Lab 05

Connor Brown

11:59PM March 16, 2019

Load the Boston housing data frame and create the vector y (the median value) and matrix X (all other features) from the data frame. Name the columns the same as Boston except for the first, name it "(Intercept)".

```r
data(Boston, package = "MASS")
summary(Boston)
```

```
##       crim                zn             indus            chas        
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000  
##  1st Qu.: 0.08204   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000  
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000  
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917  
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000  
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000  
##       nox               rm             age              dis        
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130  
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100  
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207  
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795  
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188  
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127  
##       rad              tax           ptratio          black       
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32  
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38  
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44  
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67  
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23  
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90  
##      lstat            medv      
##  Min.   : 1.73   Min.   : 5.00  
##  1st Qu.: 6.95   1st Qu.:17.02  
##  Median :11.36   Median :21.20  
##  Mean   :12.65   Mean   :22.53  
##  3rd Qu.:16.95   3rd Qu.:25.00  
##  Max.   :37.97   Max.   :50.00  
```

```r
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
```

```
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
y = Boston$medv
X = as.matrix(cbind(1, Boston[ , 1:13]))
colnames(X)[1] = "(Intercept)"
```

Run the OLS linear model to get b, the vector of coefficients. Do not use `lm`.

```
b = solve(t(X) %*% X) %*% t(X) %*% y
```

Find the hat matrix for this regression `H` and find its rank. Is this rank expected?

```
H = X %*% solve(t(X) %*% X) %*% t(X)
dim(H)
```

```
## [1] 506 506
```

```
pacman::p_load(Matrix)
?rankMatrix
rankMatrix(H)
```

```
## [1] 14
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 1.123546e-13
```

Verify this is a projection matrix by verifying the two sufficient conditions. Use the `testthat` library's `expect_equal(matrix1, matrix2, tolerance = 1e-2)`.

```
pacman::p_load(testthat)

expect_equal(H, t(H), tolerance = 1e-2)

expect_equal(H %*% H, H, tolerance = 1e-2)
```

Find the matrix that projects onto the space of residuals `H_comp` and find its rank. Is this rank expected?

```
I = diag(nrow(H))
H_comp = (I - H)
rankMatrix(H_comp)
```

```
## [1] 497
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 1.123546e-13
```

```
#The rank is expected to be n-(p+1) = 506-13 = 493
```

Verify this is a projection matrix by verifying the two sufficient conditions. Use the `testthat` library.

```r
expect_equal(H_comp, t(H_comp), tolerance = 1e-2)

expect_equal(H_comp %*% H_comp, H_comp, tolerance = 1e-2)
```

Calculate $\hat{y}$.

```r
y_hat = H %*% y
y_hat = as.vector(y_hat) #it's a matrix after multiplication
```

Calculate e as the difference of y and $\hat{y}$ and the projection onto the space of the residuals. Verify the two means of calculating the residuals provide the same results.

```r
e = y - y_hat
e_2 = as.vector(H_comp %*% y)
expect_equal(e, e_2)
```

Calculate R^2 and RMSE.

```r
sse = sum(e^2)
sst = sum((y - mean(y))^2)

R_sqrd = 1 - (sse/sst)
R_sqrd
```

```
## [1] 0.7406427
```

```r
MSE = sse / (nrow(X) - ncol(X))
RMSE = sqrt(MSE)        #RMSE is sd of errors
RMSE
```

```
## [1] 4.745298
```

Verify $\hat{y}$ and e are orthogonal.

```r
t(e) %*% y_hat
```

```
##                [,1]
## [1,] -4.991142e-08
```

Verify $\hat{y} - \bar{y}$ and e are orthogonal.

```r
t(e) %*% (y_hat - mean(y))
```

```
##               [,1]
## [1,] 2.832162e-09
```

Find the cosine-squared of $y - \bar{y}$ and $\hat{y} - \bar{y}$ and verify it is the same as R^2.

```r
y_minus_y_bar = y - mean(y)
yhat_minus_y_bar = y_hat - mean(y)

len_y_minus_y_bar = as.vector(sqrt( t(y_minus_y_bar) %*% y_minus_y_bar))
len_yhat_minus_y_bar = as.vector(sqrt( t(yhat_minus_y_bar) %*% yhat_minus_y_bar ))

theta = acos( len_yhat_minus_y_bar / len_y_minus_y_bar )
theta * (180 / pi)
```

```
## [1] 30.61531
```

```r
theta
```

```
## [1] 0.5343379
```

```r
cos_theta_sqrd = cos(theta)^2
cos_theta_sqrd
```

```
## [1] 0.7406427
```

```r
expect_equal(R_sqrd, cos_theta_sqrd)
```

Verify the sum of squares identity which we learned was due to the Pythagorean Theorem (applies since the projection is specifically orthogonal).

```r
RHS = len_yhat_minus_y_bar^2 + sse

expect_equal(len_y_minus_y_bar^2, RHS)
```

Create a matrix that is $(p+1) \times (p+1)$ full of NA's. Label the columns the same columns as X. Do not label the rows. For the first row, find the OLS estimate of the y regressed on the first column only and put that in the first entry. For the second row, find the OLS estimates of the y regressed on the first and second columns of X only and put them in the first and second entries. For the third row, find the OLS estimates of the y regressed on the first, second and third columns of X only and put them in the first, second and third entries, etc. For the last row, fill it with the full OLS estimates.

```r
#Create a matrix that is $(p + 1) \times (p + 1)$ full of NA's. Label the columns the same columns as X
M = matrix(NA, nrow = ncol(X), ncol = ncol(X))
colnames(M) = colnames(X)
M
```

```
##       (Intercept) crim zn indus chas nox rm age dis rad tax ptratio black
##  [1,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##  [2,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##  [3,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##  [4,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##  [5,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##  [6,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##  [7,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##  [8,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##  [9,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
## [10,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
## [11,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
## [12,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
## [13,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
## [14,]          NA   NA NA    NA   NA  NA NA  NA  NA  NA  NA      NA    NA
##       lstat
##  [1,]    NA
##  [2,]    NA
##  [3,]    NA
##  [4,]    NA
##  [5,]    NA
##  [6,]    NA
##  [7,]    NA
##  [8,]    NA
##  [9,]    NA
## [10,]    NA
## [11,]    NA
## [12,]    NA
## [13,]    NA
## [14,]    NA
```

```r
#For the first row, find the OLS estimate of the $y$ regressed on the first column only and put that in
X_j = X[ , 1, drop = FALSE]
b = solve(t(X_j) %*% X_j) %*% t(X_j) %*% y
b
```

```
##                 [,1]
## (Intercept) 22.53281
```

```r
M[1, 1] = b
#For the second row, find the OLS estimates of the $y$ regressed on the first and second columns of $X$
X_j_2 = X[ , 1:2]
b = solve(t(X_j_2) %*% X_j_2) %*% t(X_j_2) %*% y
b
```

```
##                   [,1]
## (Intercept) 24.0331062
## crim        -0.4151903
```

```r
M[2, 1:2] = b


#The entire thing in a for loop
for(j in 1 : ncol(M)){
  X_j = X[ , 1 : j, drop = FALSE]
  b = solve(t(X_j) %*% X_j) %*% t(X_j) %*% y
  M[j, 1 : j] = b
}
round(M, 2)
```

```
##       (Intercept)  crim   zn indus chas    nox   rm   age   dis   rad
##  [1,]       22.53    NA   NA    NA   NA     NA   NA    NA    NA    NA
##  [2,]       24.03 -0.42   NA    NA   NA     NA   NA    NA    NA    NA
##  [3,]       22.49 -0.35 0.12    NA   NA     NA   NA    NA    NA    NA
##  [4,]       27.39 -0.25 0.06 -0.42   NA     NA   NA    NA    NA    NA
##  [5,]       27.11 -0.23 0.06 -0.44 6.89     NA   NA    NA    NA    NA
##  [6,]       29.49 -0.22 0.06 -0.38 7.03  -5.42   NA    NA    NA    NA
##  [7,]      -17.95 -0.18 0.02 -0.14 4.78  -7.18 7.34    NA    NA    NA
##  [8,]      -18.26 -0.17 0.01 -0.13 4.84  -4.36 7.39 -0.02    NA    NA
##  [9,]        0.83 -0.20 0.06 -0.23 4.58 -14.45 6.75 -0.06 -1.76    NA
## [10,]        0.16 -0.18 0.06 -0.21 4.54 -13.34 6.79 -0.06 -1.75 -0.05
## [11,]        2.99 -0.18 0.07 -0.10 4.11 -12.59 6.66 -0.05 -1.73  0.16
## [12,]       27.15 -0.18 0.04 -0.04 3.49 -22.18 6.08 -0.05 -1.58  0.25
## [13,]       20.65 -0.16 0.04 -0.03 3.22 -20.48 6.12 -0.05 -1.55  0.28
## [14,]       36.46 -0.11 0.05  0.02 2.69 -17.77 3.81  0.00 -1.48  0.31
##         tax ptratio black lstat
##  [1,]    NA      NA    NA    NA
##  [2,]    NA      NA    NA    NA
##  [3,]    NA      NA    NA    NA
##  [4,]    NA      NA    NA    NA
##  [5,]    NA      NA    NA    NA
##  [6,]    NA      NA    NA    NA
##  [7,]    NA      NA    NA    NA
##  [8,]    NA      NA    NA    NA
##  [9,]    NA      NA    NA    NA
## [10,]    NA      NA    NA    NA
## [11,] -0.01      NA    NA    NA
```

```
## [12,] -0.01   -1.00    NA    NA
## [13,] -0.01   -1.01  0.01    NA
## [14,] -0.01   -0.95  0.01 -0.52
```

Examine this matrix. Why are the estimates changing from row to row as you add in more predictors?

Because each iteration, we include another variable in our regression, which will change the outcome of 'solve(t(X_j) %% *X_j) %%* t(X_j)'.

Clear the workspace and load the diamonds dataset.

```
rm(list=ls())
pacman::p_load(ggplot2)
data(diamonds, package = "ggplot2")
```

Extract y, the price variable and "c", the nominal variable "color" as vectors.

```
summary(diamonds)
```

```
##      carat               cut        color       clarity
##  Min.   :0.2000   Fair     : 1610   D: 6775   SI1    :13065
##  1st Qu.:0.4000   Good     : 4906   E: 9797   VS2    :12258
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194
##  Mean   :0.7979   Premium  :13791   G:11292   VS1    : 8171
##  3rd Qu.:1.0400   Ideal    :21551   H: 8304   VVS2   : 5066
##  Max.   :5.0100                     I: 5422   VVS1   : 3655
##                                     J: 2808   (Other): 2531
##      depth          table           price             x
##  Min.   :43.00   Min.   :43.00   Min.   :  326   Min.   : 0.000
##  1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710
##  Median :61.80   Median :57.00   Median : 2401   Median : 5.700
##  Mean   :61.75   Mean   :57.46   Mean   : 3933   Mean   : 5.731
##  3rd Qu.:62.50   3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540
##  Max.   :79.00   Max.   :95.00   Max.   :18823   Max.   :10.740
##
##        y                z
##  Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 4.720   1st Qu.: 2.910
##  Median : 5.710   Median : 3.530
##  Mean   : 5.735   Mean   : 3.539
##  3rd Qu.: 6.540   3rd Qu.: 4.040
##  Max.   :58.900   Max.   :31.800
##
```

```
y = diamonds$price
c = diamonds$color
table(c)
```

```
## c
##     D     E     F     G     H     I     J
##  6775  9797  9542 11292  8304  5422  2808
```

Convert the "c" vector to X which contains an intercept and an appropriate number of dummies. Let the color G be the refernce category as it is the modal color. Name the columns of X appropriately. The first should be "(Intercept)". Delete c.

```
X = rep(1, nrow(diamonds))
X = cbind(X, diamonds$color =='D')
X = cbind(X, diamonds$color =='E')
```

```r
X = cbind(X, diamonds$color =='F')
X = cbind(X, diamonds$color =='H')
X = cbind(X, diamonds$color =='I')
X = cbind(X, diamonds$color =='J')
colnames(X) = c("(Intercept)", "is_D", "is_E", "is_F", "is_H", "is_I", "is_J")
head(X)
```

```
##      (Intercept) is_D is_E is_F is_H is_I is_J
## [1,]           1    0    1    0    0    0    0
## [2,]           1    0    1    0    0    0    0
## [3,]           1    0    1    0    0    0    0
## [4,]           1    0    0    0    0    1    0
## [5,]           1    0    0    0    0    0    1
## [6,]           1    0    0    0    0    0    1
```

```r
rm(c)
```

Repeat the iterative exercise above we did for Boston here.

```r
M = matrix(NA, nrow = ncol(X), ncol = ncol(X))
colnames(M) = colnames(X)
for(j in 1 : ncol(M)){
  X_j       = X[ , 1 : j, drop = FALSE]
  b = solve(t(X_j) %*% X_j) %*% t(X_j) %*% y
  M[j, 1 : j] = b
}
M
```

```
##      (Intercept)       is_D       is_E      is_F      is_H      is_I
## [1,]    3932.800         NA         NA        NA        NA        NA
## [2,]    4042.378  -872.4243         NA        NA        NA        NA
## [3,]    4295.543 -1125.5885 -1218.7901        NA        NA        NA
## [4,]    4491.230 -1321.2760 -1414.4776 -766.3437        NA        NA
## [5,]    4493.170 -1323.2160 -1416.4176 -768.2837  -6.50092        NA
## [6,]    4262.945 -1092.9907 -1186.1923 -538.0584 223.72444  828.9302
## [7,]    3999.136  -829.1816  -922.3832 -274.2493 487.53352 1092.7393
##          is_J
## [1,]       NA
## [2,]       NA
## [3,]       NA
## [4,]       NA
## [5,]       NA
## [6,]       NA
## [7,] 1324.682
```

Why didn't the estimates change as we added more and more features?

They did change.

Create a vector y by simulating $n = 100$ standard iid normals. Create a matrix of size 100 x 2 and populate the first column by all ones (for the intercept) and the second column by 100 standard iid normals. Find the R^2 of an OLS regression of `y ~ X`. Use matrix algebra.

```r
y = rnorm(100)
X = matrix(cbind(rep(1, 100), rnorm(100)), nrow = 100, ncol = 2)

b = solve(t(X) %*% X) %*% t(X) %*% y
```

```r
y_hat = X %*% b

e = y - y_hat
sse = sum(e^2)
sst = sum((y - mean(y))^2)
R_sqrd = 1 - sse/sst
R_sqrd
```

```
## [1] 0.0008053215
```

from the last problem. Find the R^2 of an OLS regression of `y ~ X`. You can use the `summary` function of an `lm` model.

```r
OLS_reg = lm(y ~ X)
summary(OLS_reg)$r.squared
```

```
## [1] 0.0008053215
```

Write a for loop to each time bind a new column of 100 standard iid normals to the matrix X and find the R^2 each time until the number of columns is 100. Create a vector to save all R^2's. What happened??

```r
init_Col = ncol(X) + 1
R_sqrd_vec = c()

for(j in (init_Col : nrow(X))){
  X = cbind(X, rnorm(nrow(X)))
  R_sqrd_vec = append(R_sqrd_vec, summary(lm(y~X))$r.squared)
}

R_sqrd_vec
```

```
##   [1] 0.003928274 0.006595160 0.026153115 0.036763547 0.050287109
##   [6] 0.052827253 0.077481528 0.078234375 0.078275335 0.078325190
##  [11] 0.087038876 0.087039235 0.091619265 0.115697900 0.115783473
##  [16] 0.129023231 0.129412653 0.129510137 0.130206004 0.142651264
##  [21] 0.184768317 0.202754756 0.217800417 0.225888968 0.231770320
##  [26] 0.249206953 0.256771043 0.257510571 0.259888841 0.261287053
##  [31] 0.273904972 0.276787587 0.280338841 0.317398932 0.351611408
##  [36] 0.370534327 0.383667300 0.425887294 0.446666700 0.464739008
##  [41] 0.464801246 0.466120312 0.466564150 0.481558192 0.482529221
##  [46] 0.483253656 0.517906887 0.554233775 0.574631690 0.575090431
##  [51] 0.575840420 0.588283792 0.588308627 0.588516215 0.588685492
##  [56] 0.598877725 0.600676021 0.611695419 0.611752960 0.612040406
##  [61] 0.634627800 0.654837011 0.669248525 0.687198068 0.710781863
##  [66] 0.713885830 0.713894801 0.715030426 0.715180478 0.721375755
##  [71] 0.729038337 0.740045307 0.740279858 0.741954059 0.758446558
##  [76] 0.760345072 0.761183632 0.780241248 0.781065685 0.790778649
##  [81] 0.799150498 0.802757106 0.805062300 0.819127484 0.825172389
##  [86] 0.834451879 0.841493213 0.883916176 0.890246932 0.893600393
##  [91] 0.957999052 0.968699221 0.981468730 0.989241180 0.995973128
##  [96] 0.997941399 0.999639357 1.000000000
```

```r
#R^2 goes up every time we add a new column to the design matrix.
```

Add one final column to X to bring the number of columns to 101. Then try to compute R^2. What happens and why?

```r
X = cbind(X, rnorm(nrow(X)))
summary(lm(y~X))$r.squared
```

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

```r
#R^2 is one, y now exists in the Colspace[X], so y can be described as
#a linear combination of the columns of X, and e is 0, so R^2 is 1.
```