Lab 6

Jonathan Griffith

2025-04-17

6.5.2 Ridge Regression and the Lasso

```
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.4.3
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.4.3
```

```
## Loading required package: Matrix
```

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

```
library(pls)
```

```
## Warning: package 'pls' was built under R version 4.4.3
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

We first load in our data and remove missing values. Then we put this data into an X matrix that has dummy variables for qualitative predictors and Y vector that has the Salary data.

```
Hitters <- na.omit(Hitters)
sum(is.na(Hitters))
```

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

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

Ridge Regression

We will first fit a ridge regression and provide a wide range of lambda values for different fits.

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

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

Let's look at the coefficient estimates for the model using $\lambda = 11498$. Notice that these estimates are very small relative to those with a smaller lambda.

```
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
```

```
#l2 norm for this lambda
paste('L2 Norm: ', sqrt(sum(coef(ridge.mod)[-1,50]^2)))
```

```
## [1] "L2 Norm:  6.36061242142793"
```

Now we look at the coefficients when $\lambda = 705$ along with their l2 norm. We notice that these coefficients are much larger relative to those above with a larger lambda.

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

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

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

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

```
paste('L2 Norm: ', sqrt(sum(coef(ridge.mod)[-1,60]^2)))
```

```
## [1] "L2 Norm:  57.1100142625331"
```

We can make predictions using this model and input new lambda values to see what coefficients are produced. Below, we'll show the predicted coefficients for $\lambda = 50$.

```
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
```

Now we will create a train test split to estimate the test error of both ridge and lasso.

```
set.seed(1)
```

```
# Sample the total number of rows without replacement to assign train and test values as a mask
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
y.test <- y[test]
```

Now we fit a model using the train set and use this model to make predictions on the test set using $\lambda = 4$. We then measure the MSE to see how it performs.

```
ridge.mod <- glmnet(x[train,], y[train], alpha=0,
                    lambda = grid, thresh = 1e-12)
ridge.pred <- predict(ridge.mod, s=4, newx=x[test,])
mean((ridge.pred - y.test)^2)
```

```
## [1] 142199.2
```

To calculate the MSE of a model that just fits the intercept, we can do the following:

```
mean((mean(y[train]) - y.test)^2)
```

```
## [1] 224669.9
```

Fitting a model with a very large lambda is essentially the same as just fitting the intercept, as shown below.

```
ridge.pred <- predict(ridge.mod, s=1e10, newx = x[test,])
mean((ridge.pred - y.test)^2)
```

```
## [1] 224669.8
```

We'll now compare our model with $\lambda = 4$ to standard OLS to see if ridge gives an improvement with this lambda value. Notice below that the coefficients are essentially the same for every predictor suggesting that $\lambda = 4$ does not improve much (if at all) on standard OLS.

```
ridge.pred <- predict(ridge.mod, s = 0, newx = x[test,],
                      exact = T, x = x[train,], y = y[train])
mean((ridge.pred - y.test)^2)
```

```
## [1] 168588.6
```

```
lm(y ~ x, subset = train)
```

```
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)       xAtBat        xHits       xHmRun        xRuns         xRBI
##    274.0145      -0.3521      -1.6377       5.8145       1.5424       1.1243
##      xWalks       xYears      xCAtBat       xCHits      xCHmRun       xCRuns
##      3.7287     -16.3773      -0.6412       3.1632       3.4008      -0.9739
##       xCRBI      xCWalks     xLeagueN   xDivisionW     xPutOuts     xAssists
##     -0.6005       0.3379     119.1486    -144.0831       0.1976       0.6804
##     xErrors  xNewLeagueN
##     -4.7128     -71.0951
```

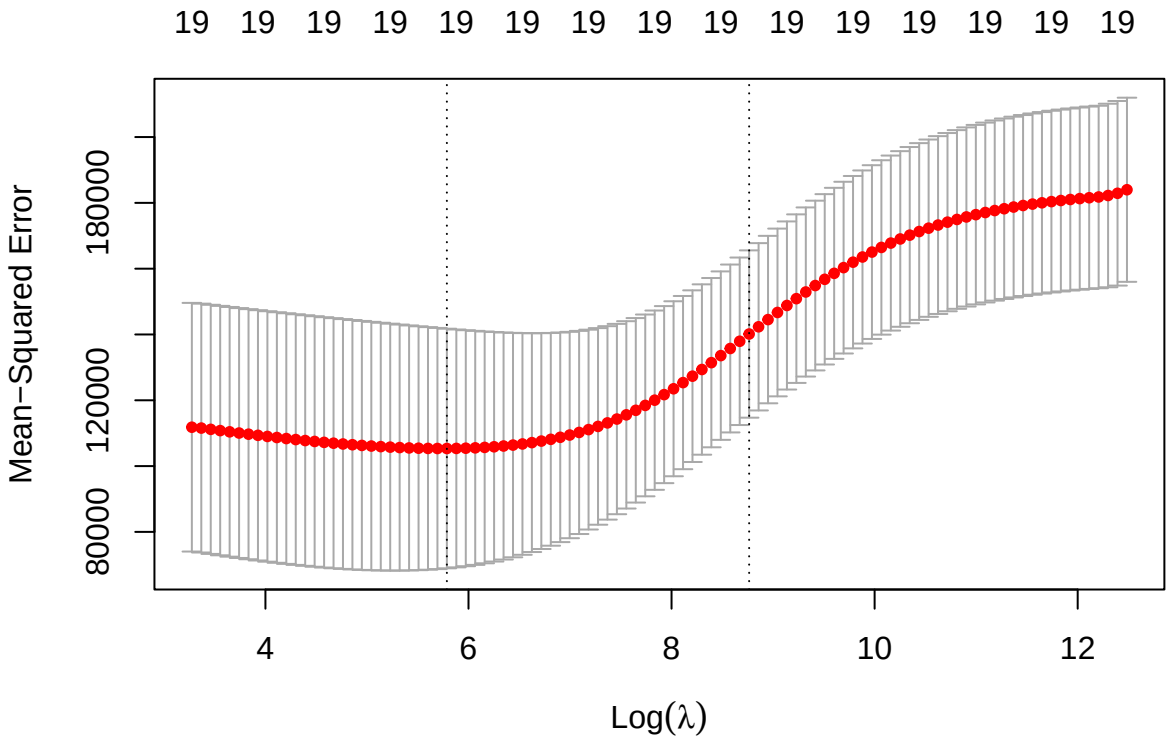
```
predict(ridge.mod, s = 0, exact = T, type = 'coefficients', x = x[train,], y = y[train])[1:20,]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  274.0200994   -0.3521900   -1.6371383    5.8146692    1.5423361    1.1241837
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##    3.7288406  -16.3795195   -0.6411235    3.1629444    3.4005281   -0.9739405
```

```
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   -0.6003976    0.3378422  119.1434637 -144.0853061    0.1976300    0.6804200
##       Errors   NewLeagueN
##   -4.7127879  -71.0898914
```

Now we'll use cross-validation to test a range of lambda values to see if there is an optimal lambda for improvement on OLS. THe default is 10-fold, which we will use.

```
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
```

We see that the smallest error is $\lambda = 326$ and now we will check to test the MSE associated with this lambda value.

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

```
## [1] 139856.6
```

We see an improvement over $\lambda = 4$, and we will fit the full dataset to see our coefficient values associated with this lambda value using the full dataset model. Notice that ridge regression includes all variables and doesn't set any equal to like lasso.

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

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  15.44383120   0.07715547   0.85911582   0.60103106   1.06369007   0.87936105
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
```

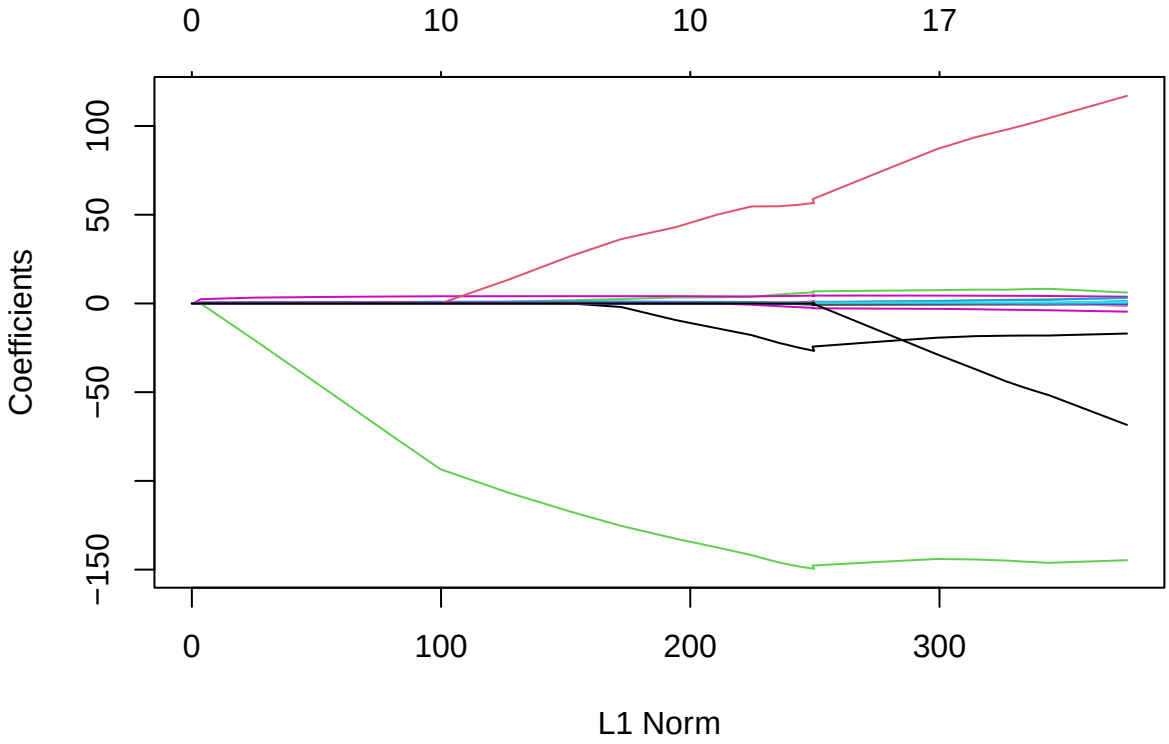
```
##   1.62444617   1.35254778   0.01134999   0.05746654   0.40680157   0.11456224
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.12116504   0.05299202 22.09143197 -79.04032656   0.16619903   0.02941950
##       Errors   NewLeagueN
##  -1.36092945   9.12487765
```

The Lasso

Now we will use the Lasso model and see if that outperforms the ridge and OLS models. Notice in the plot below that some coefficients are set to exactly zero based on the value of lambda.

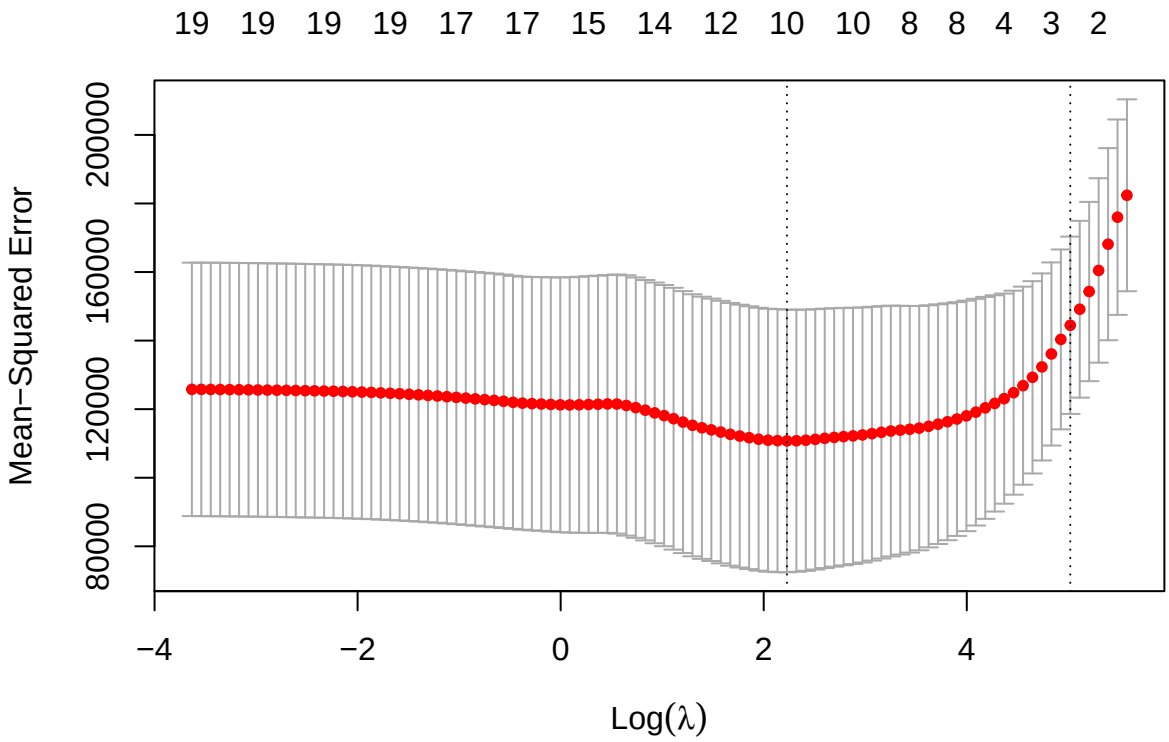
```
lasso.mod <- glmnet(x[train,], y[train], alpha=1, lambda=grid)
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



Next we perform cross-validation and compute the associated test error. Notice that the MSE associated with the optimal lambda from this test is very similar to the lowest MSE associated with optimal lambda in ridge regression. Lasso shows itself to outperform OLS and be similar to Lasso.

```
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
```

Now we show the number of nonzero coefficient estimates to show the advantage that Lasso has over ridge. We have a sparse model that is more interpretable. We have a resulting 11 nonzero coefficient estimates indicating the 11 variables Lasso produced as the most important.

```
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
```

6.5.3 PCR and PLS Regression

Now we move on to principal component regression and partial least squares

Principal Components Regression

We will first fit a PCR model and use the same 'Hitters' dataset as in the previous section to predict Salary. We will scale each predictor variable to remove any effect from the original scale native to each. We also compute the cross-validation for each possible value of M, the number of principal components used, which is

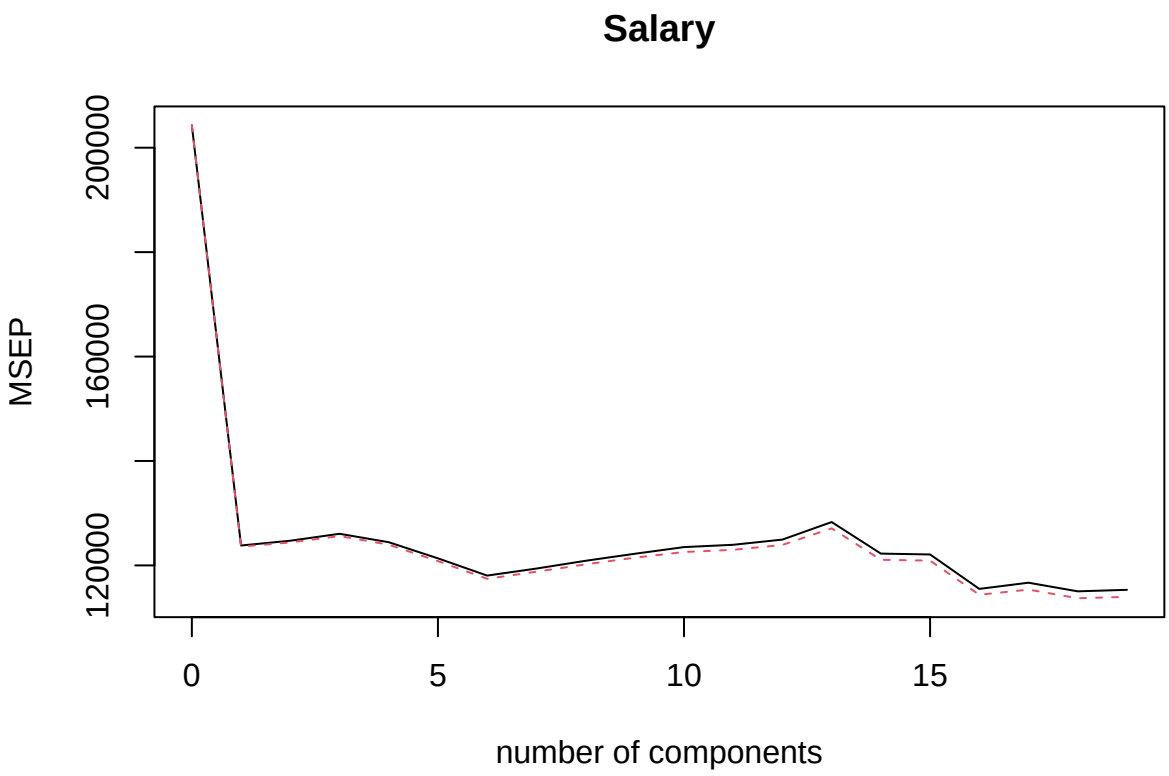
built in to the function and defaults as 10-fold. Below in the summary, we can see how much variance is captured from the predictors in the bottom TRAINING row.

```
set.seed(2)

pcr.fit <- pcr(Salary ~ ., data = Hitters, scale=TRUE, validation = 'CV')

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

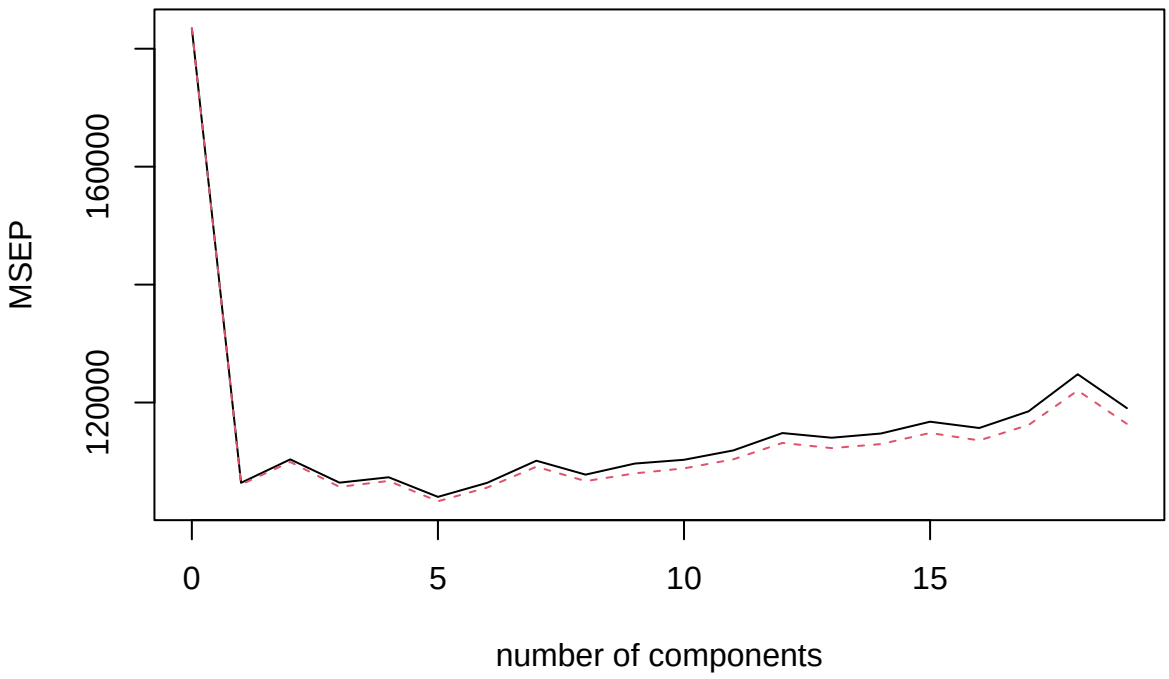
We also plot the cross-validation scores as MSEs. We see below that the MSE is lowest with few components and with many components, with higher values in between, suggesting that we should try a small number of components.

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



Now we will perform PCR on the training data and evaluate its test set performance. Notice in the plot of all of the MSEs associated with number of components below that the smallest MSE is associated with five components.

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



Now we will compute the test MSE using $M = 5$ components. Notice that it is similar to the results form ridge regression and lasso. The downside with PCR is the results aren't very interpretable. We don't know the linear combination of predictor variables used to create these five components.

```
pcr.pred <- predict(pcr.fit, x[test,], ncomp=5)
mean((pcr.pred - y.test)^2)
```

```
## [1] 142811.8
```

Lastly, we will fit the full data set using PCR and $M = 5$ components.

```
pcr.fit <- pcr(y ~ x, scale = TRUE, ncomp=5)
summary(pcr.fit)
```

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

Partial Least Squares

We will now implement partial least squares using the plsr() function in the same library as with PCR, 'pls'.

```
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  8 comps
## X         39.13    48.80    60.09    75.07    78.58    81.12    88.21    90.71
## Salary    46.36    50.72    52.23    53.03    54.07    54.77    55.05    55.66
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         93.17     96.05     97.08     97.61     97.97     98.70     99.12
## Salary    55.95     56.12     56.47     56.68     57.37     57.76     58.08
##         16 comps  17 comps  18 comps  19 comps
## X          99.61     99.70     99.95    100.00
## Salary     58.17     58.49     58.56     58.62
```

We see above that the lowest CV error occurs at $M = 1$ partial least squares direction. Now we will evaluate the test set MSE. Notice that it is slightly higher than ridge, lasso, and PCR.

```
pls.pred <- predict(pls.fit, x[test,], ncomp = 1)
mean((pls.pred - y.test)^2)
```

```
## [1] 151995.3
```

Finally, we perform PLS using the full data set and $M = 1$. We see a similar amount of variance explained using $M = 1$ as we did in PCR using $M = 5$, which makes sense since PLS is accounting for the predictors and the response in explaining variance, rather than just the predictors with PCR.

```
pls.fit <- plsr(Salary ~ ., data = Hitters, scale=TRUE, ncomp = 1)
summary(pls.fit)
```

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