Lab 5

Sakib Salim

11:59PM March 17, 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)".

```
data(Boston, package = "MASS")
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)
pacman::p_load(Matrix)
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
```

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

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

Calculate $\hat{y}$.

```
yhat = H %*% y
head(yhat)
```

```
##       [,1]
## 1 30.00384
## 2 25.02556
## 3 30.56760
## 4 28.60704
## 5 27.94352
## 6 25.25628
```

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.

```
e = y - yhat
e_2 = H_comp %*% y
expect_equal(e, e_2)
```

Calculate R^2 and RMSE.

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

Rsquared = 1 - sse / sst
Rsquared
```

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

```
mse = sse / (nrow(X) - ncol(X))
rmse = sqrt(mse) #rmse is standard deviation of errors
rmse
```

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

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

```
t(e) %*% yhat
```

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

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

```
t(e) %*% (yhat - 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.

```
y_minus_y_bar = y - mean(y)
yhat_minus_y_bar = yhat - mean(y)
len_y_minus_y_bar = sqrt(  sum(y_minus_y_bar^2)  )
len_yhat_minus_y_bar = sqrt(  sum(yhat_minus_y_bar^2)  )

theta = acos( (t(y_minus_y_bar) %*% yhat_minus_y_bar) / (len_y_minus_y_bar * len_yhat_minus_y_bar) )
cos_theta_sqrd = cos(theta)^2
cos_theta_sqrd
```

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

```
Rsquared
```

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

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

```
len_y_minus_y_bar^2 - len_yhat_minus_y_bar^2 - sse
```

```
## [1] 5.666152e-09
```

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.

```
M = matrix(NA, nrow = ncol(X), ncol = ncol(X))
colnames(M) = colnames(X)
X_j = X[, 1, drop = FALSE]
b = solve(t(X_j) %*% X_j) %*% t(X_j) %*% y
M[1, 1] = b
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
```

```
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?

TO-DO

Clear the workspace and load the diamonds dataset.

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

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

```r
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  
## 
```

```r
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')
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
```

Repeat the iterative exercise above we did for Boston here.

```
#TO-DO
```

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

TO-DO

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.

```
n = 100
p = 2
y = rnorm(n)
X = rep(1,n)
X = cbind(X,rnorm(n))
H = X %*% solve(t(X) %*% X) %*% t(X)
yhat = H %*% y
e = y-yhat
ybar = mean(y)
dev_mean = y-ybar
SSE = sum(e^2)
SST = sum(dev_mean^2)
Rsqd = 1-SSE/SST
Rsqd
```

```
## [1] 0.0008131394
```

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

```
## [1] 0.0008131394
```

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.

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
#TO-DO
rsqd_vec = rep(NA,n-2)
for ( i in 1:(n-2)){
  new_col = rnorm(n)
  X = cbind(X, new_col)
  rsqd_vec[i] = summary(lm(y~X))$r.squared
}
rsqd_vec
```

```
##   [1] 0.005646113 0.006192343 0.015101381 0.015163477 0.016042681
##   [6] 0.016894059 0.070935916 0.071602192 0.071618863 0.095176931
##  [11] 0.111961401 0.146683300 0.149520911 0.181424339 0.183503600
##  [16] 0.184761449 0.197067914 0.219540296 0.230490084 0.232196511
##  [21] 0.241700067 0.264591637 0.270196341 0.272040539 0.329763681
##  [26] 0.360152706 0.360155852 0.367003617 0.369670162 0.369670239
##  [31] 0.397094664 0.398046716 0.404810688 0.406153866 0.408662459
##  [36] 0.430403075 0.501532084 0.504722914 0.569409700 0.569463429
##  [41] 0.569544420 0.588041543 0.593076422 0.606398567 0.608325363
##  [46] 0.616043615 0.617623623 0.618285891 0.625633874 0.633043337
##  [51] 0.633056671 0.642483816 0.645178991 0.650467386 0.655971151
##  [56] 0.676035081 0.677479836 0.681365468 0.723715369 0.748088057
##  [61] 0.748388262 0.753757737 0.780605523 0.786418346 0.787114088
##  [66] 0.792707298 0.796163147 0.800279230 0.800722126 0.811423285
##  [71] 0.833366444 0.833748114 0.835716269 0.847148544 0.861493194
##  [76] 0.896400915 0.897428485 0.897998421 0.898124228 0.899123400
##  [81] 0.939155807 0.939650009 0.939839792 0.940869635 0.942925375
##  [86] 0.952473101 0.952952473 0.960092286 0.973971695 0.975954544
##  [91] 0.975982878 0.976922677 0.978859169 0.980222696 0.984167494
##  [96] 0.991929715 0.994988672 1.000000000
```

```r
#R^2 = 1 when the number of columns (i.e. features) equals the number of observations
```

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
#TO-DO
X = cbind(X,rnorm(n))
summary(lm(y~X))$r.squared
```

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

```r
#The supremum of R^2 is 1 and there is no way it could be greater than 1 since SSE/SST > 0.
```