Homework 2

Hyungkyu Lim

Question 1

(10 points) (Exercise 9 modified, ISL) In this exercise, we will predict the numberof applicatiions received using the other variables in the College data set in the ISLR package. (a) Split the data set into a training set and a test set. Fit a linear model using least squares on the training set, and report the test error obtained.

```r
library(ISLR)
attach(College)
set.seed(12345)

split_data <- sample(nrow(College), nrow(College) * 0.7)
train_model <- College[split_data, ]
test_model <- College[-split_data, ]


ls_model <- lm(Apps ~ ., data = train_model)
summary(ls_model)
```

```
##
## Call:
## lm(formula = Apps ~ ., data = train_model)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4833.1  -413.7   -35.8   283.4  7286.8
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept) -946.90287  507.48834  -1.866  0.06262 .
## PrivateYes  -415.61700  169.04301  -2.459  0.01427 *
## Accept         1.63147    0.04597  35.487  < 2e-16 ***
## Enroll        -0.98277    0.23754  -4.137 4.09e-05 ***
## Top10perc     47.93095    7.08030   6.770 3.46e-11 ***
## Top25perc    -13.69648    5.65994  -2.420  0.01586 *
## F.Undergrad    0.04475    0.04333   1.033  0.30215
## P.Undergrad    0.04527    0.04628   0.978  0.32845
## Outstate      -0.08496    0.02389  -3.556  0.00041 ***
## Room.Board     0.14311    0.05700   2.511  0.01235 *
## Books          0.16812    0.27510   0.611  0.54138
## Personal       0.09890    0.07683   1.287  0.19860
## PhD          -10.76883    5.94368  -1.812  0.07059 .
## Terminal      -0.71242    6.26472  -0.114  0.90950
## S.F.Ratio     33.81281   17.58229   1.923  0.05501 .
## perc.alumni    2.46075    5.03070   0.489  0.62494
## Expend         0.08387    0.01436   5.843 9.02e-09 ***
## Grad.Rate      7.53478    3.59343   2.097  0.03649 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

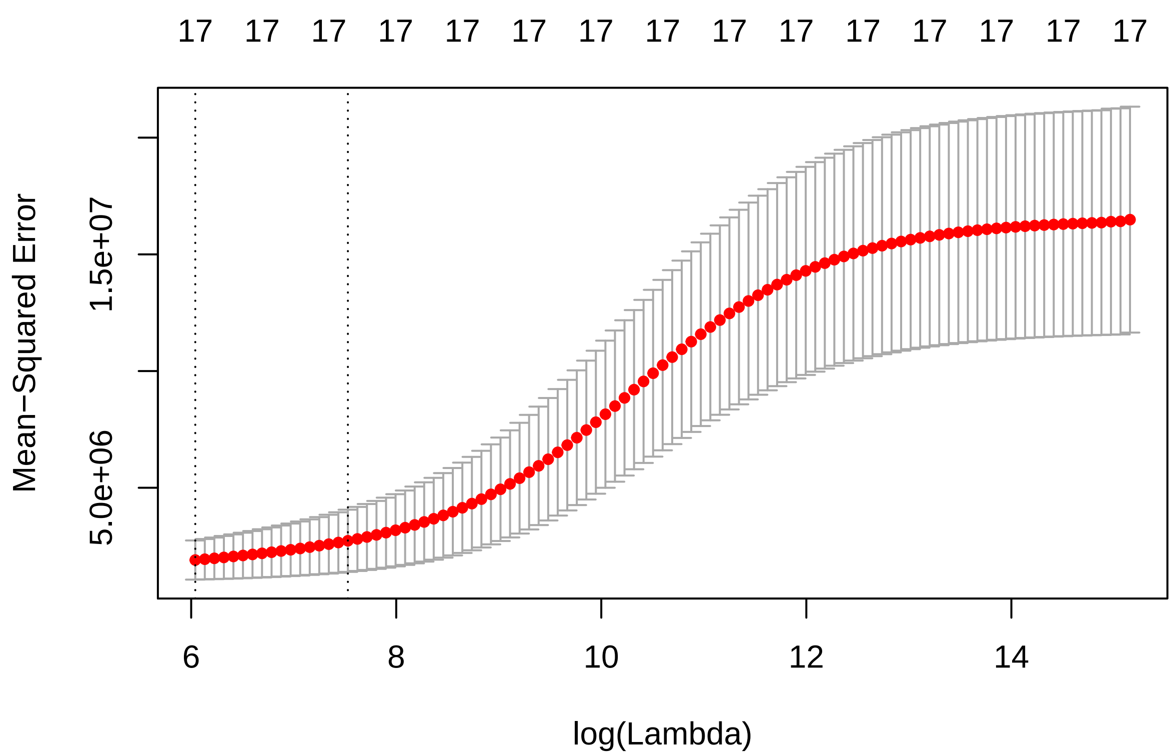
```
## Residual standard error: 1068 on 525 degrees of freedom
## Multiple R-squared:  0.9325, Adjusted R-squared:  0.9304
## F-statistic: 426.9 on 17 and 525 DF,  p-value: < 2.2e-16
```

```
predicted <- predict(ls_model, test_model)

error <- mean((test_model$Apps - predicted)^2)

error
```

```
## [1] 992428.6
```

(b) Fit a ridge regression model on the training set, with lambda chosen by crossvalidation. Report the test error obtained.

```
library(glmnet)

train_ridge <- model.matrix(Apps ~ ., data = train_model)
test_ridge <- model.matrix(Apps ~ ., data = test_model)


ridge.mod <- glmnet(train_ridge, train_model$Apps, alpha = 0)

cv.ridge <- cv.glmnet(train_ridge, train_model$Apps, alpha = 0)

plot(cv.ridge)
```




```
ridge_bestlam <- cv.ridge$lambda.min

ridge_bestlam
```

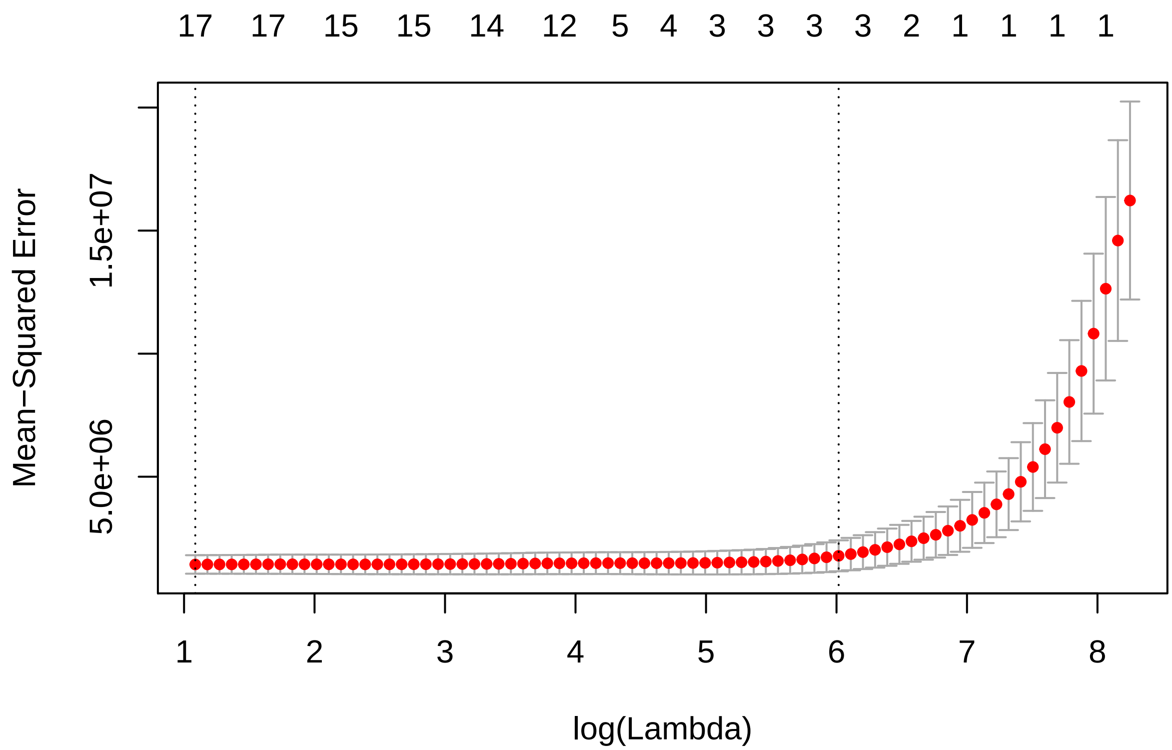
```
## [1] 419.8756
```

```r
pred_ridge <- predict(cv.ridge, s = ridge_bestlam, newx = test_ridge)

ridge_test_error <- mean((test_model$Apps - pred_ridge)^2)

ridge_test_error
```

```
## [1] 1031295
```

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

```r
train_lasso <- model.matrix(Apps ~ ., data = train_model)

test_lasso <- model.matrix(Apps ~ ., data = test_model)

lasso_model <- glmnet(train_lasso, train_model$Apps, alpha = 1)

cv.lasso <- cv.glmnet(train_lasso, train_model$Apps, alpha = 1)

plot(cv.lasso)
```




```r
lasso_bestlam <- cv.lasso$lambda.min

lasso_bestlam
```

```
## [1] 2.962139
```

```r
pred_lasso <- predict(lasso_model, s = lasso_bestlam, newx = test_lasso)

lasso_test_error <- mean((test_model$Apps - pred_lasso)^2)

lasso_test_error
```

```
## [1] 990031.4
```

```r
predict(lasso_model, s = lasso_bestlam, type = "coefficients")
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -983.92017482
## (Intercept)    .
## PrivateYes  -403.76001216
## Accept         1.61066773
## Enroll        -0.80673182
## Top10perc     45.63026643
## Top25perc    -11.84695304
## F.Undergrad    0.01947699
## P.Undergrad    0.04849116
## Outstate      -0.08234905
## Room.Board     0.14000705
## Books          0.16000475
## Personal       0.09434622
## PhD          -10.13472940
## Terminal      -0.68365116
## S.F.Ratio     32.34479867
## perc.alumni    1.41587567
## Expend         0.08323618
## Grad.Rate      7.25037455
```

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

```r
library(pls)
```

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

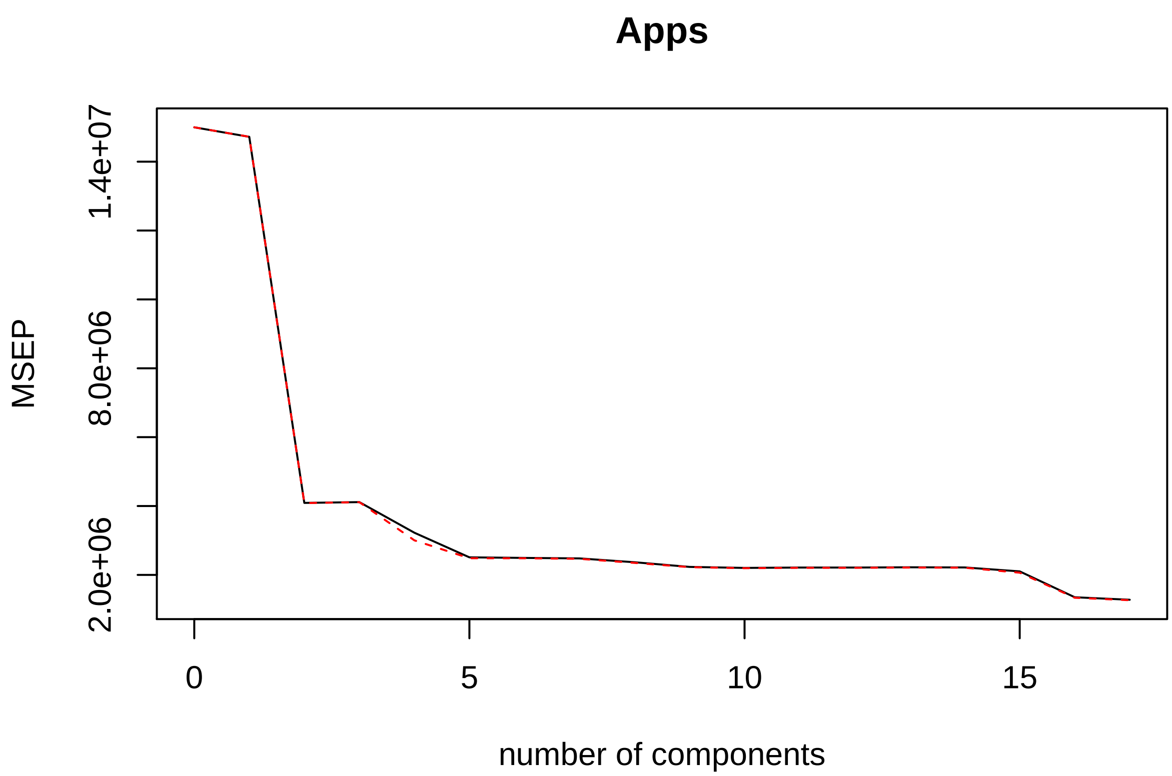
```
## The following object is masked from 'package:stats':
##
##     loadings
```

```r
set.seed(2)

pcr_model = pcr(Apps ~ ., data = College, scale = TRUE, validation = "CV")
summary(pcr_model)
```

```
## Data:    X dimension: 777 17
##  Y dimension: 777 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            3873     3837     2022     2028     1796     1584     1579
## adjCV         3873     3837     2021     2028     1733     1576     1576
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1575     1539     1494      1484      1488      1488      1490
## adjCV     1571     1533     1492      1482      1485      1485      1488
##        14 comps  15 comps  16 comps  17 comps
```

```
## CV         1489       1450       1162       1131
## adjCV      1486       1437       1156       1125
##
## TRAINING: % variance explained
##        1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       31.670    57.30    64.30    69.90    75.39    80.38    83.99
## Apps     2.316    73.06    73.07    82.08    84.08    84.11    84.32
##        8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X        87.40    90.50     92.91     95.01     96.81      97.9     98.75
## Apps     85.18    85.88     86.06     86.06     86.10      86.1     86.13
##        15 comps  16 comps  17 comps
## X         99.36     99.84    100.00
## Apps      90.32     92.52     92.92
```

```
validationplot(pcr_model, val.type = "MSEP")
```

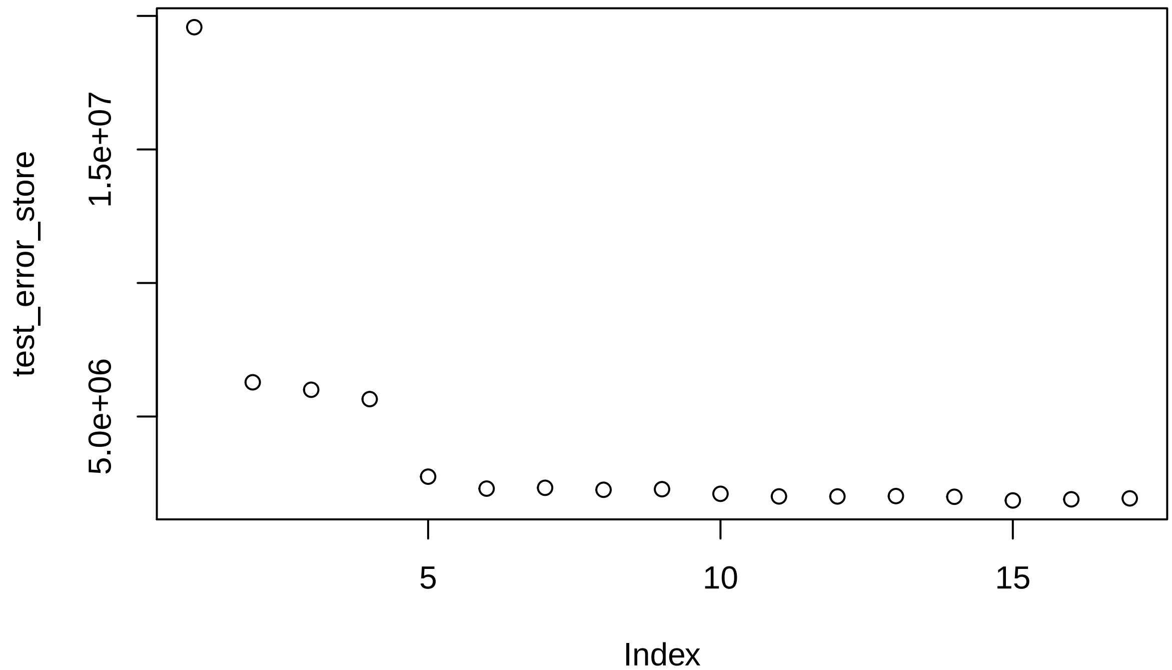



```
train <- sample(nrow(College), nrow(College) * 0.7)
test = -train
y_test = College$Apps[test]
y_train = College$Apps[train]


pcr_model <- pcr(Apps ~ ., data = College, subset = train, sclae = TRUE,
    validation = "CV")
summary(pcr_model)
```

```
## Data:     X dimension: 543 17
##  Y dimension: 543 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            3565     3512     1472     1466     1429     1130     1046
## adjCV         3565     3546     1470     1464     1427     1125     1044
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1046     1047     1048      1022     994.8     996.2     993.5
## adjCV     1043     1045     1045      1019     992.1     993.5     990.8
##        14 comps  15 comps  16 comps  17 comps
## CV        994.3     975.6     970.6     956.9
## adjCV     991.6     972.5     967.4     953.6
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X      47.281    88.22    95.02    97.51    98.60    99.39    99.91
## Apps    5.416    83.51    83.76    85.05    90.84    92.05    92.06
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       99.96   100.00    100.00    100.00    100.00    100.00    100.00
## Apps    92.14    92.15     92.55     92.92     92.92     92.97     92.99
##       15 comps  16 comps  17 comps
## X       100.00    100.00    100.00
## Apps     93.33     93.43     93.65
```

```r
training_error_store <- c()
test_error_store <- c()
for (i in 1:17) {
    pcr_pred_test = predict(pcr_model, College[test, ], ncomp = i)
    test_error <- mean((pcr_pred_test - y_test)^2)
    test_error_store <- c(test_error_store, test_error)
}


plot(test_error_store)
```




```
pcr_pred = predict(pcr_model, College[test, ], cnomp = 5)

pcr_test_error <- mean((pcr_pred - y_test)^2)
```

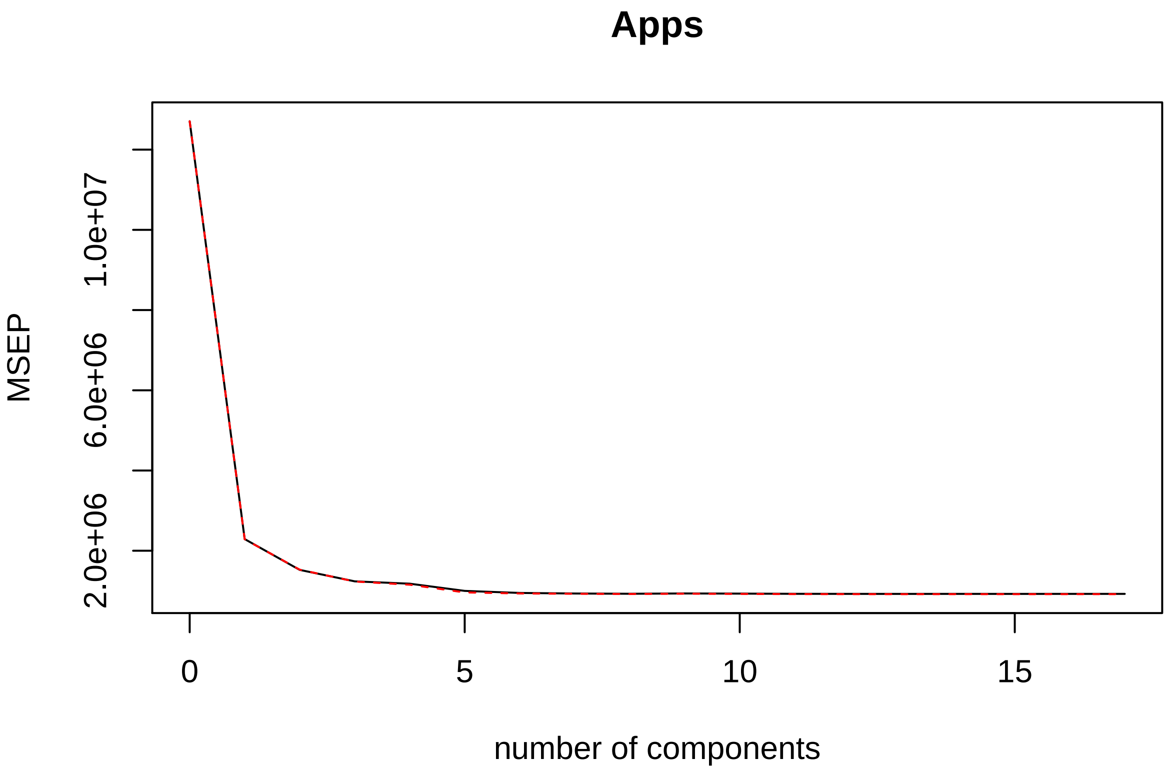
(f) Fit a PLS model on the training set, with k chosen by crossvalidation. Report the test error obtained, along with the value of k selected by cross-validation

```
pls_model = plsr(Apps ~ ., data = College, subset = train, scale = TRUE,
    validation = "CV")
summary(pls_model)
```

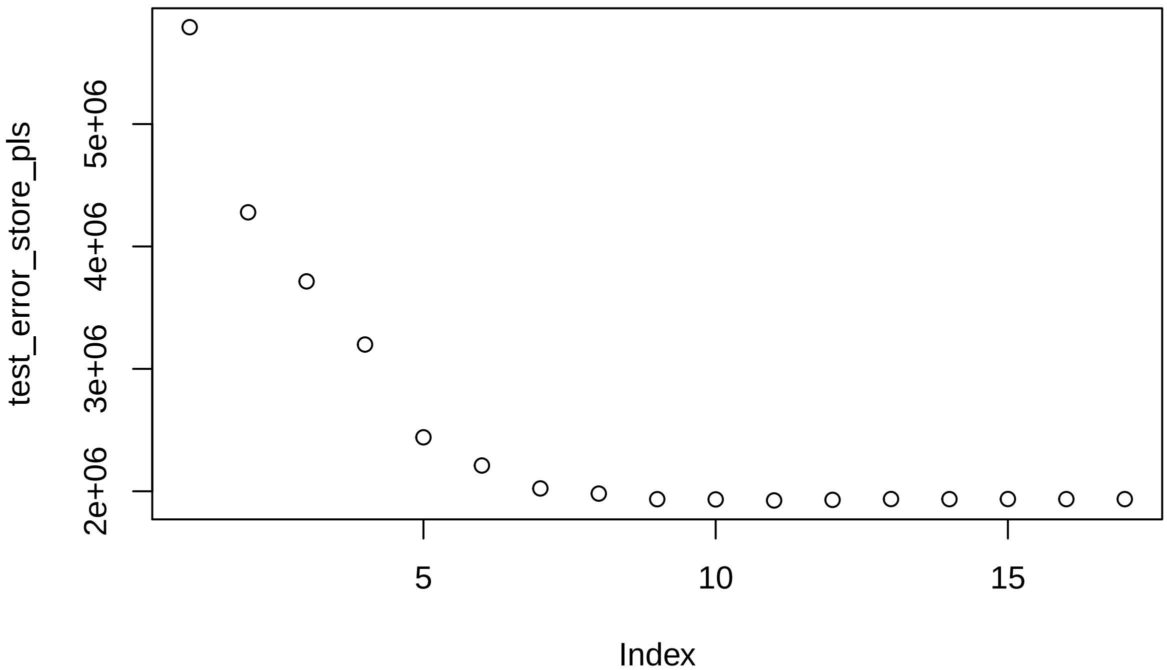
```
## Data:    X dimension: 543 17
##  Y dimension: 543 1
## Fit method: kernelpls
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            3565     1514     1234     1112     1086    999.4    973.5
## adjCV         3565     1512     1236     1110     1074    980.3    967.0
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       966.3    962.9    967.0     964.9     962.2     961.4     961.4
## adjCV    962.4    959.9    963.5     961.0     958.6     957.9     957.9
##        14 comps  15 comps  16 comps  17 comps
## CV        961.6     961.3     961.4     961.4
## adjCV     958.1     957.9     957.9     957.9
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       26.37    42.54    62.97    65.73    67.69    72.13    76.29
## Apps    82.28    88.31    90.76    92.01    93.20    93.43    93.53
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
```

```
## X        81.45    84.12     85.53     88.71     91.70     93.08     95.18
## Apps     93.55    93.58     93.63     93.64     93.65     93.65     93.65
##       15 comps  16 comps  17 comps
## X         97.29     99.11    100.00
## Apps      93.65     93.65     93.65
```

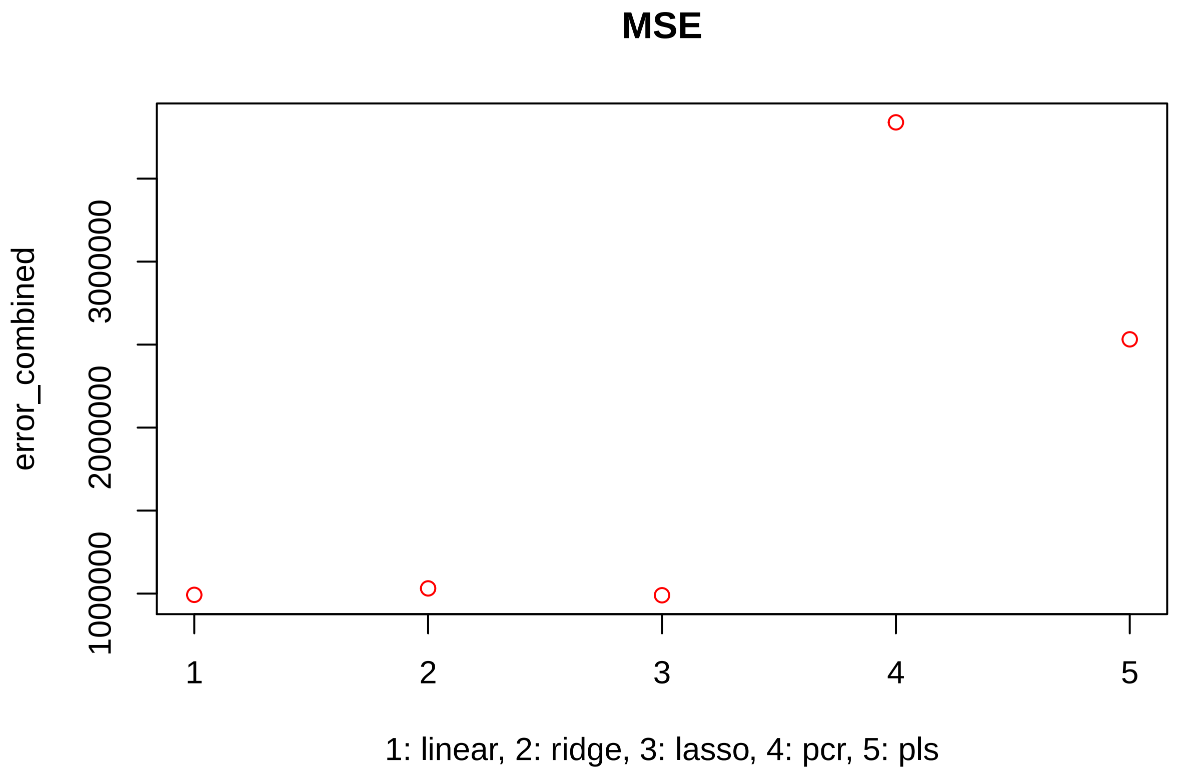
```
validationplot(pls_model, val.type = "MSEP")
```




```
training_error_store_pls <- c()
test_error_store_pls <- c()
for (i in 1:17) {
    pls_pred_test = predict(pls_model, College[test, ], ncomp = i)
    test_error_pls <- mean((pls_pred_test - y_test)^2)
    test_error_store_pls <- c(test_error_store_pls, test_error_pls)
}


plot(test_error_store_pls)
```




```
pls_pred = predict(pls_model, College[test, ], cnomp = 8)

pls_test_error <- mean((pls_pred - y_test)^2)


error_combined <- c(error, ridge_test_error, lasso_test_error,
    pcr_test_error, pls_test_error)


plot(error_combined, main = "MSE", xlab = "1: linear, 2: ridge, 3: lasso, 4: pcr, 5: pls",
    col = "red")
```



##Question 2 (10 points) The insurance company benchmark data set gives information on customers. Specifically, it contains 86 variables on product-usage data and sociodemographic data derived from zip area codes. There are 5,822 customers in the training set and another 4,000 in the test set. The data were collected to answer the following questions: Can you predict who will be interested in buying a caravan insurance policy and give an explanation why? Compute the OLS estimates and compare them with those obtained from the following variableselection algorithms: Forwards Selection, Backwards Selection, Lasso regression, and Ridge regression. Support your answer.

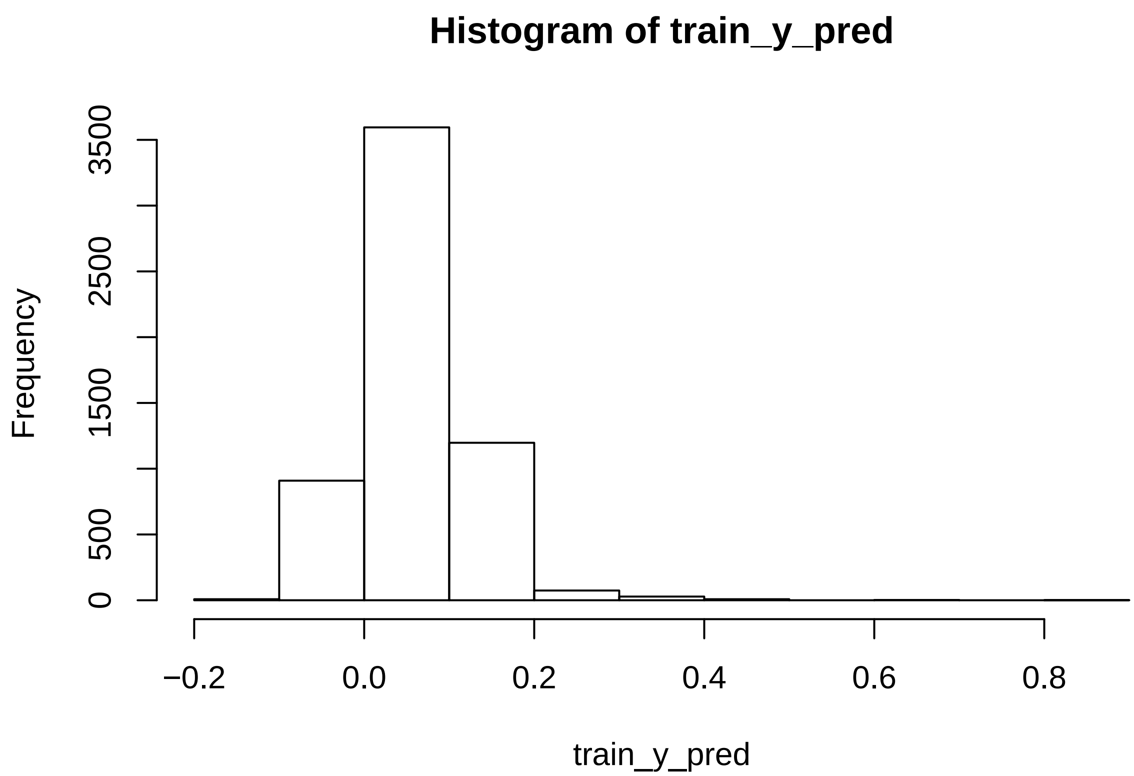
```
train_data <- read.table("~/desktop/EAS506/ticdata2000.txt")
train_y <- train_data$V86
train_x <- train_data[-86]


test_x <- read.table("~/desktop/EAS506/ticeval2000.txt")
test_y <- read.table("~/desktop/EAS506/tictgts2000.txt")

linear_mod <- lm(V86 ~ ., data = train_data)
summary(linear_mod)
```

```
##
## Call:
## lm(formula = V86 ~ ., data = train_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.67293 -0.08720 -0.04593 -0.00639  1.04628
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.7685381  0.4298406   1.788 0.073835 .
## V1           0.0035209  0.0022512   1.564 0.117866
```

```
## V2          -0.0072642  0.0076739  -0.947 0.343875
## V3          -0.0012739  0.0071737  -0.178 0.859055
## V4           0.0107473  0.0049596   2.167 0.030279 *
## V5          -0.0154869  0.0101044  -1.533 0.125405
## V6          -0.0056016  0.0056016  -1.000 0.317353
## V7          -0.0002069  0.0060664  -0.034 0.972795
## V8           0.0003569  0.0054592   0.065 0.947874
## V9          -0.0030237  0.0058038  -0.521 0.602399
## V10          0.0086829  0.0075479   1.150 0.250036
## V11          0.0020367  0.0072008   0.283 0.777310
## V12          0.0055682  0.0076295   0.730 0.465526
## V13         -0.0038250  0.0065474  -0.584 0.559107
## V14         -0.0050625  0.0066861  -0.757 0.448980
## V15         -0.0026253  0.0069795  -0.376 0.706824
## V16          0.0021357  0.0068161   0.313 0.754038
## V17         -0.0048456  0.0071396  -0.679 0.497358
## V18         -0.0113977  0.0073004  -1.561 0.118525
## V19          0.0021884  0.0045182   0.484 0.628153
## V20         -0.0004665  0.0052201  -0.089 0.928796
## V21         -0.0050974  0.0050426  -1.011 0.312122
## V22          0.0041254  0.0044806   0.921 0.357228
## V23         -0.0006060  0.0044709  -0.136 0.892190
## V24          0.0019733  0.0044532   0.443 0.657690
## V25         -0.0013674  0.0051653  -0.265 0.791225
## V26         -0.0031701  0.0050198  -0.632 0.527724
## V27         -0.0012603  0.0044827  -0.281 0.778603
## V28          0.0024879  0.0049115   0.507 0.612502
## V29         -0.0008866  0.0047145  -0.188 0.850832
## V30         -0.0454201  0.0376622  -1.206 0.227872
## V31         -0.0432242  0.0376290  -1.149 0.250730
## V32          0.0085964  0.0075592   1.137 0.255502
## V33          0.0077871  0.0068554   1.136 0.256038
## V34          0.0047215  0.0072646   0.650 0.515762
## V35         -0.0561024  0.0444643  -1.262 0.207094
## V36         -0.0593733  0.0443897  -1.338 0.181097
## V37          0.0070879  0.0051150   1.386 0.165884
## V38          0.0069414  0.0049276   1.409 0.158986
## V39          0.0049679  0.0050144   0.991 0.321862
## V40          0.0059267  0.0052728   1.124 0.261053
## V41         -0.0098939  0.0069270  -1.428 0.153258
## V42          0.0063044  0.0045645   1.381 0.167277
## V43          0.0029097  0.0022664   1.284 0.199250
## V44          0.0284931  0.0166017   1.716 0.086166 .
## V45         -0.0101533  0.0205121  -0.495 0.620625
## V46         -0.0201220  0.0390424  -0.515 0.606301
## V47          0.0102787  0.0026346   3.901 9.67e-05 ***
## V48          0.0014405  0.0148574   0.097 0.922765
## V49         -0.0061279  0.0079415  -0.772 0.440364
## V50         -0.0249190  0.0415892  -0.599 0.549083
## V51          0.0588044  0.0557610   1.055 0.291662
## V52          0.0121481  0.0142358   0.853 0.393504
## V53         -0.0062440  0.0370186  -0.169 0.866060
## V54          0.0078683  0.0152793   0.515 0.606598
## V55         -0.0155397  0.0064753  -2.400 0.016433 *
```

```
## V56          0.0098926  0.0335157   0.295 0.767880
## V57          0.1937254  0.0793370   2.442 0.014644 *
## V58          0.0647933  0.0256913   2.522 0.011696 *
## V59          0.0132643  0.0035906   3.694 0.000223 ***
## V60         -0.1917507  0.1439848  -1.332 0.182998
## V61         -0.0299076  0.0269224  -1.111 0.266666
## V62         -0.0107777  0.0549693  -0.196 0.844564
## V63         -0.0441620  0.0307404  -1.437 0.150883
## V64         -0.0184858  0.0288890  -0.640 0.522269
## V65         -0.0377952  0.0323794  -1.167 0.243154
## V66          0.0185448  0.0529740   0.350 0.726296
## V67          0.0180904  0.1374585   0.132 0.895300
## V68          0.0002821  0.0127496   0.022 0.982347
## V69         -0.0214816  0.0652955  -0.329 0.742175
## V70          0.0203252  0.0310683   0.654 0.513004
## V71          0.0563675  0.1589388   0.355 0.722866
## V72         -0.0804238  0.0944352  -0.852 0.394455
## V73         -0.0395651  0.0353795  -1.118 0.263484
## V74         -0.0010526  0.0728240  -0.014 0.988468
## V75         -0.0236462  0.0467611  -0.506 0.613101
## V76          0.0372344  0.0154024   2.417 0.015661 *
## V77         -0.0464279  0.0954471  -0.486 0.626684
## V78         -0.4050642  0.1898715  -2.133 0.032938 *
## V79         -0.2304561  0.1243310  -1.854 0.063852 .
## V80         -0.0211374  0.0116048  -1.821 0.068593 .
## V81          0.4958051  0.2815591   1.761 0.078304 .
## V82          0.3633887  0.0885318   4.105 4.11e-05 ***
## V83          0.0416061  0.0408644   1.018 0.308650
## V84          0.0959436  0.0699079   1.372 0.169983
## V85          0.1312250  0.0983836   1.334 0.182319
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.23 on 5736 degrees of freedom
## Multiple R-squared:  0.0729, Adjusted R-squared:  0.05916
## F-statistic: 5.306 on 85 and 5736 DF,  p-value: < 2.2e-16
```

```r
train_y_pred <- predict(linear_mod, newdata = train_x)


hist(train_y_pred)
```




```
which(train_y_pred > 0.5)
```

```
##  131  763 2954
##  131  763 2954
```

```
summary(train_y_pred)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.17130  0.01668  0.05396  0.05977  0.09501  0.83797
```

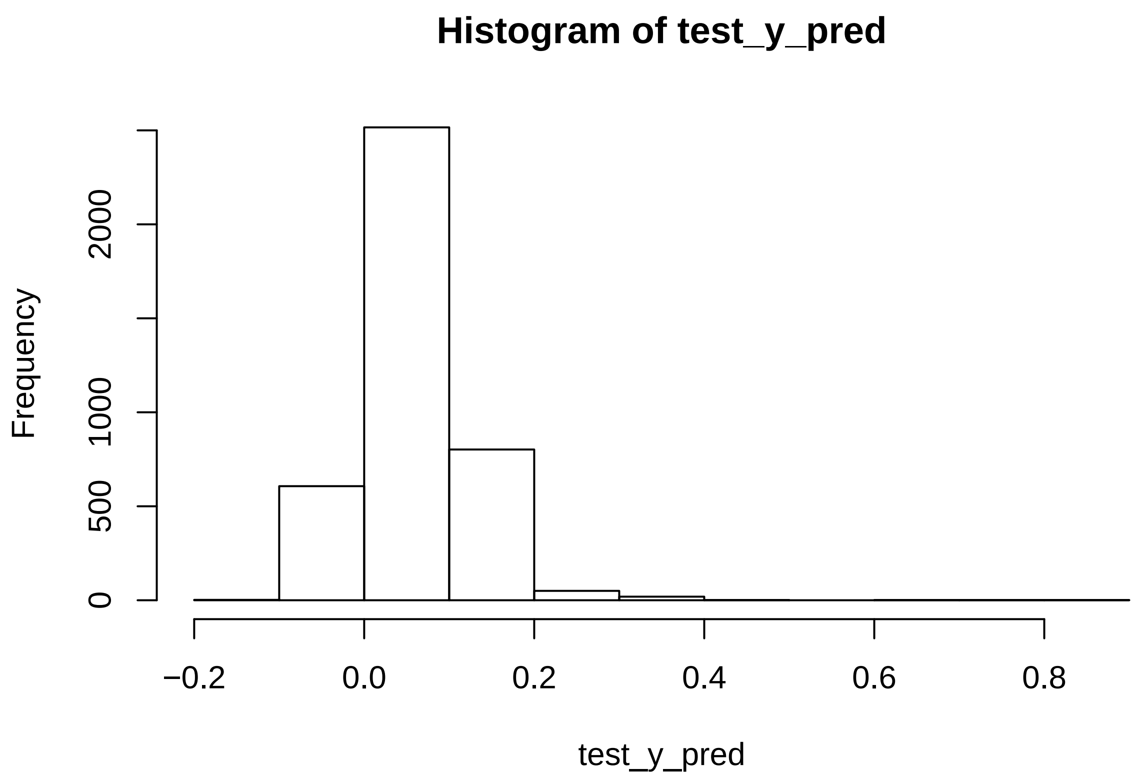
```
train_error_lm <- mean((train_y_pred - train_y)^2)

train_error_lm
```

```
## [1] 0.05210329
```

```
test_y_pred <- predict(linear_mod, newdata = test_x)

summary(test_y_pred)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.12796  0.01631  0.05435  0.05929  0.09379  0.80824
```

```
hist(test_y_pred)
```




```
max(test_y_pred)
```

```
## [1] 0.8082358
```

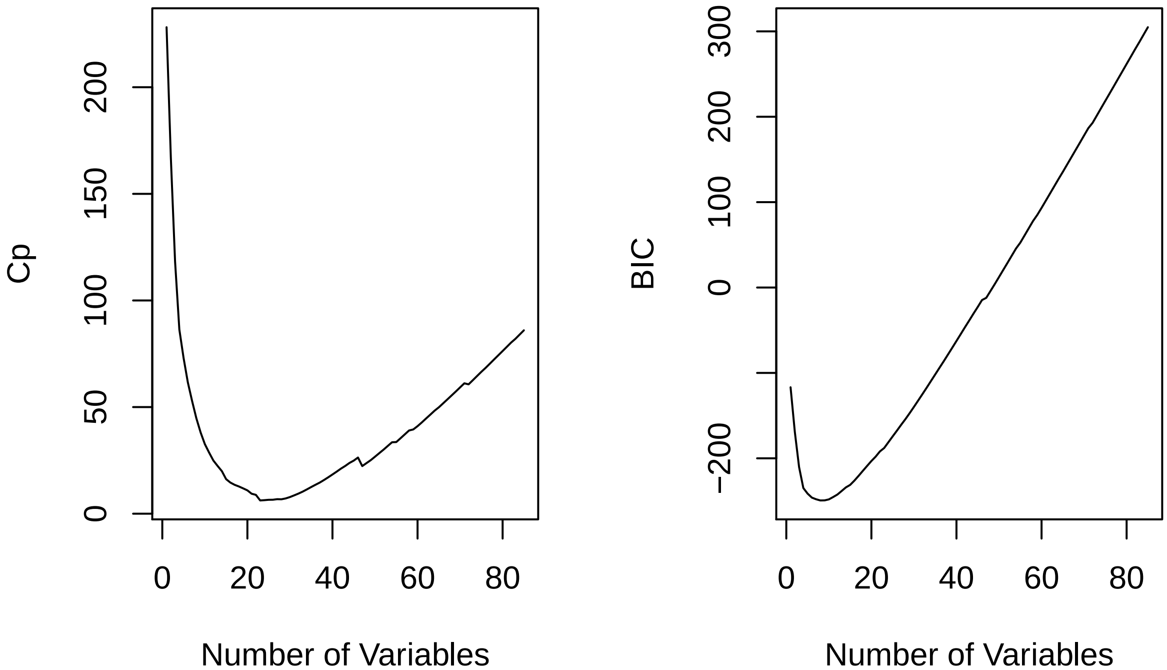
```
which(test_y_pred > 0.5)
```

```
##  576 2863 3139
##  576 2863 3139
```

```
test_error_lm <- mean((test_y_pred - test_y)^2)

test_error_lm
```

```
## [1] 0.053985
```

```
## Forward


regfit_forward <- regsubsets(V86 ~ ., data = train_data, nvmax = 86,
    method = "forward")

reg_summary = summary(regfit_forward)

par(mfrow = c(1, 2))

plot(reg_summary$cp, xlab = "Number of Variables", ylab = "Cp",
    type = "l")

plot(reg_summary$bic, xlab = "Number of Variables", ylab = "BIC",
    type = "l")
```




```
which(reg_summary$cp == min(reg_summary$cp))
```

```
## [1] 23
```

```
which(reg_summary$bic == min(reg_summary$bic))
```

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

```
coef_forward_bic <- coef(regfit_forward, 8)

coef_forward_bic
```

```
##  (Intercept)          V10          V18          V43          V44
## -0.022141589  0.005775927 -0.006291720  0.004805098  0.012538290
##          V47          V59          V82          V85
##  0.010545078  0.005886931  0.284878668  0.080443574
```

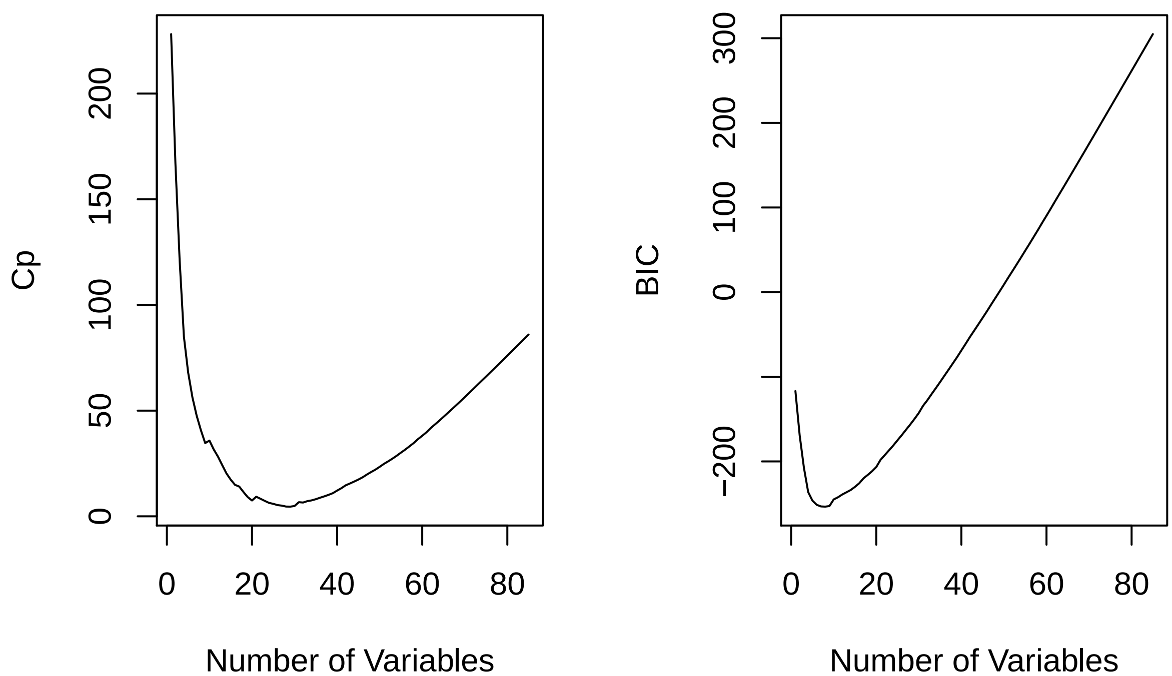
```
forward_lm <- lm(V86 ~ V10 + V18 + V43 + V44 + V47 + V59 + V82 +
    V85, data = train_data)


train_y_pred_forward <- predict(forward_lm, newdata = train_x)

forward_train_error <- mean((train_y_pred_forward = train_y)^2)

forward_train_error
```

```
## [1] 0.05977327
```

```
test_y_pred_forward <- predict(forward_lm, newdata = test_x)

forward_test_error <- mean((test_y_pred_forward - test_y)^2)

forward_test_error
```

```
## [1] 0.05421567
```

```r
## Backward

regfit_backward <- regsubsets(V86 ~ ., data = train_data, nvmax = 86,
    method = "backward")

reg_summary_back = summary(regfit_backward)

par(mfrow = c(1, 2))

plot(reg_summary_back$cp, xlab = "Number of Variables", ylab = "Cp",
    type = "l")

plot(reg_summary_back$bic, xlab = "Number of Variables", ylab = "BIC",
    type = "l")
```




```r
which(reg_summary_back$cp == min(reg_summary_back$cp))
```

```
## [1] 29
```

```r
which(reg_summary_back$bic == min(reg_summary_back$bic))
```

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

```r
coef_backward_bic <- coef(regfit_backward, 8)
coef_backward_bic
```

```
##  (Intercept)          V10          V18          V21          V46
##  0.001850234  0.006879012 -0.007523787 -0.008752079 -0.019827878
##          V47          V59          V82          V85
##  0.011057523  0.010985109  0.283583028  0.080852868
```

```r
backward_lm <- lm(V86 ~ V10 + V18 + V21 + V46 + V47 + V59 + V82 +
    V85, data = train_data)
```

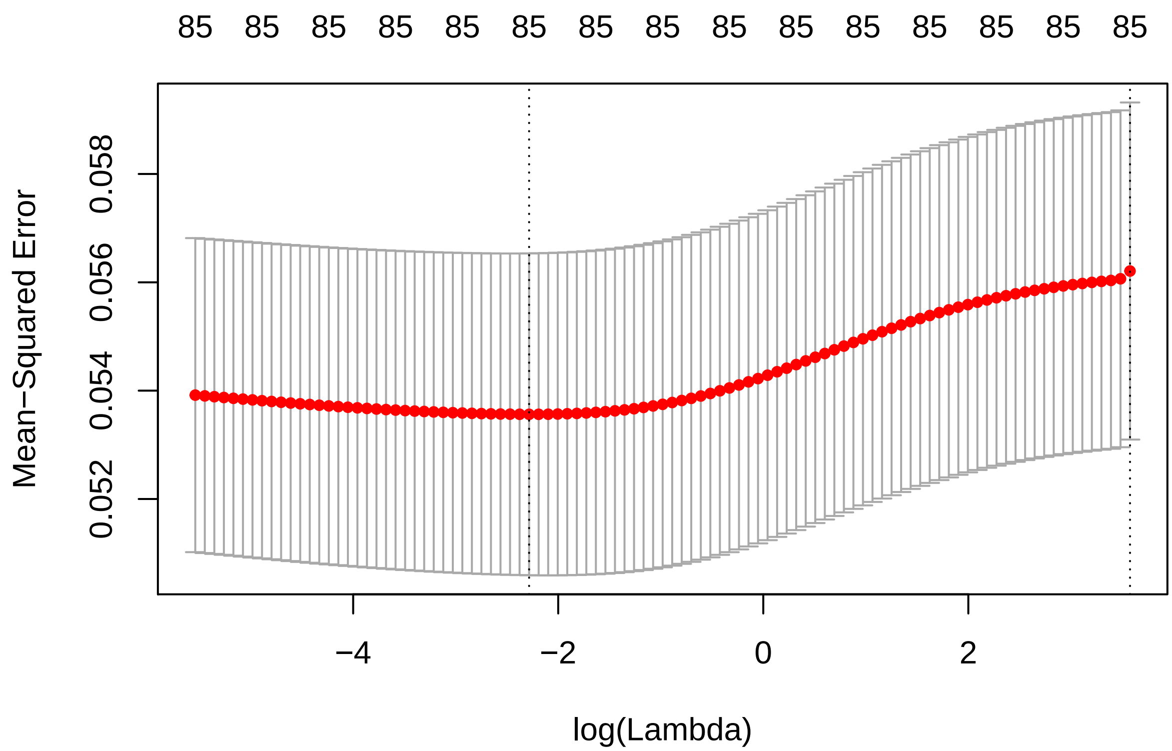
```r
train_y_pred_backward <- predict(backward_lm, newdata = train_x)

backward_train_error <- mean((train_y_pred_backward - train_y)^2)

backward_train_error
```

```
## [1] 0.05309043
```

```r
test_y_pred_backward <- predict(backward_lm, newdata = test_x)

backward_test_error <- mean((test_y_pred_backward - test_y)^2)

backward_test_error
```

```
## [1] 0.05411259
```

```r
# Ridge

train_ridge_x <- as.matrix(train_data[, c(1:85)])
train_ridge_y <- train_data[, c(86)]

test_ridge_x <- as.matrix(test_x)
test_ridge_y <- test_y

cv.ridge <- cv.glmnet(train_ridge_x, train_ridge_y, alpha = 0)
plot(cv.ridge)
```




```r
ridge_bestlam <- cv.ridge$lambda.min

ridge_bestlam
```

```
## [1] 0.1018902
```

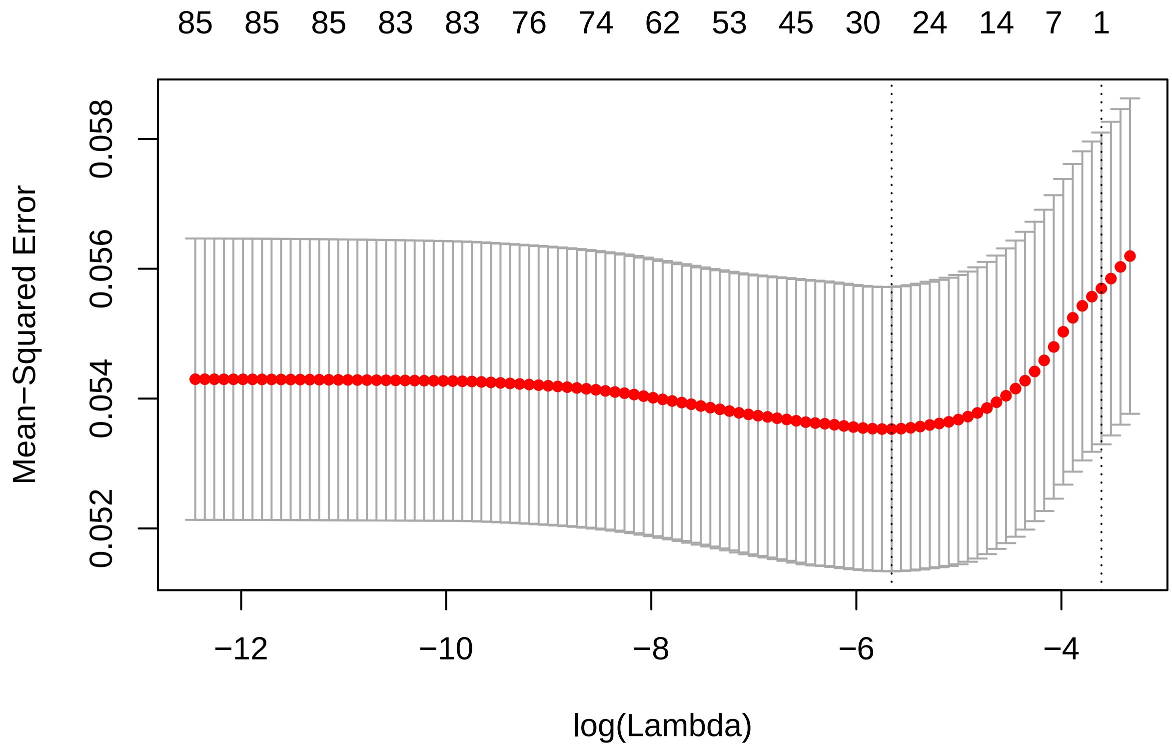
```r
ridge.mod = glmnet(train_ridge_x, train_ridge_y, alpha = 0)

ridge.pred <- predict(ridge.mod, s = ridge_bestlam, type = "coefficients")

ridge.pred2 <- predict(ridge.mod, s = ridge_bestlam, newx = train_ridge_x,
    type = "response")


ridge_train_error <- mean((ridge.pred2 - train_ridge_y)^2)

ridge_train_error
```

```
## [1] 0.05263956
```

```r
ridge.pred3 <- predict(ridge.mod, s = ridge_bestlam, newx = test_ridge_x,
    type = "response")

ridge_test_error <- mean((ridge.pred3 - test_ridge_y)^2)

ridge_test_error
```

```
## [1] 0.05369624
```

```r
# lasso

train_lasso_x <- as.matrix(train_data[, c(1:85)])
train_lasso_y <- train_data[, c(86)]

test_lasso_x <- as.matrix(test_x)
test_lasso_y <- test_y

cv.lasso = cv.glmnet(train_lasso_x, train_lasso_y, alpha = 1)
plot(cv.lasso)
```




```
lasso_bestlam <- cv.lasso$lambda.min
lasso_bestlam
```

```
## [1] 0.003495312
```

```
lasso.mod = glmnet(train_lasso_x, train_lasso_y, alpha = 1)

lasso.pred <- predict(lasso.mod, s = lasso_bestlam, type = "coefficients")

lasso.pred2 <- predict(lasso.mod, s = lasso_bestlam, newx = train_lasso_x,
    type = "response")

lasso_train_error <- mean((lasso.pred2 - train_lasso_y)^2)

lasso_train_error
```

```
## [1] 0.05278685
```

```
lasso.pred3 <- predict(lasso.mod, s = lasso_bestlam, newx = test_lasso_x,
    type = "response")

lasso_test_error <- mean((lasso.pred3 - test_lasso_y)^2)

lasso_test_error
```

```
## [1] 0.0537716
```

Question3

(10 points) (Exercise 9 modified, ISL) We have seen that as the number of features used in a model increases, the training error will necessarily decrease, but the test error may not. We will now explore this in a simulated

data set. Generate a data set with p = 20 features, n = 1, 000 observations, and an associated quantitative response vector generated according to the model Y=XB+ewhere B has some elements that are exactly equal to zero. Split your data set into a training set containing 100 observations and a test set containing 900 observations.Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size. Plot the test set MSE associated with the best model of each size.

```
set.seed(12345)

x <- matrix(rnorm(1000 * 20), nrow = 1000, ncol = 20)

b <- rnorm(20)

b[c(2, 4, 5, 7, 10, 15)] <- 0

random_error <- rnorm(1000)

y <- x %*% b + random_error

train <- sample(seq(1000), 100, replace = FALSE)

test <- (-train)

x_train <- x[train, ]

x_test <- x[test, ]

y_train <- y[train]

y_test <- y[test]

data_train <- data.frame(y = y_train, x = x_train)

reg_fit <- regsubsets(y ~ ., data = data_train, nvmax = 20)

train_mat <- model.matrix(y ~ ., data = data_train, nvmax = 20)

validation_error <- rep(NA, 20)
for (i in 1:20) {
    coefi = coef(reg_fit, id = i)
    pred <- train_mat[, names(coefi)] %*% coefi
    validation_error[i] <- mean((pred - y_train)^2)
}
# I applied this function from ISR lab chaper6.

data_test <- data.frame(y = y_test, x = x_test)

test_mat <- model.matrix(y ~ ., data = data_test, nvmax = 20)

validation_error_test <- rep(NA, 20)
for (i in 1:20) {
    coefi = coef(reg_fit, id = i)
    pred <- test_mat[, names(coefi)] %*% coefi
    validation_error_test[i] <- mean((pred - y_test)^2)
}
```

```r
# I applied this function from ISR lab chaper6.

par(mfrow = c(1, 2))

plot(validation_error, xlab = "Number of predictors", ylab = "Training MSE",
     col = "blue", type = "l")
plot(validation_error_test, xlab = "Number of predictors", ylab = "Test MSE",
     col = "red", type = "l")
```