Lab 4

Christian Guaraca

11:59PM March 10, 2021

Load up the famous iris dataset. We are going to do a different prediction problem. Imagine the only input x is Species and you are trying to predict y which is Petal.Length. A reasonable prediction is the average petal length within each Species. Prove that this is the OLS model by fitting an appropriate `lm` and then using the predict function to verify.

```
data(iris)
mod = lm(Petal.Length ~ Species, iris)
mean(iris$Petal.Length[iris$Species == "setosa"])
```

```
## [1] 1.462
```

```
mean(iris$Petal.Length[iris$Species == "versicolor"])
```

```
## [1] 4.26
```

```
mean(iris$Petal.Length[iris$Species == "virginica"])
```

```
## [1] 5.552
```

```
predict(mod, data.frame(Species = c("setosa")))
```

```
##     1
## 1.462
```

```
predict(mod, data.frame(Species = c("versicolor")))
```

```
##    1
## 4.26
```

```
predict(mod, data.frame(Species = c("virginica")))
```

```
##     1
## 5.552
```

Construct the design matrix with an intercept, X, without using `model.matrix`.

```
X <- cbind(1, iris$Species == "versicolor", iris$Species == "virginica" )

head(X)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    1    0    0
## [3,]    1    0    0
## [4,]    1    0    0
## [5,]    1    0    0
## [6,]    1    0    0
```

Find the hat matrix H for this regression.

```
H = X %*% solve(t(X) %*% X) %*% t(X)
Matrix::rankMatrix(H)
```

```
## [1] 3
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 3.330669e-14
```

```
#head(H)
```

Verify this hat matrix is symmetric using the `expect_equal` function in the package `testthat`.

```
pacman::p_load(testthat)
expect_equal(H, t(H))
```

Verify this hat matrix is idempotent using the `expect_equal` function in the package `testthat`.

```
expect_equal(H, H%*%H)
```

Using the `diag` function, find the trace of the hat matrix.

```
sum(diag(H))
```

```
## [1] 3
```

```
#trace same as the rank
```

It turns out the trace of a hat matrix is the same as its rank! But we don't have time to prove these interesting and useful facts..

For masters students: create a matrix $X_\perp$.

```
#TO-DO
```

Using the hat matrix, compute the $\hat{y}$ vector and using the projection onto the residual space, compute the e vector and verify they are orthogonal to each other.

```
y = iris$Petal.Length
y_hat = H %*% y
e = (diag(nrow(iris))-H) %*% y
e
```

```
##          [,1]
##   [1,] -0.062
##   [2,] -0.062
##   [3,] -0.162
##   [4,]  0.038
##   [5,] -0.062
##   [6,]  0.238
##   [7,] -0.062
##   [8,]  0.038
##   [9,] -0.062
##  [10,]  0.038
##  [11,]  0.038
##  [12,]  0.138
##  [13,] -0.062
##  [14,] -0.362
```

```
##  [15,] -0.262
##  [16,]  0.038
##  [17,] -0.162
##  [18,] -0.062
##  [19,]  0.238
##  [20,]  0.038
##  [21,]  0.238
##  [22,]  0.038
##  [23,] -0.462
##  [24,]  0.238
##  [25,]  0.438
##  [26,]  0.138
##  [27,]  0.138
##  [28,]  0.038
##  [29,] -0.062
##  [30,]  0.138
##  [31,]  0.138
##  [32,]  0.038
##  [33,]  0.038
##  [34,] -0.062
##  [35,]  0.038
##  [36,] -0.262
##  [37,] -0.162
##  [38,] -0.062
##  [39,] -0.162
##  [40,]  0.038
##  [41,] -0.162
##  [42,] -0.162
##  [43,] -0.162
##  [44,]  0.138
##  [45,]  0.438
##  [46,] -0.062
##  [47,]  0.138
##  [48,] -0.062
##  [49,]  0.038
##  [50,] -0.062
##  [51,]  0.440
##  [52,]  0.240
##  [53,]  0.640
##  [54,] -0.260
##  [55,]  0.340
##  [56,]  0.240
##  [57,]  0.440
##  [58,] -0.960
##  [59,]  0.340
##  [60,] -0.360
##  [61,] -0.760
##  [62,] -0.060
##  [63,] -0.260
##  [64,]  0.440
##  [65,] -0.660
##  [66,]  0.140
##  [67,]  0.240
##  [68,] -0.160
```

```
##  [69,]  0.240
##  [70,] -0.360
##  [71,]  0.540
##  [72,] -0.260
##  [73,]  0.640
##  [74,]  0.440
##  [75,]  0.040
##  [76,]  0.140
##  [77,]  0.540
##  [78,]  0.740
##  [79,]  0.240
##  [80,] -0.760
##  [81,] -0.460
##  [82,] -0.560
##  [83,] -0.360
##  [84,]  0.840
##  [85,]  0.240
##  [86,]  0.240
##  [87,]  0.440
##  [88,]  0.140
##  [89,] -0.160
##  [90,] -0.260
##  [91,]  0.140
##  [92,]  0.340
##  [93,] -0.260
##  [94,] -0.960
##  [95,] -0.060
##  [96,] -0.060
##  [97,] -0.060
##  [98,]  0.040
##  [99,] -1.260
## [100,] -0.160
## [101,]  0.448
## [102,] -0.452
## [103,]  0.348
## [104,]  0.048
## [105,]  0.248
## [106,]  1.048
## [107,] -1.052
## [108,]  0.748
## [109,]  0.248
## [110,]  0.548
## [111,] -0.452
## [112,] -0.252
## [113,] -0.052
## [114,] -0.552
## [115,] -0.452
## [116,] -0.252
## [117,] -0.052
## [118,]  1.148
## [119,]  1.348
## [120,] -0.552
## [121,]  0.148
## [122,] -0.652
```

```
## [123,]  1.148
## [124,] -0.652
## [125,]  0.148
## [126,]  0.448
## [127,] -0.752
## [128,] -0.652
## [129,]  0.048
## [130,]  0.248
## [131,]  0.548
## [132,]  0.848
## [133,]  0.048
## [134,] -0.452
## [135,]  0.048
## [136,]  0.548
## [137,]  0.048
## [138,] -0.052
## [139,] -0.752
## [140,] -0.152
## [141,]  0.048
## [142,] -0.452
## [143,] -0.452
## [144,]  0.348
## [145,]  0.148
## [146,] -0.352
## [147,] -0.552
## [148,] -0.352
## [149,] -0.152
## [150,] -0.452
```

Compute SST, SSR and SSE and R^2 and then show that SST = SSR + SSE.

```
SSE = t(e) %*% e
SSE
```

```
##         [,1]
## [1,] 27.2226
```

```
y_bar = mean(y)
SST = t(y - y_bar) %*% (y - y_bar)
SST
```

```
##          [,1]
## [1,] 464.3254
```

```
Rsq = 1 - SSE/SST
Rsq
```

```
##           [,1]
## [1,] 0.9413717
```

```
SSR = t(y_hat - y_bar) %*% (y_hat - y_bar)
SSR
```

```
##          [,1]
## [1,] 437.1028
```

```
expect_equal(SSR + SSE, SST)
```

Find the angle θ between y - $\bar{y}1$ and $\hat{y} - \bar{y}1$ and then verify that its cosine squared is the same as the R^2

from the previous problem.

```
theta = acos(t(y - y_bar) %*% (y_hat - y_bar) / sqrt(SST * SSR))
theta * (180 / pi)
```

```
##          [,1]
## [1,] 14.01245
```

Project the y vector onto each column of the X matrix and test if the sum of these projections is the same as yhat.

```
proj1 = (X[,1] %*% t(X[,1]) / as.numeric(t(X[,1]) %*% X[,1])) %*% y
proj2 = (X[,2] %*% t(X[,2]) / as.numeric(t(X[,2]) %*% X[,2])) %*% y
proj3 = (X[,3] %*% t(X[,3]) / as.numeric(t(X[,3]) %*% X[,3])) %*% y
```

Construct the design matrix without an intercept, X, without using `model.matrix`.

```
x_int = cbind(as.numeric(iris$Species == "setosa"), iris$Species == "versicolor", iris$Species == "virg
head(x_int)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    1    0    0
## [3,]    1    0    0
## [4,]    1    0    0
## [5,]    1    0    0
## [6,]    1    0    0
```

Find the OLS estimates using this design matrix. It should be the sample averages of the petal lengths within species.

```
y = iris$Petal.Length
H_int = x_int %*% solve(t(x_int) %*% x_int) %*% t(x_int)
y_hat_int = H_int %*% y

unique(y_hat_int)
```

```
##       [,1]
## [1,] 1.462
## [2,] 4.260
## [3,] 5.552
```

```
mean(iris$Petal.Length[iris$Species == "setosa"])
```

```
## [1] 1.462
```

```
mean(iris$Petal.Length[iris$Species == "versicolor"])
```

```
## [1] 4.26
```

```
mean(iris$Petal.Length[iris$Species == "virginica"])
```

```
## [1] 5.552
```

Verify the hat matrix constructed from this design matrix is the same as the hat matrix constructed from the design matrix with the intercept. (Fact: orthogonal projection matrices are unique).

```
expect_equal(H, H_int)
```

Project the y vector onto each column of the X matrix and test if the sum of these projections is the same as yhat.

```
proj1 = (x_int[,1] %*% t(x_int[,1]) / as.numeric(t(x_int[,1]) %*% x_int[,1])) %*% y
proj2 = (x_int[,2] %*% t(x_int[,2]) / as.numeric(t(x_int[,2]) %*% x_int[,2])) %*% y
proj3 = (x_int[,3] %*% t(x_int[,3]) / as.numeric(t(x_int[,3]) %*% x_int[,3])) %*% y

expect_equal(proj1+proj2+proj3, y_hat_int)
```

Convert this design matrix into Q, an orthonormal matrix.

```
Q = qr.Q(qr(x_int))

sum(Q[, 1]^2)
```

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

```
sum(Q[, 2]^2)
```

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

```
sum(Q[, 3]^2)
```

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

```
Q[, 1] %*% Q[, 2]
```

```
##      [,1]
## [1,]    0
```

```
Q[, 1] %*% Q[, 3]
```

```
##      [,1]
## [1,]    0
```

```
Q[, 2] %*% Q[, 3]
```

```
##      [,1]
## [1,]    0
```

Project the y vector onto each column of the Q matrix and test if the sum of these projections is the same as yhat.

```
pro1 = (Q[,1] %*% t(Q[,1]) / as.numeric(t(Q[,1]) %*% Q[,1])) %*% y
pro2 = (Q[,2] %*% t(Q[,2]) / as.numeric(t(Q[,2]) %*% Q[,2])) %*% y
pro3 = (Q[,3] %*% t(Q[,3]) / as.numeric(t(Q[,3]) %*% Q[,3])) %*% y

expect_equal(pro1+pro2+pro3, y_hat_int)
```

Find the $p = 3$ linear OLS estimates if Q is used as the design matrix using the `lm` method. Is the OLS solution the same as the OLS solution for X?

```
model_Q = lm(Petal.Length ~ 0 + Q, iris)
model_Q
```

```
##
## Call:
## lm(formula = Petal.Length ~ 0 + Q, data = iris)
##
## Coefficients:
##     Q1      Q2      Q3
## -10.34  -30.12  -39.26
```

```r
model_x = lm(y ~ X, iris)
model_x
```

```
##
## Call:
## lm(formula = y ~ X, data = iris)
##
## Coefficients:
## (Intercept)           X1           X2           X3
##       1.462           NA        2.798        4.090
```

```r
#The solutions are not the same
```

Use the predict function and ensure that the predicted values are the same for both linear models: the one created with X as its design matrix and the one created with Q as its design matrix.

```r
predict(model_Q, data.frame(Q))
```

```
##     1     2     3     4     5     6     7     8     9    10    11    12    13
## 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462
##    14    15    16    17    18    19    20    21    22    23    24    25    26
## 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462
##    27    28    29    30    31    32    33    34    35    36    37    38    39
## 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462
##    40    41    42    43    44    45    46    47    48    49    50    51    52
## 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 4.260 4.260
##    53    54    55    56    57    58    59    60    61    62    63    64    65
## 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260
##    66    67    68    69    70    71    72    73    74    75    76    77    78
## 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260
##    79    80    81    82    83    84    85    86    87    88    89    90    91
## 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260
##    92    93    94    95    96    97    98    99   100   101   102   103   104
## 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 5.552 5.552 5.552 5.552
##   105   106   107   108   109   110   111   112   113   114   115   116   117
## 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552
##   118   119   120   121   122   123   124   125   126   127   128   129   130
## 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552
##   131   132   133   134   135   136   137   138   139   140   141   142   143
## 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552
##   144   145   146   147   148   149   150
## 5.552 5.552 5.552 5.552 5.552 5.552 5.552
```

```r
predict(model_x, data.frame(X[1]))
```

```
## Warning: 'newdata' had 1 row but variables found have 150 rows
```

```
## Warning in predict.lm(model_x, data.frame(X[1])): prediction from a rank-
## deficient fit may be misleading
```

```
##     1     2     3     4     5     6     7     8     9    10    11    12    13
## 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462
##    14    15    16    17    18    19    20    21    22    23    24    25    26
## 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462
##    27    28    29    30    31    32    33    34    35    36    37    38    39
## 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462
##    40    41    42    43    44    45    46    47    48    49    50    51    52
```

```
## 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 1.462 4.260 4.260
##    53    54    55    56    57    58    59    60    61    62    63    64    65
## 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260
##    66    67    68    69    70    71    72    73    74    75    76    77    78
## 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260
##    79    80    81    82    83    84    85    86    87    88    89    90    91
## 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260
##    92    93    94    95    96    97    98    99   100   101   102   103   104
## 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 4.260 5.552 5.552 5.552 5.552
##   105   106   107   108   109   110   111   112   113   114   115   116   117
## 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552
##   118   119   120   121   122   123   124   125   126   127   128   129   130
## 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552
##   131   132   133   134   135   136   137   138   139   140   141   142   143
## 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552 5.552
##   144   145   146   147   148   149   150
## 5.552 5.552 5.552 5.552 5.552 5.552 5.552
```

Clear the workspace and load the boston housing data and extract X and y. The dimensions are $n = 506$ and $p = 13$. 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
B = MASS::Boston
int = rep(1, nrow(B))
x = cbind(int,B[, 1:13])
y = B[, 14]

p_plus_one = matrix(data = NA, nrow = 14, ncol = 14)
colnames(p_plus_one) = c(colnames(x))
for(i in 1:ncol(p_plus_one)){
  x_1 = x[, 1:i]
  x_1 = as.matrix(x_1)
  p_plus_one[i, 1:i] = solve(t(x_1) %*% x_1) %*% t(x_1) %*% y
}
p_plus_one
```

```
##               int       crim         zn       indus     chas        nox
##  [1,]  22.5328063         NA         NA          NA       NA         NA
##  [2,]  24.0331062 -0.4151903         NA          NA       NA         NA
##  [3,]  22.4856281 -0.3520783 0.11610909          NA       NA         NA
##  [4,]  27.3946468 -0.2486283 0.05850082 -0.41557782       NA         NA
##  [5,]  27.1128031 -0.2287981 0.05928665 -0.44032511 6.894059         NA
##  [6,]  29.4899406 -0.2185190 0.05511047 -0.38348055 7.026223  -5.424659
##  [7,] -17.9546350 -0.1769135 0.02128135 -0.14365267 4.784684  -7.184892
##  [8,] -18.2649261 -0.1727607 0.01421402 -0.13089918 4.840730  -4.357411
##  [9,]   0.8274820 -0.1977868 0.06099257 -0.22573089 4.577598 -14.451531
## [10,]   0.1553915 -0.1780398 0.06095248 -0.21004328 4.536648 -13.342666
## [11,]   2.9907868 -0.1795543 0.07145574 -0.10437742 4.110667 -12.591596
## [12,]  27.1523679 -0.1840321 0.03909990 -0.04232450 3.487528 -22.182110
## [13,]  20.6526280 -0.1599391 0.03887365 -0.02792186 3.216569 -20.484560
## [14,]  36.4594884 -0.1080114 0.04642046  0.02055863 2.686734 -17.766611
```

```
##              rm           age       dis         rad         tax    ptratio
##  [1,]       NA            NA        NA          NA          NA         NA
##  [2,]       NA            NA        NA          NA          NA         NA
##  [3,]       NA            NA        NA          NA          NA         NA
##  [4,]       NA            NA        NA          NA          NA         NA
##  [5,]       NA            NA        NA          NA          NA         NA
##  [6,]       NA            NA        NA          NA          NA         NA
##  [7,] 7.341586            NA        NA          NA          NA         NA
##  [8,] 7.386357 -0.0236248493        NA          NA          NA         NA
##  [9,] 6.752352 -0.0556354540 -1.760312          NA          NA         NA
## [10,] 6.791184 -0.0562612189 -1.748296 -0.04529059          NA         NA
## [11,] 6.664084 -0.0546675064 -1.727933  0.15926305 -0.01434060         NA
## [12,] 6.075744 -0.0451880522 -1.583852  0.25472196 -0.01221262 -0.9962062
## [13,] 6.123072 -0.0459320518 -1.554912  0.28157503 -0.01173838 -1.0142228
## [14,] 3.809865  0.0006922246 -1.475567  0.30604948 -0.01233459 -0.9527472
##             black      lstat
##  [1,]          NA         NA
##  [2,]          NA         NA
##  [3,]          NA         NA
##  [4,]          NA         NA
##  [5,]          NA         NA
##  [6,]          NA         NA
##  [7,]          NA         NA
##  [8,]          NA         NA
##  [9,]          NA         NA
## [10,]          NA         NA
## [11,]          NA         NA
## [12,]          NA         NA
## [13,] 0.013620833         NA
## [14,] 0.009311683 -0.5247584
```

Why are the estimates changing from row to row as you add in more predictors?

#They are changing row from row because as the model recieves more predictors it is trying to adjust to it for all the data it is taking in order to have a good fitting line.

Create a vector of length $p + 1$ and compute the R^2 values for each of the above models.

```
R_sq = array(dim = 14)
y_bar = mean(y)
SST = sum((y - y_bar)^2)
for(i in 1:nrow(p_plus_one)){
  t = c(p_plus_one[i, 1:i], rep(0, nrow(p_plus_one) - i))
  y_hat = x_1 %*% t
  SSR = sum((y_hat - y_bar)^2)
  Rsq = SSR / SST
  R_sq[i] = Rsq
}
R_sq
```

```
##  [1] 5.382448e-30 1.507805e-01 2.339884e-01 2.937136e-01 3.295277e-01
##  [6] 3.313127e-01 5.873770e-01 5.894902e-01 6.311488e-01 6.319479e-01
## [11] 6.396628e-01 6.703141e-01 6.842043e-01 7.406427e-01
```

Is R^2 monotonically increasing? Why?

#The R^2 is monotonically increasing because it is filling the whole colspace.