STAT488_HW3_Ronquillo

Melchor Ronquillo

2022-10-05

Chapter 5, Question 3: We now review k-fold cross-validation.

```
a) Explain how k-fold cross validaation is implemented

    K-fold criss valdiation randomly divides set observations into k groups
    or folds of equal size. The first fold is treated as a validation set
    and is fit on the remaining k-1 folds. The mean squared error is computed
    on observations on the held out folds, and the whole process getd repeated
    k number of times where a different group of observations is treated
    as a validation set each time. There will be k amount of estimates, and
    the actual k- fold cross validation estimate is the average of the values.

b) What are the advantages and disadvantages of k-fold cross-validation
    relative to:

    i) The validation set approach?

        advantage of k-fold relative to set approach: variability of k-fold is
        typically much lower than variablity in the test error estimates that
        results from the validation set approach validation set may tend to
        overestimate the test error for model fit on entire dataset

      disadvantage: performing k-fold CV for k=5 or 10 leads to an intermediate
      level of bias which is substantially more than valdiation set approach

    ii) LOOCV? (Leave One Out Cross Validation)

        advantage of k-fold relative to LOOCV: often gives more accurate
        estimates of test error rate (bias-variance trade off),
        LOOCV may pose computational problems while k-fold is more feasible,
        based on procedure's variance LOOCV also has higher variance than does
        k-fold with k<n

        disadvantage: estimates of prediction error from k-fold is typically
        biased upward, therefore form the perspective of bias reduction LOOCV
        is to be preferred to k-fold CV
```

Chapter 5, Question 8: We will now perform cross-validation on a simulated data set.

```
a) Generate a simulated data set as follows:
```

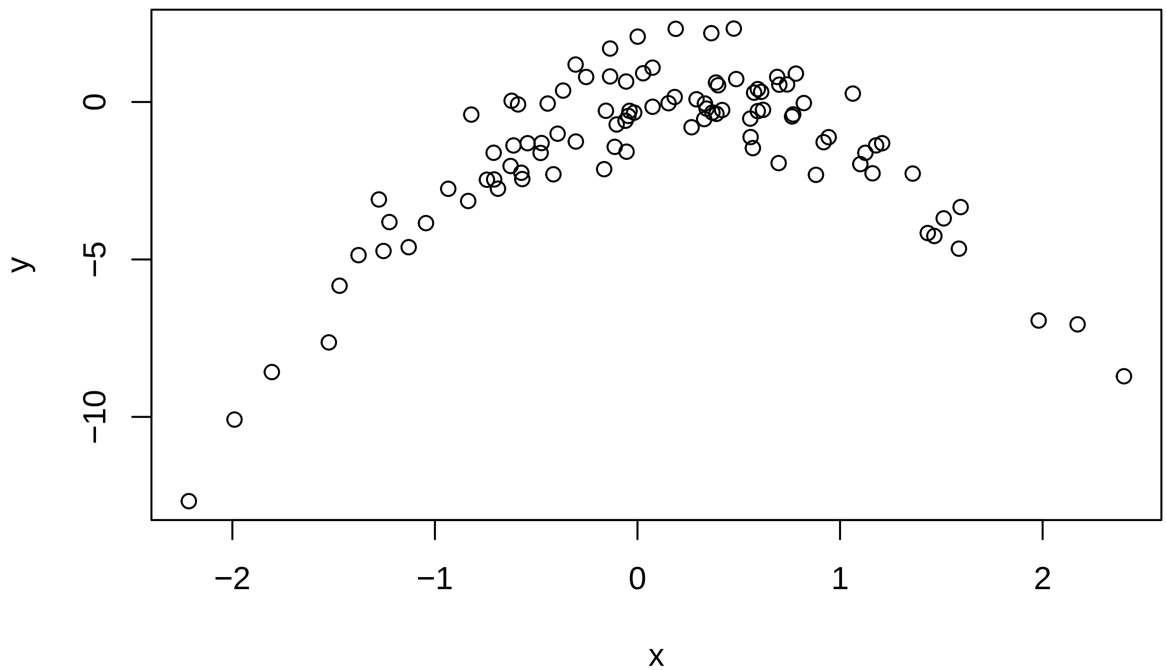
```r
set.seed(1)
x <- rnorm(100)
y <- x - 2 * x^2 + rnorm(100)
```

In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

```
n (number of data points) = 100, p = 2
model: Y = X - 2 * X^2 + error
```

b) Create a scatterplot of X against Y . Comment on what you find.

```r
plot(x, y)
```




```
The shape of the scatterplot is parabolic which is expected
given the equation was quadratic. Points are a little scattered /
stray away from the main shape towards the top of the graph
```

c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

```r
library(ISLR)
library(boot)
set.seed(2)
dataf <- data.frame(x,y)
cv.error <- rep(0, 4)

for (i in (1:4)) {
  glm.fit <- glm(y ~ poly(x, i), data = dataf)
  cv.error[i] <- cv.glm(dataf, glm.fit)$delta[1]
}

cv.error
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

```
i. Y = B0 + B1X + e, error = 7.2881616
ii. Y = B0 + B1X + B2X^2 + e, error = 0.9374236
iii. Y = B0 + B1X + B2X^2 + B3X^3 + e, error = 0.9566218
iv. Y = B0 + B1X + B2X^2 + B3X^3 + B4X^4 + e, error = 0.9539049
```

```
d) Repeat (c) using another random seed, and report your results.
   Are your results the same as what you got in (c)? Why?
```

```r
set.seed(72)
dataf2 <- data.frame(x,y)
cv.error2 <- rep(0, 4)

for (j in (1:4)) {
  glm.fit2 <- glm(y ~ poly(x, j), data = dataf2)
  cv.error2[j] <- cv.glm(dataf2, glm.fit2)$delta[1]
}

cv.error2
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

```
i. Y = B0 + B1X + e, error = 7.2881616
ii. Y = B0 + B1X + B2X^2 + e, error = 0.9374236
iii. Y = B0 + B1X + B2X^2 + B3X^3 + e, error = 0.9566218
iv. Y = B0 + B1X + B2X^2 + B3X^3 + B4X^4 + e, error = 0.9539049

The results from this compared to the results in part c do not differ.
Nothing really changed in terms of the process so using another
random seed had no real effect.
```

```
e) Which of the models in (c) had the smallest LOOCV error?
   Is this what you expected? Explain your answer.

     The 2nd model / quadratic model had the smallest LOOCV error as this was
     the model used to generate the data initially
```

```
f) Comment on the statistical significance of the coefficient estimates that
   results from fitting each of the models in (c) using least squares.
   Do these results agree with the conclusions drawn based on the
   cross-validation results?
```

```r
summary(glm.fit)$coef
```

```
##                  Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)  -1.5500226 0.09590514 -16.1620379 5.169227e-29
## poly(x, i)1   6.1888256 0.95905143   6.4530695 4.590732e-09
## poly(x, i)2 -23.9483049 0.95905143 -24.9708243 1.593826e-43
## poly(x, i)3   0.2641057 0.95905143   0.2753822 7.836207e-01
```

```
## poly(x, i)4   1.2570950 0.95905143   1.3107691 1.930956e-01
```

```
    Both linear and quadratic are statistically significant (P < 0.5)
    while the higher orders are not (P > 0.5) which agree with the
    conclusions drawn on the cross-validation results.
```

Chapter 6, Question 3:

Suppose we estimate the regression coefficients in a linear regression model by minimizing for a particular value of lambda. For parts (a) through (e), indicate which of i. through v. is correct. Justify your answer.

i. Increase initially, and then eventually start decreasing in an inverted U shape.
ii. Decrease initially, and then eventually start increasing in a U shape.
iii. Steadily increase.
iv. Steadily decrease.
v. Remain constant.

As we increase lambda from 0…

```
a) Training RSS:
    as lambda increases from 0, the impact of the shrinkage penalty grows,
    and the ridge regression coefficient estimates will approach zero therefore
    training RSS steadily decreases (iv).

b) Test RSS:
    the test RSS will decrease like the training RSS but will start to
    increase in a U shape because the continuous increase in lanbda will cause
    the model to start overfitting (ii).

c) Variance:
    variance will steadlily increase as lambda increases because there will
    be more predictor variales added to the model. More variables leads to
    better fitting until it starts overfitting which makes variance increase
    (iii).

d) Squared bias:
    squared bias will steadily decrease as a result of the increase of lambda
    from zero since the model will have more predictor variables which leads
    to better fitting and decreasing bias (iv).

e) Irreducible error:
    irreducible error will remain constant since it has nothing to do with the
    model (v).
```

Chapter 6, Question 9:

In this exercise, we will predict the number of applications received using the other variables in the College data set.

```
data <- "/Users/melchorronquillo/Desktop/Files/R code/College.csv"
College <- data.frame(read.csv(data))
#College
```

```
a) Split the data set into training and test set:
```

```
set.seed(3889)
# randomly scramble data
u <- runif(nrow(College))
```

```r
#u

#split to 70% training and 30% testing data
train <- College[u <= 0.7,]
test <- College[u >  0.7,]
#train
#test
```

b) Fit a linear model using least squares on the training set, and report the test error obtained.

```r
#train linear model
lm.train = lm(Apps ~ ., data = train)
#apply trained model on test data
lm.test = predict(lm.train, test, type = 'response')
#find test error by taking mean of squared difference of predictions and actual
  # Mean squared error
lm.err = mean((lm.test - test$Apps)^2)
lm.err
```

```
## [1] 822231.4
```

Linear model test error = 822231.4

c) Fit a ridge regression model on the training set, with lambda chosen by cross-validation. Report the test error obtained.

```r
# must pass in an x matrix as well as a y vector
xmat <- model.matrix(Apps ~ ., data = train)[, -1]
yvect <- train$Apps

# The glmnet() function has an alpha argument that determines what type of model
# is fit. Alpha = 0 = ridge regression, Alpha = 1 =  lasso model.
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-4
```

```r
# cv.glmnet() uses cross-validation to choose the tuning parameter lambda
cv.out <- cv.glmnet(xmat, yvect, alpha = 0)
#plot(cv.out)

# find which value of lambda results in smallesr CV error
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 384.1338
```

lambda chosen by cross validation = 384.1338

```r
# create test matrix
xtst <- model.matrix(Apps ~ ., data = test)[, -1]

# train ridge with training data, alpha = 0 for ridge, lambda = found from CV
```

```r
ridge.mod <- glmnet(xmat, yvect, alpha = 0, lambda = bestlam)

# predict on test data
ridge.pred <- predict(ridge.mod, s = bestlam, newx = xtst)

# find test error
ridge.err = mean((ridge.pred - test$Apps)^2)
ridge.err
```

```
## [1] 873087.4
```

    Ridge test error = 873087.4

d) Fit a lasso model (pg.278) on the training set, with lambda chosen by cross- validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```r
# repeat finding lambda by cross validation for lasso
cv.out_lasso <- cv.glmnet(xmat, yvect, alpha = 1)
#plot(cv.out.lasso)

bestlam_lasso <- cv.out_lasso$lambda.min
bestlam_lasso
```

```
## [1] 9.082804
```

    lambda chosen by cross validation = 23.02822

```r
# repeat process from ridge with lasso
lasso.mod <- glmnet(xmat, yvect, alpha = 1, lambda = bestlam_lasso)
lasso.pred <- predict(lasso.mod, s = bestlam, newx = xtst)
lasso.err = mean((lasso.pred - test$Apps)^2)
lasso.err
```

```
## [1] 819381.5
```

```r
# refit model on the full data set using the value of lambda chosen by
  # cross-validation and examine the coefficient estimates.
coef = predict(lasso.mod, type = "coefficients", s = bestlam)[1:18, ]
coef
```

```
##   (Intercept)    PrivateYes        Accept        Enroll     Top10perc
## -787.55150661 -495.02112569    1.55933706   -0.54032691   40.60730680
##     Top25perc    F.Undergrad   P.Undergrad      Outstate    Room.Board
##   -6.91070138     0.00000000    0.05446115   -0.08171715    0.15159243
##         Books      Personal           PhD      Terminal     S.F.Ratio
##    0.01064362     0.05575801   -5.58279489   -4.06435274   14.09492981
##   perc.alumni        Expend     Grad.Rate
##    2.85489411     0.08322920    6.78062119
```

    Lasso test error = 828416.6
    Number of non-zero coefficient estimates = 13

e) Fit a PCR model on the training set, with M chosen by cross- validation. Report the test error obtained, along with the value of M selected by cross-validation.

```r
# Principal components regression (PCR) can be performed using the pcr() function
# which is part of the pls library
install.packages("pls", repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/b4/vbzhgztj3tj299xgxnklp28c0000gn/T//RtmpBqd5kD/downloaded_packages
```

```r
library(pls)
```

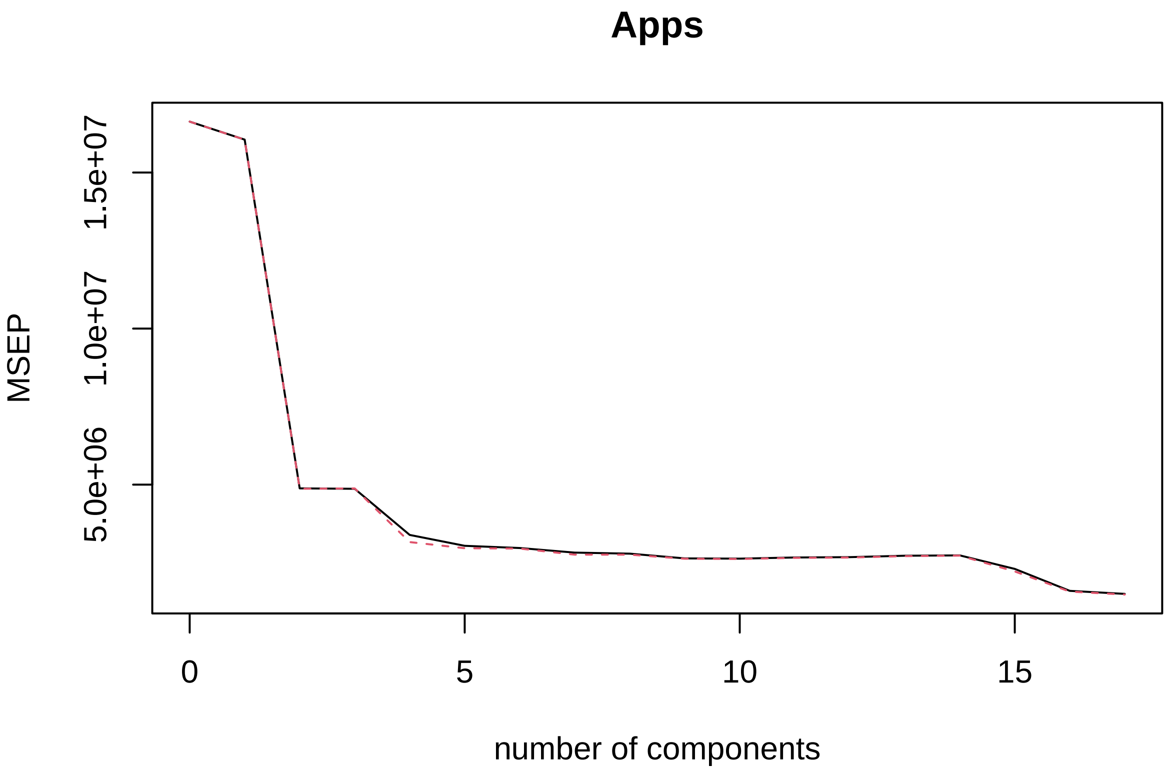
```
##
## Attaching package: 'pls'

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

```r
set.seed(123)

# fit training data to pcr(), similar to lm() but adding scale = "True" standardizes
  # each predictor, validation = "CV" causes pcr() to compute 10-fold CV error
  # for each M
pcr.fit <- pcr(Apps ~ ., data = train, scale = TRUE, validation = "CV")

# use summary to find where M components have smallest / biggest decrease in
  # CV error
summary(pcr.fit)
```

```
## Data:    X dimension: 556 17
##  Y dimension: 556 1
## Fit method: svdpc
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            4078     4007     2209     2206     1841     1744     1723
## adjCV         4078     4007     2206     2209     1778     1721     1718
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1680     1670     1624      1622      1633      1636      1650
## adjCV     1660     1660     1620      1618      1629      1632      1646
##        14 comps  15 comps  16 comps  17 comps
## CV         1653      1517      1264      1225
## adjCV      1651      1490      1254      1216
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X      31.699    57.15    63.74    69.54    75.14    80.34    83.86    87.31
## Apps    4.472    71.66    72.56    82.21    83.16    83.18    84.49    84.54
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       90.39     92.84     94.90     96.78     97.87     98.70     99.35
## Apps    85.00     85.14     85.15     85.16     85.19     85.24     91.26
##       16 comps  17 comps
```

```
## X          99.82    100.00
## Apps       92.62     92.97
```

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



Based on summary and plot, it appears that M = 10 is where the smallest corss-validation error occurs with the CV error = 1622. It is smaller than M = 11 with CV error = 1633 and M = 13 with CV error = 1624. Anything greater than M = 10 would mean including more components than needed and would not really accomplish dimension reduction.

```
# fit test data on trained pcr model using ncomp = M where samllest CV error occurs
pcr.pred <- predict(pcr.fit, test, ncomp = 10)
#find test error of PCR
pcr.err = mean((pcr.pred - test$Apps)^2)
pcr.err
```

```
## [1] 1212949
```

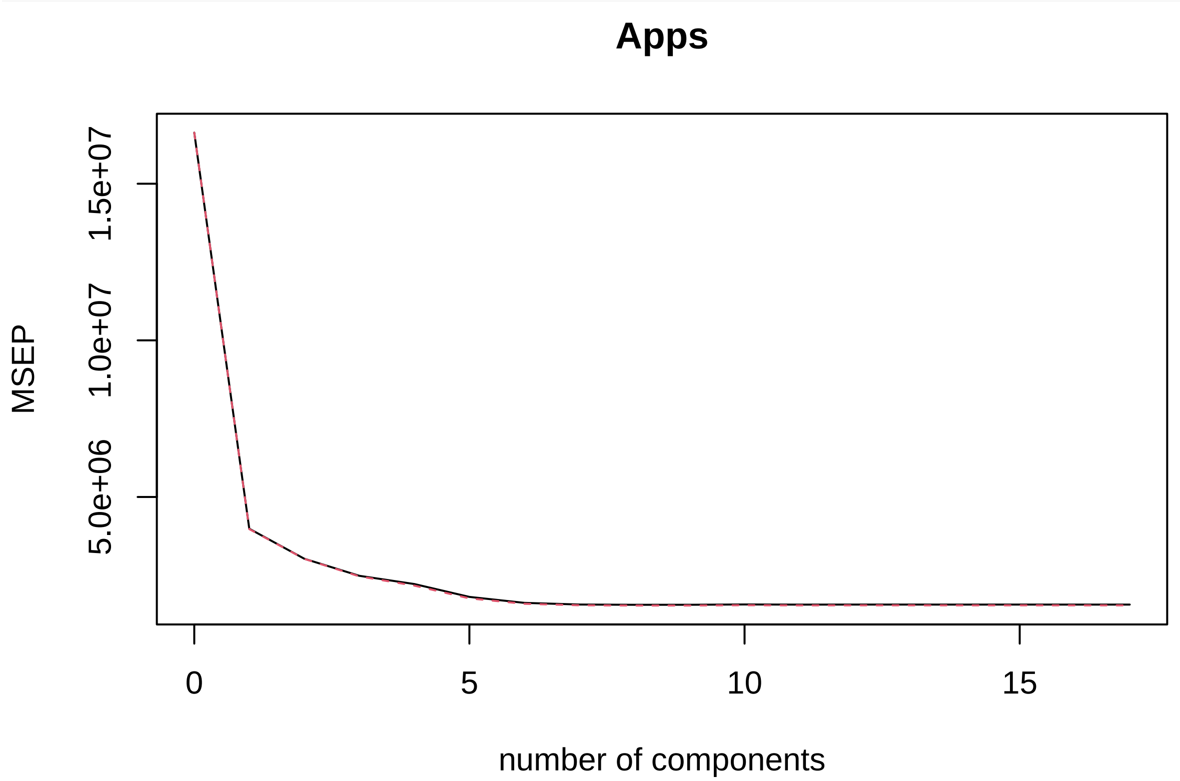
PCR test error : 121949

f) Fit a PLS model on the training set, with M chosen by cross- validation. Report the test error obtained, along with the value of M selected by cross-validation.

```
# similar process to PCR, just replace with plsr()
set.seed(124)
pls.fit <- plsr(Apps ~ ., data = train,scale = TRUE, validation = "CV")
summary(pls.fit)
```

```
## Data:     X dimension: 556 17
##  Y dimension: 556 1
## Fit method: kernelpls
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            4078     1997     1740     1576     1490     1345     1272
## adjCV         4078     1993     1739     1570     1473     1328     1261
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1252     1248     1248      1253      1251      1251      1251
## adjCV     1242     1239     1239      1243      1241      1241      1241
##        14 comps  15 comps  16 comps  17 comps
## CV         1251      1251      1251      1251
## adjCV      1241      1241      1241      1241
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       25.86    43.99    62.69    65.28    68.51    73.41    76.13    79.22
## Apps    76.94    83.59    87.14    90.66    92.45    92.77    92.83    92.89
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       82.27     84.72     87.21     90.66     92.31     93.82     96.85
## Apps    92.93     92.95     92.96     92.97     92.97     92.97     92.97
##       16 comps  17 comps
## X        98.25    100.00
## Apps     92.97     92.97
```

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



```
Since 8 and 9 returned the excact same values for CV error, M = 8
is what I will use since the CV error dropped at M = 8 first. The same
logic in finding M for PLS applises for PCR, as the smallest
cross-validation error occurs at M=8 with CV error = 1248 which is
less than M=7 with CV error = 1252 and M=10 with CV error = 1253
```

```r
pls.pred <- predict(pls.fit, test, ncomp = 8)
pls.err = mean((pls.pred - test$Apps)^2)
pls.err
```

```
## [1] 840329.1
```

```
PLS test error = 840329.1
```

g) Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

```
Linear model test error = 822231.4
Ridge test error = 873087.4
Lasso test error = 828416.6
PCR test error : 121949
PLS test error = 840329.1

There are no real major differences among the test errors besides the PCR
test error which had an extremely high error. Accuracy of each model can
be determined by taking the R-squared of each:
```

```r
# R^2 = 1 - (model test error / actual data test error)
  # R-squared is how well the regression model explains observed data.

# take average of Apps in test data, get MSE as actual test error
avg.apps <- mean(test$Apps)
#avg.apps
actual.err <- mean((avg.apps - test$Apps)^2)
#actual.err

lmR2 <- (1 - (lm.err / actual.err))
lmR2
```

```
## [1] 0.9236275
```

```r
ridgeR2 <- (1 - (ridge.err / actual.err))
ridgeR2
```

```
## [1] 0.9189038
```

```r
lassoR2 <- (1 - (lasso.err / actual.err))
lassoR2
```

```
## [1] 0.9238923
```

```r
pcrR2 <- (1 - (pcr.err / actual.err))
pcrR2
```

```
## [1] 0.887336
```

```r
plsR2 <- (1 - (pls.err / actual.err))
plsR2
```

```
## [1] 0.9219465
```

```
Linear model R-Squared = 0.9236275
Ridge R-Squared = 0.9189038
Lasso R-Squared = 0.923053
PCR R-Squared = 0.887336
PLS R-Squared = 0.9219465

Since most of the models have R-squared values close to 1, it means that
these models can fit the data pretty well, resulting in pretty accurate
predictions.
```