HW-10-PCA_Wes_Wilson

Packages

```
library(tidyverse)
library(ISLR2)
library(pls)
library(MASS)
library(leaps)
library(glmnet)
```

Data and Functions

```
data("Boston")
data("College")
split_train_and_test_data <- function(split_pct, data, seed){
  set.seed(seed)
  z <- sample(nrow(data), split_pct * nrow(data))
  train <- data[z,]
  test <- data[-z,]

  return(list(train = train, test = test))
}

calc_mse <- function(y_actual, y_predicted) {
  if (!is.vector(y_actual) || !is.vector(y_predicted)) {
    warning("Both y_actual and y_predicted should be vectors.")
    return(NULL)  # Return NULL to indicate an issue
  }
  mse <- mean((y_actual - y_predicted)^2)

  return(mse)
}
```

College Applications Continued

This continues the College data problem from the previous h/w. (Chap. 6, # 9 cd, p.286-287)

Predict the number of applications received based on the other variables in the College data set.

- This data set is from our textbook. Access it with `library(ISLR2)`.

Now fit models using two additional regularization methods using the validation set based on 50% split where appropriate and `set.seed(123)` where appropriate:

```r
model_data <- split_train_and_test_data(split_pct = .5,
                                        seed = 123,
                                        data = College)
```

(a) PCR model, with M, the number of principal components, chosen by cross-validation

```r
pcr_reg <- pcr(Apps ~ ., data = model_data$train, validation = "CV")
summary(pcr_reg)
```

```
Data:   X dimension: 388 17
    Y dimension: 388 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            3258     3258     1482     1463     1342     1235     1076
adjCV         3258     3258     1481     1462     1339     1234     1073
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV        1075     1082     1083      1065      1050      1053      1047
adjCV     1073     1080     1080      1062      1047      1049      1044
       14 comps  15 comps  16 comps  17 comps
CV         1042      1039      1042      1023
adjCV      1038      1035      1038      1018
```

```
TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X      53.738    87.86    95.79    97.67    98.77    99.40    99.91    99.96
Apps    1.547    79.79    80.49    83.71    86.45    89.88    89.91    89.91
      9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
X      100.00    100.00    100.00    100.00    100.00    100.00    100.00
Apps    89.92     90.36     90.65     90.69     90.85     91.05     91.19
      16 comps  17 comps
X        100.0    100.00
Apps      91.2     91.58
```

(b) PLS model, with M, the number of principal components, chosen by cross-validation.

```
pls_reg <- plsr(Apps ~ ., data = model_data$train, validation = "CV")
summary(pls_reg)
```

```
Data:   X dimension: 388 17
    Y dimension: 388 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            3258     1472     1430     1256     1121     1070     1066
adjCV         3258     1466     1446     1256     1120     1068     1065
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV        1072     1075     1056      1041      1034      1030      1028
adjCV     1070     1073     1059      1038      1031      1026      1025
       14 comps  15 comps  16 comps  17 comps
CV         1029      1029      1026      1014
adjCV      1025      1026      1023      1010

TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X       35.02    86.48    93.11    97.29    98.47    99.40    99.77    99.96
Apps    80.22    81.34    85.77    88.84    89.82    89.91    89.93    89.95
      9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
X       99.99    100.00       100    100.00    100.00     100.0     100.0
Apps    90.24     90.75        91     91.16     91.19      91.2      91.2
      16 comps  17 comps
X       100.00    100.00
```

```
Apps     91.22     91.58
```

(c) Evaluate performance of each method in terms of **prediction MSE** and add the results into the summary table.

(d) Comment on the results obtained.

With the PCR model, around 6 components appears to be the optimal number. Once we move beyond 10 or so components, the changes in the variance explained by the x values and the rmse is nominal.

With the plsr model, around 9 components looks to be an optimal number; though I suppose that is arguable depending on who you ask. I am going create new models that have the optimal number of components and use that to predict and evaluate performance.

```
pcr_predictions <- predict(object = pcr_reg,
                           newdata = model_data$train, ncomp = 6) %>%
                           as.vector()

plsr_redictions <- predict(object = pls_reg, newdata = model_data$train,
                           ncomp = 9) %>% as.vector()

# the pls functions already does this; but doing again to better understand.
calc_mse(y_actual = model_data$train$Apps,
         y_predicted = pcr_predictions)
```

```
[1] 1068315
```

```
calc_mse(y_actual = model_data$train$Apps,
         y_predicted = plsr_redictions)
```

```
[1] 1030691
```

Do on the test data now.

```
pcr_predictions <- pcr_predictions <- predict(object = pcr_reg,
                           newdata = model_data$test, ncomp = 6) %>%
                           as.vector()

plsr_redictions <- predict(object = pls_reg, newdata = model_data$test,
                           ncomp = 9) %>% as.vector()
```

```
pcr_rmse <- calc_mse(y_actual = model_data$test$Apps,
         y_predicted = pcr_predictions) %>% sqrt()

plsr_rmse <- calc_mse(y_actual = model_data$test$Apps,
         y_predicted = plsr_redictions) %>% sqrt()
```

- How accurately can we predict the number of college applications?

  ```
  tibble(method = c("step_lse", "step_k_fold", "ridge", "lasso_1se", "lasso_min", "pcr"
         results = c(sqrt(1327026), sqrt(1264962), sqrt(2092402),
         sqrt(1528732), sqrt(1391186), pcr_rmse, plsr_rmse))
  ```

  ```
  # A tibble: 7 x 2
    method      results
    <chr>         <dbl>
  1 step_lse      1152.
  2 step_k_fold   1125.
  3 ridge         1447.
  4 lasso_1se     1236.
  5 lasso_min     1179.
  6 pcr           1286.
  7 pls           1258.
  ```

  - Is there much difference among the test errors resulting from these five approaches?
  - Which method appears most accurate?

There does not appear to be a drastic difference in the model accuracy between our approaches. We are able to predict the number of applications with rmse of between 1,150 and 1,300. It looks like the stepwise approach was the most accurate with 12 predictors.

- Look at help for: `pls::validationplot()`, `pls::R2()`, `pls::MSEP()`

```
pls::RMSEP(object = pcr_reg, estimate = "all",
           newdata = model_data$test)
```

```
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
train         3249     3224     1461     1435     1311     1196     1034
CV            3258     3258     1482     1463     1342     1235     1076
adjCV         3258     3258     1481     1462     1339     1234     1073
test          4401     4404     2155     2096     1973     1742     1286
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
```

```
train     1032     1032     1032      1009     993.4     991.5     982.7
CV        1075     1082     1083      1065    1049.9    1052.6    1047.2
adjCV     1073     1080     1080      1062    1046.8    1049.4    1043.8
test      1276     1270     1270      1221    1193.3    1192.3    1205.9
       14 comps  15 comps  16 comps  17 comps
train     971.9     964.4     963.9       943
CV       1041.8    1038.5    1042.4      1023
adjCV    1037.7    1034.5    1038.1      1018
test     1181.6    1163.5    1161.0      1172
```

```
# this appears to do the same thing that I did earlier manually, but faster. I can pass ne
```

Comparison of Dimension Reduction Methods

We will now try to predict per capita crime rate in the Boston data set in the {MASS} package (`library(MASS)`).

- Leave `rad` as an integer.
- Use `set.seed(1234)` and 50% validation where appropriate.

```
model_data <- split_train_and_test_data(split_pct = .5,
                                        seed = 1234, data = Boston)
```

a. Try out at least four of the dimension reduction methods that we explored over the last two weeks, such as the best subset selection, lasso, ridge regression, PCR, and PLS. Discuss results.

Principal Component Analysis Model

```
pcr_model <- pcr(crim ~ ., data = model_data$train, validation = "CV")
summary(pcr_model)
```

```
Data:   X dimension: 253 13
    Y dimension: 253 1
Fit method: svdpc
Number of components considered: 13

VALIDATION: RMSEP
```

```
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           10.32    8.701    8.735    8.734    8.722    8.660    8.345
adjCV        10.32    8.696    8.726    8.724    8.712    8.647    8.331
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       8.304    8.247    8.233     8.203     8.246     8.248     8.227
adjCV    8.289    8.231    8.217     8.185     8.224     8.225     8.203

TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X       82.53    96.85    98.99    99.68    99.86    99.92    99.96    99.99
crim    29.95    30.74    30.81    31.12    32.91    38.24    39.15    40.17
      9 comps  10 comps  11 comps  12 comps  13 comps
X      100.00    100.00    100.00     100.0     100.0
crim    40.55     41.36     42.07      42.2      42.7
```

It appears that 7 components is the optimal number. After that, the increases are extremely marginal. Still, the model is only accounting for ~42% of the variance and the RMSE is pretty high.

RMSE = ~ 8.7

Partial Least Squares Model

```
plsr_model <- plsr(crim ~ ., data = model_data$train, validation = "CV")
summary(plsr_model)
```

```
Data:   X dimension: 253 13
    Y dimension: 253 1
Fit method: kernelpls
Number of components considered: 13

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           10.32    8.749    8.763    8.675    8.529    8.362    8.234
adjCV        10.32    8.741    8.752    8.662    8.515    8.348    8.220
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       8.226    8.236    8.217     8.237     8.303     8.318     8.300
adjCV    8.210    8.219    8.200     8.218     8.278     8.291     8.273
```

```
TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X       82.48    96.79    98.18    99.36    99.81    99.92    99.95    99.97
crim    30.22    30.92    33.16    35.68    38.24    40.24    40.79    41.18
      9 comps  10 comps  11 comps  12 comps  13 comps
X       99.99    100.00    100.00    100.00     100.0
crim    41.49     41.71     42.18     42.29      42.7
```

The pls model performs quite similarly and its pretty hard to tell what the optimal number of components should be. After one component, the changes in variance and RMSE is pretty small.

RMSE = ~8.7

Stepwise Regression

```
y_train <- model_data$train$crim
full_regression <- lm(crim ~ ., data = model_data$train)
step_output <- step(object = full_regression, method = "backward")
```

```
Start:  AIC=1066.32
crim ~ zn + indus + chas + nox + rm + age + dis + rad + tax +
    ptratio + black + lstat + medv

          Df Sum of Sq   RSS    AIC
- age      1      0.00 15328 1064.3
- chas     1     22.82 15350 1064.7
- indus    1     23.96 15352 1064.7
- tax      1     29.55 15357 1064.8
- black    1     62.15 15390 1065.3
- ptratio  1     85.09 15413 1065.7
<none>                 15328 1066.3
- nox      1    135.92 15464 1066.5
- rm       1    164.66 15492 1067.0
- zn       1    170.49 15498 1067.1
- lstat    1    223.60 15551 1068.0
- dis      1    331.43 15659 1069.7
- medv     1    622.51 15950 1074.4
- rad      1   1320.70 16648 1085.2

Step:  AIC=1064.32
```

```
crim ~ zn + indus + chas + nox + rm + dis + rad + tax + ptratio +
    black + lstat + medv

          Df Sum of Sq   RSS    AIC
- chas     1     22.86 15350 1062.7
- indus    1     23.96 15352 1062.7
- tax      1     29.55 15357 1062.8
- black    1     62.21 15390 1063.3
- ptratio  1     85.37 15413 1063.7
<none>                 15328 1064.3
- nox      1    146.70 15474 1064.7
- rm       1    174.32 15502 1065.2
- zn       1    177.64 15505 1065.2
- lstat    1    240.94 15568 1066.3
- dis      1    357.01 15685 1068.1
- medv     1    625.62 15953 1072.4
- rad      1   1328.81 16656 1083.3

Step:  AIC=1062.69
crim ~ zn + indus + nox + rm + dis + rad + tax + ptratio + black +
    lstat + medv

          Df Sum of Sq   RSS    AIC
- tax      1     25.43 15376 1061.1
- indus    1     27.71 15378 1061.2
- black    1     65.67 15416 1061.8
- ptratio  1     84.66 15435 1062.1
<none>                 15350 1062.7
- nox      1    164.37 15515 1063.4
- zn       1    177.55 15528 1063.6
- rm       1    178.03 15528 1063.6
- lstat    1    241.90 15592 1064.7
- dis      1    362.41 15713 1066.6
- medv     1    663.50 16014 1071.4
- rad      1   1312.14 16662 1081.4

Step:  AIC=1061.11
crim ~ zn + indus + nox + rm + dis + rad + ptratio + black +
    lstat + medv

          Df Sum of Sq   RSS    AIC
- black    1     64.59 15440 1060.2
- indus    1     91.95 15468 1060.6
- ptratio  1     93.80 15470 1060.7
<none>                 15376 1061.1
- zn       1    156.90 15533 1061.7
- nox      1    169.01 15545 1061.9
- rm       1    171.53 15547 1061.9
- lstat    1    244.71 15620 1063.1
- dis      1    363.02 15739 1065.0
- medv     1    639.24 16015 1069.4
- rad      1   2698.70 18074 1100.0

Step:  AIC=1060.17
crim ~ zn + indus + nox + rm + dis + rad + ptratio + lstat +
    medv

          Df Sum of Sq   RSS    AIC
- indus    1     87.57 15528 1059.6
- ptratio  1    110.00 15550 1060.0
<none>                 15440 1060.2
- zn       1    162.05 15602 1060.8
- nox      1    179.36 15620 1061.1
- rm       1    219.95 15660 1061.8
- lstat    1    282.44 15723 1062.8
- dis      1    379.04 15820 1064.3
- medv     1    741.43 16182 1070.0
- rad      1   2984.79 18425 1102.9

Step:  AIC=1059.6
crim ~ zn + nox + rm + dis + rad + ptratio + lstat + medv

          Df Sum of Sq   RSS    AIC
<none>                 15528 1059.6
- ptratio  1    132.06 15660 1059.8
- zn       1    176.56 15705 1060.5
- rm       1    251.19 15779 1061.7
- lstat    1    273.81 15802 1062.0
- nox      1    274.82 15803 1062.0
- dis      1    308.96 15837 1062.6
- medv     1    695.31 16223 1068.7
- rad      1   2916.19 18444 1101.2
```

```
step_summary <- summary(step_output)

step_best_model <- lm(formula = crim ~ zn + nox + rm +
                          dis + rad + ptratio +
                          lstat + medv, data = model_data$train)


step_predictions <- predict(object = step_best_model, newdata = model_data$train)

sqrt(calc_mse(y_actual = y_train, y_predicted = step_predictions))
```

```
[1] 7.834251
```

Stepwise regression selection selected 7 variables from the original 12 and resulted with rmse of 7.83.

Ridge Regression

```
x_train <- model.matrix(lm(formula = crim ~ .,
                            data = model_data$train))[,-1]

r_train <- cv.glmnet(x = x_train, y = y_train, alpha = 0)
```

```
ridge_predictions <- predict(r_train, newx = x_train) %>% as.vector()
ridge_predictions
```

```
  [1] 1.612530 3.283893 5.885659 3.807628 5.172257 3.927413 5.889116 4.456629
  [9] 5.632322 5.989434 6.057029 2.824525 4.544568 3.335639 3.103409 3.144530
 [17] 3.692734 5.331243 4.800348 3.385360 5.617881 3.174858 3.730548 2.699924
 [25] 4.010553 5.075539 2.573706 4.032486 5.753436 2.423660 3.429051 4.158854
 [33] 6.069259 2.625611 3.561721 6.143913 5.654299 5.469102 2.753608 4.151851
 [41] 4.543778 3.312781 2.392795 4.080233 3.428088 3.501909 6.448969 3.504647
 [49] 2.626781 2.826792 3.998475 5.539902 3.121006 2.862221 5.223336 4.928210
 [57] 3.051871 5.728671 5.458974 3.536729 3.493848 4.034943 3.916215 4.669889
 [65] 2.569136 5.181098 4.284687 3.397671 2.940994 3.161263 3.972334 2.660683
 [73] 3.135902 3.808443 5.172708 2.478554 2.470502 3.634692 3.725109 3.406174
 [81] 4.822377 4.662451 5.857774 3.022307 3.563799 5.873913 3.601747 3.941436
 [89] 5.717717 2.811296 4.250093 3.124699 3.517978 2.687095 6.255103 5.794497
 [97] 4.661772 5.583352 5.222162 3.335123 3.675303 6.200523 3.925205 4.731010
```

```
[105] 4.173565 3.684115 3.660640 4.307700 5.754074 3.737641 4.875767 3.720710
[113] 5.535622 3.187702 4.251081 4.390558 3.860719 3.492540 5.702490 3.083529
[121] 4.814622 2.952592 3.483246 3.872219 2.761304 3.268598 5.165511 2.738989
[129] 5.569939 4.644664 3.256783 5.739562 3.204553 6.102026 5.185986 3.921102
[137] 6.382558 3.524596 3.140672 3.311948 5.906258 5.415257 6.005334 3.236285
[145] 3.113515 5.289821 5.127521 6.737446 2.946142 2.879296 4.256979 4.756471
[153] 6.294249 4.227496 3.868277 2.988178 3.484438 5.770035 5.274886 2.875234
[161] 1.960236 3.150972 5.415163 2.926357 4.849388 2.467219 5.475836 5.037559
[169] 3.841829 6.034724 5.470242 6.039459 3.611647 3.144789 5.133556 3.540403
[177] 2.704194 5.344277 3.003421 5.455231 4.288729 2.710407 5.364268 4.305857
[185] 3.012002 4.361344 2.930048 2.885458 4.669527 2.895473 3.405247 4.448731
[193] 3.414483 5.575409 3.432226 3.397185 5.961657 2.255470 6.081084 3.062482
[201] 3.775953 4.205282 3.607564 3.255160 5.554167 3.115011 5.847252 3.205094
[209] 3.087763 5.146055 2.501293 2.083660 3.182924 3.149880 2.568591 3.522708
[217] 2.410190 3.457947 2.017172 2.265563 3.633992 3.547817 1.954263 4.116003
[225] 4.037627 4.104865 5.003165 5.566102 3.277276 5.152197 3.324057 5.541940
[233] 3.673378 6.057449 3.874799 2.350740 3.632658 3.390823 2.656352 5.252871
[241] 4.129647 2.326829 3.166769 3.328426 5.568014 5.473942 3.477075 5.519424
[249] 4.106352 3.355919 4.239522 2.832617 3.198019
```

```r
sqrt(calc_mse(y_actual = y_train, y_predicted = ridge_predictions))
```

```
[1] 9.683361
```

The ridge regression doesn't remove any variables and the rmse is 9.46.

Overall, there was not a huge difference in performance. The stepwise selected model performed the best, with an rmse of ~ 7.9. The PCR and PLSR model perform similarly and the ridge regression did the worst.

Create Plots

a. For all possible values of K, compare

- regression models that are based on the best subset of -variables.

```r
n_predictors <- ncol(model_data$train) - 1

# empty data frame to store the results.
results <- data.frame(matrix(NA, ncol = 3, nrow = n_predictors))
colnames(results) <- c("Variables", "adjusted_r2", "rmse")
```

```r
for(k in 1:n_predictors) {
  #browser()
  predictor_combinations <- regsubsets(crim ~ .,
                                       data = model_data$train,
                                       nvmax = k, intercept = FALSE)

  # Fit models for each combination and obtain adjusted R-squared and MSE
  models <- summary(predictor_combinations)
  adj_r_squared_values <- models$rsq

  # Find the index of the model with the highest adjusted R-squared
  best_model_index <- which.max(adj_r_squared_values)

  # Extract the names of selected predictors
  selected_predictors <- names(coef(predictor_combinations, id = best_model_index))

  # Fit a linear model using the selected predictors
  lm_model <- lm(crim ~ .,
                 data = model_data$train[,c("crim", selected_predictors)])

  # Calculate MSE
  y_predicted <- predict(lm_model, newdata = model_data$train)
  rmse_value <- sqrt(mean((model_data$train$crim - y_predicted)^2))


  # Store results for the current k
results[k, 1] <- length(names(coef(predictor_combinations, id = best_model_index)))
results[k, 2] <- adj_r_squared_values[best_model_index]
results[k, 3] <- rmse_value
}

step_results <- results
step_results
```

```
  Variables adjusted_r2     rmse
1         1   0.4090749 8.318242
2         2   0.4545558 8.121705
3         3   0.4709485 8.044823
4         4   0.4744398 8.018398
5         5   0.4811558 7.956229
```

```
6           6   0.4870767 7.921192
7           7   0.4938450 7.867494
8           8   0.4977103 7.838929
9           9   0.4999329 7.819515
10         10   0.5010587 7.795773
11         11   0.5018515 7.789602
12         12   0.5025528 7.783521
13         13   0.5025659 7.783521
```

- PCR based on first principal components;
- The `pls::R2()` and `pls::MSEP()` functions calculate this automatically, so I will just extract those values as a vector.

```r
pcr_model <- pcr(crim ~ . -1, data = model_data$train,
                 validation = "CV", scaled = TRUE)

pcr_r_squared <- pls::R2(pcr_model, estimate = "test",
                         newdata = model_data$test)$val %>%
data.frame() %>%
  pivot_longer(cols = everything(),
               names_to = "component",
               values_to = "r_squared") %>%
  slice(-1)

pcr_rmse <- pls::RMSEP(pcr_model, estimate = "test",
                       newdata = model_data$test)$val %>%
data.frame() %>%
  pivot_longer(cols = everything(),
               names_to = "component",
               values_to = "rmse") %>%
  slice(-1) %>%
  dplyr::select(rmse)

pcr_results <- bind_cols(pcr_r_squared, pcr_rmse)
```

- PLS based on first PLS components.

```r
plsr_model <- plsr(crim ~ ., data = model_data$train,
                   validation = "CV", scaled = TRUE)

plsr_r_squared <- pls::R2(plsr_model, estimate = "test",
                          newdata = model_data$test)$val %>%
data.frame() %>%
  pivot_longer(cols = everything(),
               names_to = "component",
               values_to = "r_squared") %>%
  slice(-1)

plsr_rmse <- pls::RMSEP(plsr_model, estimate = "test",
                        newdata = model_data$test)$val %>%
data.frame() %>%
  pivot_longer(cols = everything(),
               names_to = "component",
               values_to = "rmse") %>%
  slice(-1) %>%
  dplyr::select(rmse)

plsr_results <- bind_cols(plsr_r_squared, plsr_rmse)
```

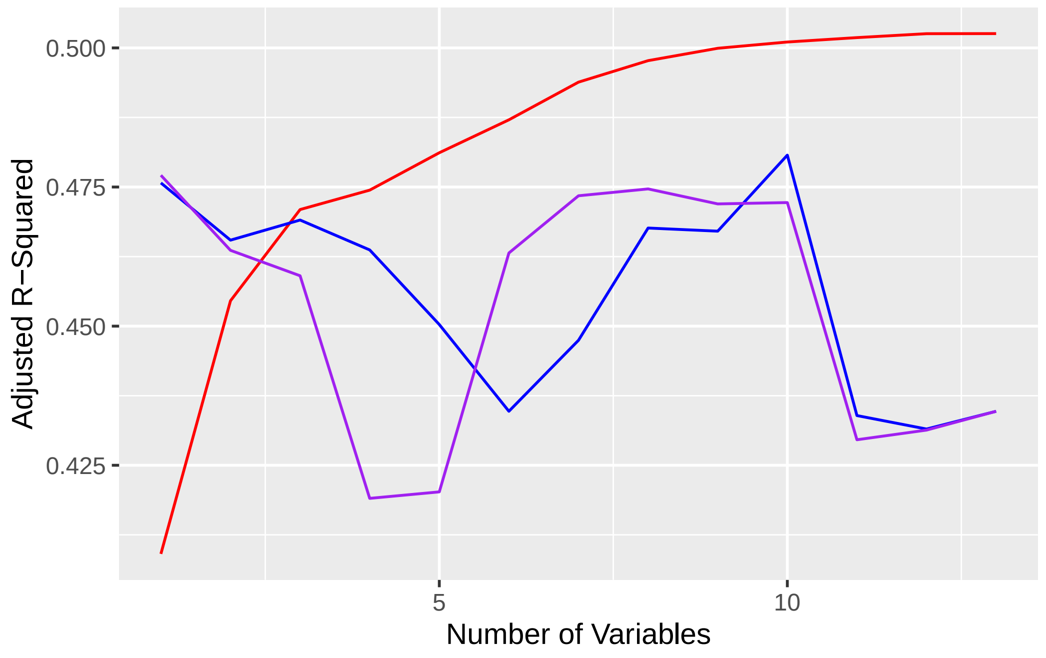
For each , compare these methods in terms of the **explained proportion of the total variation of crime rate per capita** (adjusted) and in terms of **the prediction mean-squared error**.

- Hint: Look at the regsubsets output object list elements. The `$adjR2` will be useful. You may find it helpful to use the `$which` matrix in a for-loop to get the cross-validated MSEP.

c. Create one plot comparing adjusted- 2 for different values of   for subset regression, PCR, and PLS and interpret the plot.

```
r_squared_data <- cbind(vars = step_results$Variables,
                        step_r2 = step_results$adjusted_r2,
                        pcr_r2 = pcr_results$r_squared,
                        plsr_r2 = plsr_results$r_squared) %>%
  as.data.frame()


ggplot(data = r_squared_data, aes(x = vars, y = step_r2)) +
  geom_line(color = "red") +
  geom_line(mapping = aes(y = pcr_r2), color = "blue") +
  geom_line(mapping = aes(y = plsr_r2), color = "purple") +
  labs(x = "Number of Variables", y = "Adjusted R-Squared")
```

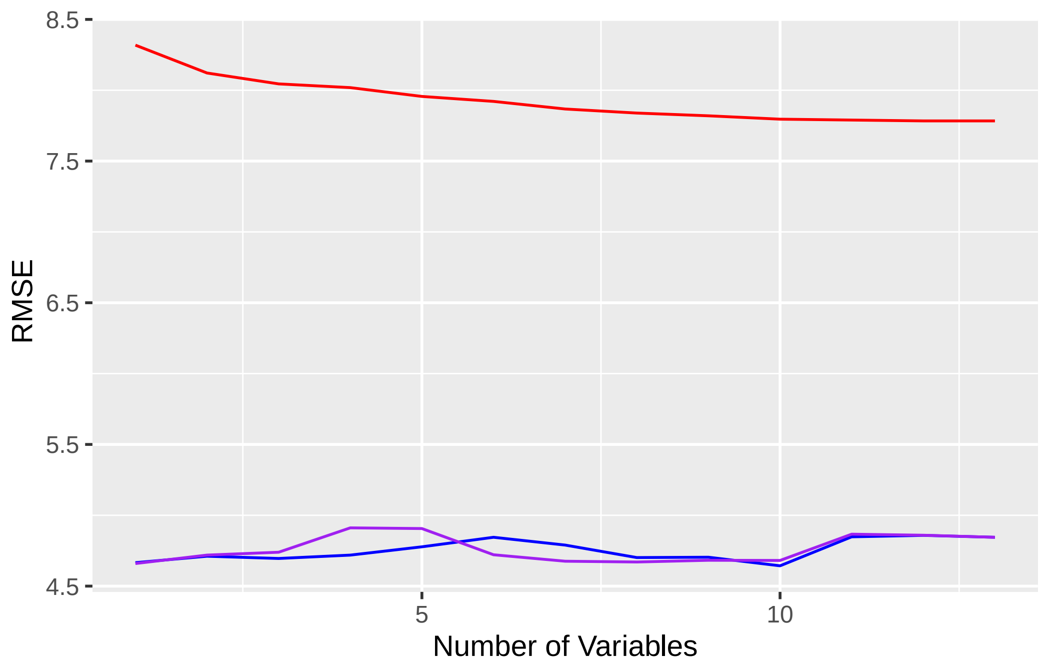


Based on the r-squared values, the stepwise model selection performed best with a max adjusted r-squared value of roughly 50% when it used all of the variables. Interestingly, the partial least squared model was outperforming the PCR model until the number of dimensions exceeded 8 or so. I am guessing this is an issue with scaling the variables.

c. Create one plot comparing Prediction MSE for different values of for subset regression, PCR, and PLS and interpret the plot.

```r
rmse_data <- cbind(vars = step_results$Variables,
                        step_rmse = step_results$rmse,
                        pcr_rmse = pcr_results$rmse,
                        plsr_rmse = plsr_results$rmse) %>%
  as.data.frame()


ggplot(data = rmse_data, aes(x = vars, y = step_rmse)) +
  geom_line(color = "red") +
  geom_line(mapping = aes(y = pcr_rmse), color = "blue") +
  geom_line(mapping = aes(y = plsr_rmse), color = "purple") +
  labs(x = "Number of Variables", y = " RMSE")
```



e. Propose a model (or set of models) that seem to perform well on this data set, and justify your answer. Make sure you are evaluating model performance on basis of cross-validation, as opposed to using training error.

f. Does your chosen model involve all of the features in the data set? Why or why not?

I am not entirely sure that I set this up correctly. I don't fully understand estimate argument in the PCR/PLS function. I have changed it several times and get widely different results. It decided to make the estimate "test" which given that the model was created with the training makes sense to me.

I would select the PCR models with somewhere between 2-3 components. As we can see from the plot, the RMSE is pretty stable for with 2 components vs 10; therefore, its not worth the degrees-of-freedom price we pay to get an extra .001 in precision in the rmse.