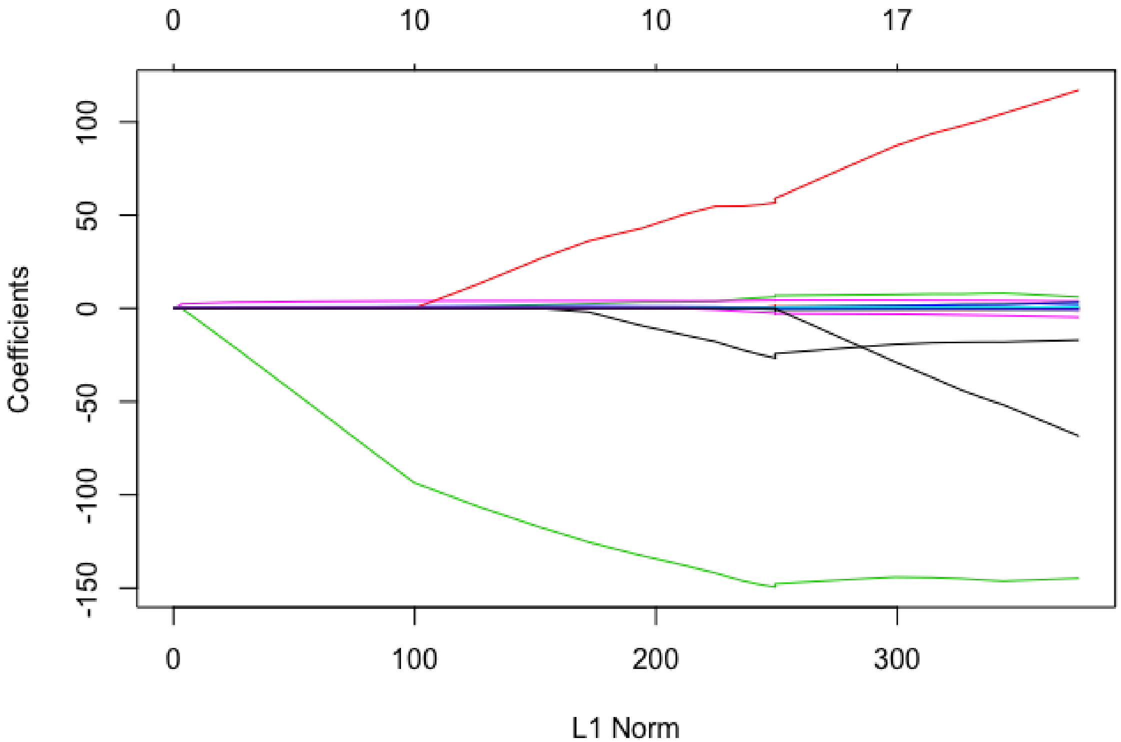
As expected, none of the coefficients are zero—ridge regression does not perform variable selection!

The Lasso

We saw that ridge regression with a wise choice of λ can outperform least squares as well as the null model on the Hitters data set. We now ask whether the lasso can yield either a more accurate or a more interpretable model than ridge regression.

In order to fit a lasso model, we once again use the `glmnet()` function; however, this time we use the argument `alpha=1`. Other than that change, we proceed just as we did in fitting a ridge model.

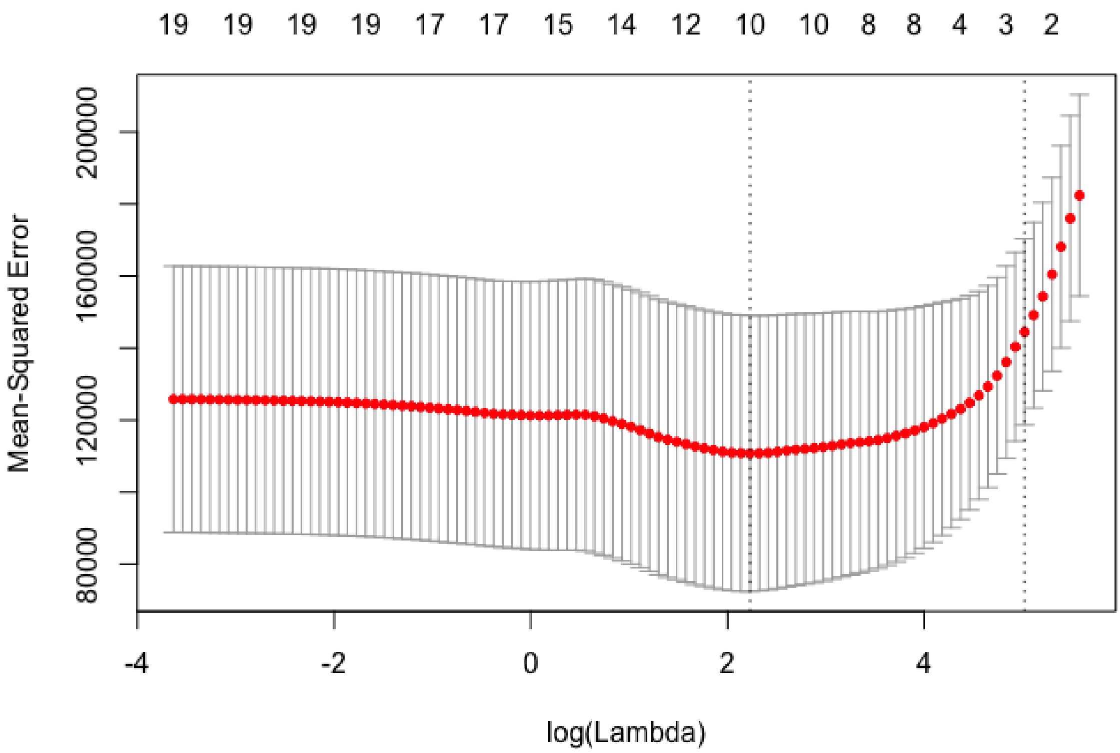
```
lasso.mod=glmnet(x[train,],y[train],alpha=1,lambda=grid)
plot(lasso.mod)
```



We can see from the coefficient plot that depending on the choice of tuning parameter, some of the coefficients will be exactly equal to zero.

We now perform cross-validation and compute the associated test error.

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




```
bestlam=cv.out$lambda.min
lasso.pred=predict(lasso.mod,s=bestlam,newx=x[test,])
mean((lasso.pred-y.test)^2)
```

```
## [1] 143673.6
```

This is substantially lower than the test set MSE of the null model and of least squares, and very similar to the test MSE of ridge regression with λ chosen by cross-validation.

However, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that 12 of the 19 coefficient estimates are exactly zero. So the lasso model with λ chosen by cross-validation contains only seven variables.

```
out=glmnet(x,y,alpha=1,lambda=grid)
lasso.coef=predict(out,type="coefficients",s=bestlam)[1:20,]
lasso.coef
```

```
##   (Intercept)        AtBat         Hits        HmRun         Runs
##    1.27479059  -0.05497143   2.18034583   0.00000000   0.00000000
```

```
##            RBI        Walks        Years       CAtBat        CHits 
##     0.00000000   2.29192406  -0.33806109   0.00000000   0.00000000 
##         CHmRun        CRuns         CRBI       CWalks      LeagueN 
##     0.02825013   0.21628385   0.41712537   0.00000000  20.28615023 
##      DivisionW      PutOuts      Assists       Errors   NewLeagueN 
## -116.16755870   0.23752385   0.00000000  -0.85629148   0.00000000
```

```
lasso.coef[lasso.coef!=0]
```

```
##   (Intercept)         AtBat          Hits         Walks         Years 
##    1.27479059   -0.05497143    2.18034583    2.29192406   -0.33806109 
##        CHmRun         CRuns          CRBI       LeagueN     DivisionW 
##    0.02825013    0.21628385    0.41712537   20.28615023 -116.16755870 
##       PutOuts        Errors 
##    0.23752385   -0.85629148
```

Lab 3: PCR and PLS Regression

Principal Components Regression

```
library(pls)
```

Principal components regression (PCR) can be performed using the `pcr()` function, which is part of the `pls` library.

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

- Setting `scale=TRUE` has the effect of standardizing each predictor.
- Setting `validation="CV"` causes `pcr()` to compute the ten-fold cross-validation error for each possible value of M

The resulting fit can be examined using `summary()`.

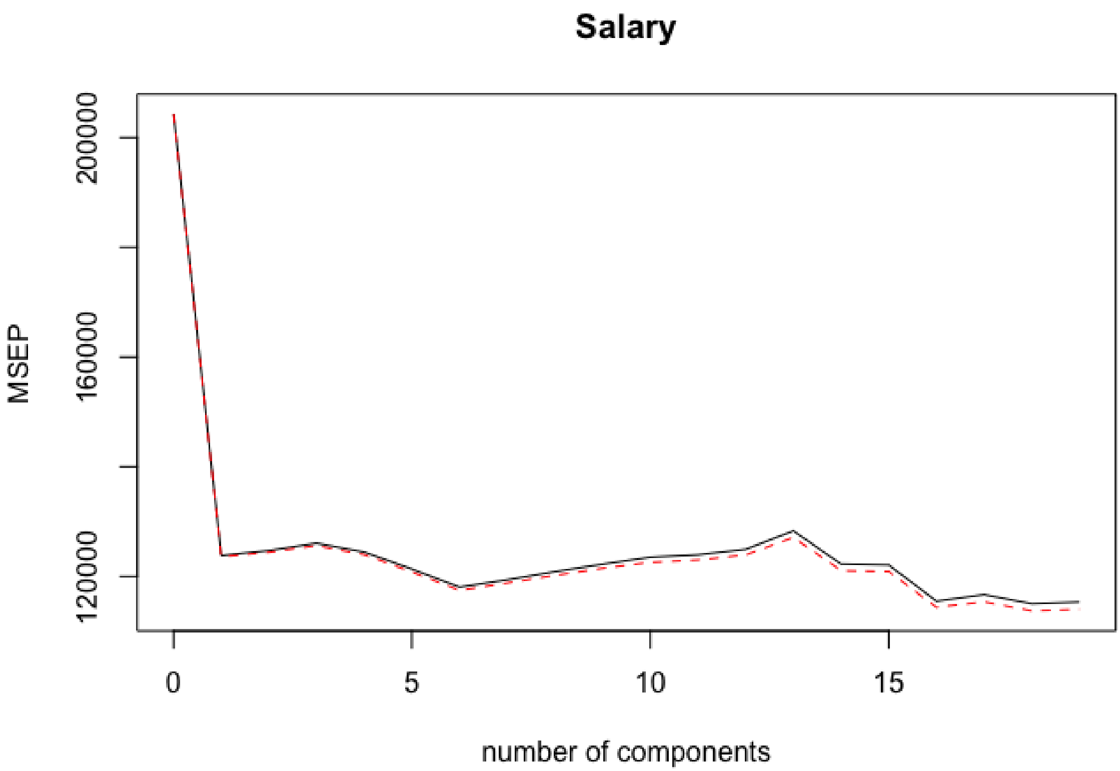
```
summary(pcr.fit)
```

```
## 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    351.9    353.2    355.0    352.8    348.4    343.6
## adjCV          452    351.6    352.7    354.4    352.1    347.6    342.7
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       345.5    347.7    349.6     351.4     352.1     353.5     358.2
## adjCV    344.7    346.7    348.5     350.1     350.7     352.0     356.5
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        349.7     349.4     339.9     341.6     339.2     339.6
## adjCV     348.0     347.7     338.2     339.7     337.2     337.6
##
## 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
```

Note that `pcr()` reports the root mean squared error; in order to obtain the usual MSE, we must square this quantity.

One can also plot the cross-validation scores using the `validationplot()` function. Using `val.type="MSEP"` will cause the cross-validation MSE to be plotted.

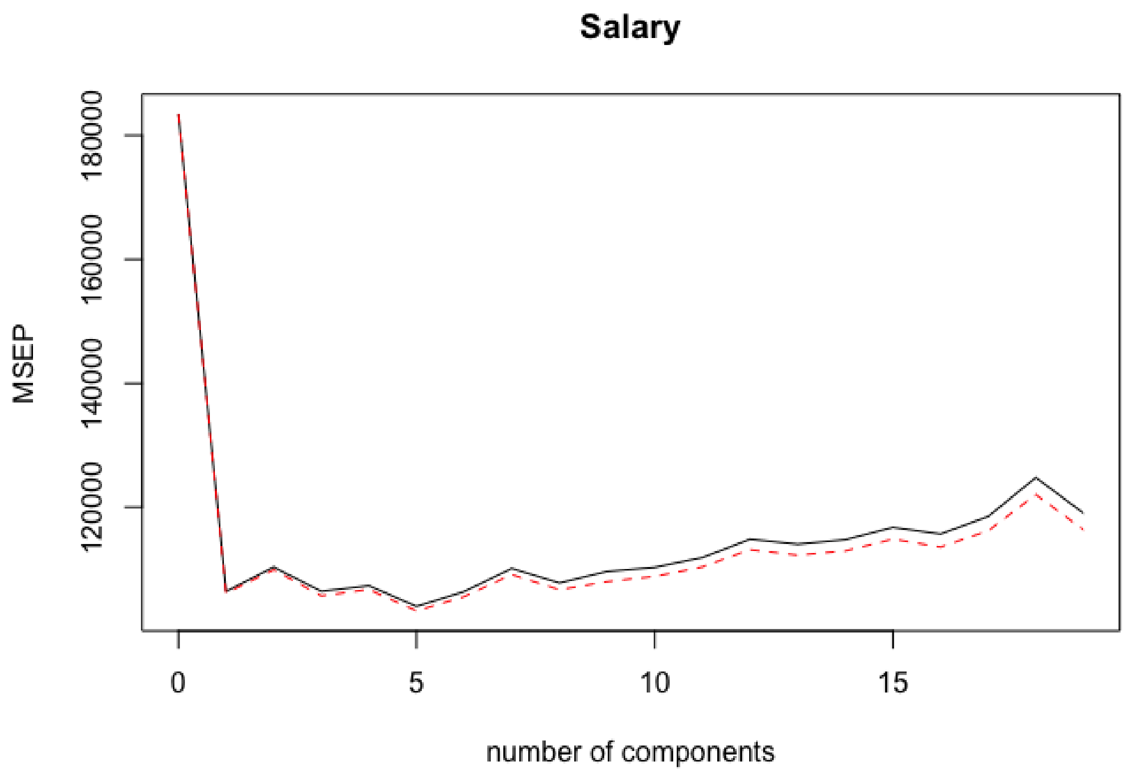
```
validationplot(pcr.fit,val.type="MSEP")
```



The `summary()` function also provides the percentage of variance explained in the predictors and in the response using different numbers of components.

We now perform PCR on the training data and evaluate its test set performance.

```
set.seed(1)
pcr.fit=pcr(Salary~., data=Hitters,subset=train,scale=TRUE, validation="CV")
validationplot(pcr.fit,val.type="MSEP")
```



Now we find that the lowest cross-validation error occurs when $M = 7$ component are used. We compute the test MSE as follows.

```
pcr.pred=predict(pcr.fit,x[test,],ncomp=7)
mean((pcr.pred-y.test)^2)
```

```
## [1] 140751.3
```

This test set MSE is competitive with the results obtained using ridge regression and the lasso. However, as a result of the way PCR is implemented, the final model is more difficult to interpret because it does not perform any kind of variable selection or even directly produce coefficient estimates.

Finally, we fit PCR on the full data set, using $M = 7$, the number of components identified by cross-validation.

```
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
```

Partial Least Squares

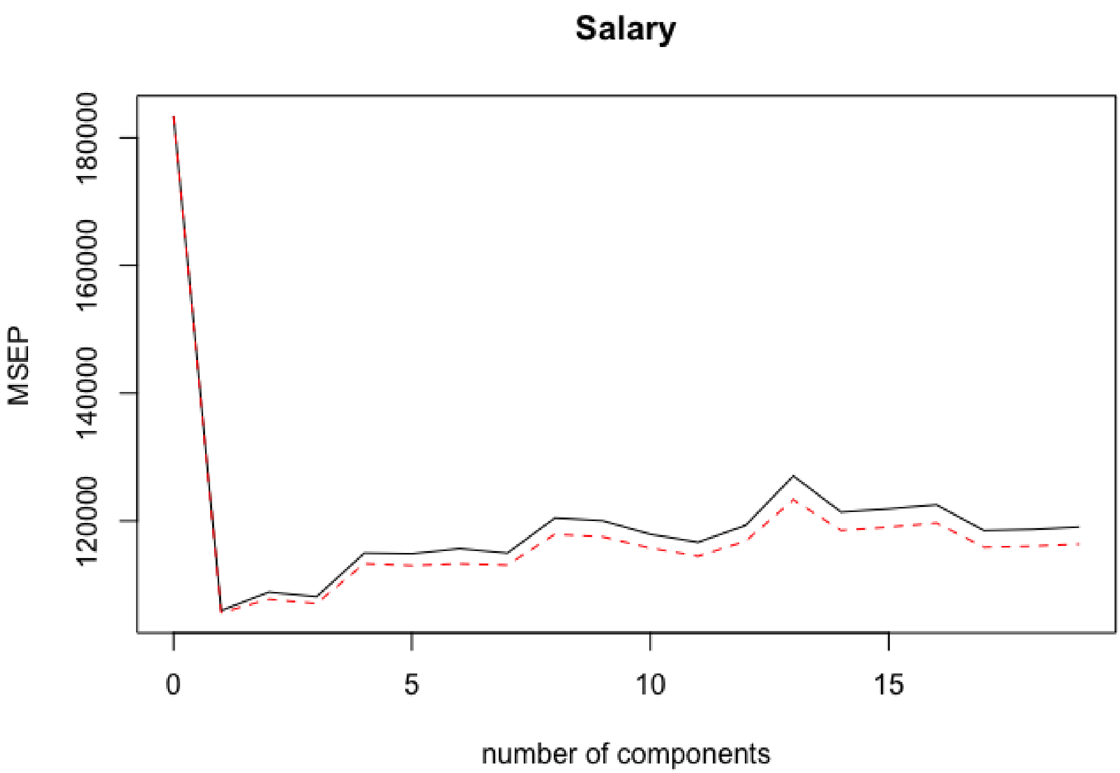
We implement partial least squares (PLS) using the `plsr()` function, also in the pls library.

```
set.seed(1)
pls.fit=plsr(Salary~., data=Hitters,subset=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           428.3    325.5    329.9    328.8    339.0    338.9    340.1
## adjCV        428.3    325.0    328.2    327.2    336.6    336.1    336.6
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       339.0    347.1    346.4     343.4     341.5     345.4     356.4
## adjCV    336.2    343.4    342.8     340.2     338.3     341.8     351.1
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        348.4     349.1     350.0     344.2     344.5     345.0
## adjCV     344.2     345.0     345.9     340.4     340.6     341.1
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         39.13    48.80    60.09    75.07    78.58    81.12    88.21
## Salary    46.36    50.72    52.23    53.03    54.07    54.77    55.05
##         8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X         90.71    93.17     96.05     97.08     97.61     97.97     98.70
## Salary    55.66    55.95     56.12     56.47     56.68     57.37     57.76
##         15 comps  16 comps  17 comps  18 comps  19 comps
```

```
## X           99.12     99.61     99.70     99.95     100.00
## Salary      58.08     58.17     58.49     58.56      58.62
```

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



The lowest cross-validation error occurs when only $M = 2$ partial least squares directions are used. We now evaluate the corresponding test set MSE.

```
pls.pred=predict(pls.fit,x[test,],ncomp=2)
mean((pls.pred-y.test)^2)
```

```
## [1] 145367.7
```

The test MSE is comparable to, but slightly higher than, the test MSE obtained using ridge regression, the lasso, and PCR.

Finally, we perform PLS using the full data set, using $M = 2$, the number of components identified by cross-validation.

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

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: kernelpls
## Number of components considered: 2
## TRAINING: % variance explained
##         1 comps  2 comps
## X         38.08    51.03
## Salary    43.05    46.40
```

Notice that the percentage of variance in Salary that the two-component PLS fit explains, 46.40%, is almost as much as that explained using the final seven-component model PCR fit, 46.69%. This is because PCR only attempts to maximize the amount of variance explained in the predictors, while PLS searches for directions that explain variance in both the predictors and the response.