Lab 4

Amy Butler

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.

#We need to prove the predictions will match the y_bar of each species #The numbers match so this worked #Find the y_bar for each species

```
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 for the previous linear model with an intercept, X, without using `model.matrix`.

#A design matrix makes column vectors that measure what we care about (in this case species) and a column of ones to fit an intercept.

```
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.

#%*% is matrix multiplication #Solve() finds the inverse of a matrix #t(X) is X transpose #The rank should be three because there are 3 columns in X and the projectin of yhat onto the column space of X should be a linear combination of the 3 columns of X.

```
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
```

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

#Pacman loads a package #If there is no error then it worked and the matrix is symmetric

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

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

#No output means it works and the matrix is idempotent

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

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

#Diag returns, extracts, or replaces the diagonal of a matrix #Trace is the sum of the diagonal #The trace is also the rank

```
sum(diag(H))

## [1] 3
```

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 (H), 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.

#H%*% does a projection #The table should display the 50 y_bar's for each of the three species #e should be a column vector of decimals with no pattern

```
y=iris$Petal.Length
y_hat= H %*% y
table(y_hat)

## y_hat
## 1.462  4.26 5.552
##    50    50    50

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.

#The SSE & SST can be written in the previous formula format we used or in matrix form like below #No output of the expect_equals means that SSR+SSE does equal SST

```
y_bar=mean(y)

SSE= t(e) %*% e
SSE

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

SST= t(y-y_bar) %*% (y-y_bar)
SSR= t(y_hat-y_bar) %*% (y_hat-y_bar)
RSQ= 1-SSE/SST
RSQ

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

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 should be close to zero #pi/180 turns theta into degrees

```
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.

#We want this to fail when adding the projections

```
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`.

```
anova_mod = lm(Petal.Length ~ 0 + Species, iris)
```

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

```
b = solve(t(X)%*%X)%*%t(X)%*%y
Model=lm(Petal.Length~X,iris)
Model

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

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).

```
X = cbind(as.integer(iris$Species == "setosa"), as.integer(iris$Species ==
"versicolor"), as.integer(iris$Species == "virginica"))
H_second = X %*% solve(t(X) %*% X) %*% t(X)
```

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

#Same format as problem above with projections

```
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
```

Convert this design matrix into Q, an orthonormal matrix.

```
qrX=qr(X)
Q=qr.Q(qrX)
```

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

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

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?

```
lm(Petal.Length~Q[,3],iris)

##
## Call:
## lm(formula = Petal.Length ~ Q[, 3], data = iris)
##
## Coefficients:
## (Intercept)       Q[, 3]
##       2.861      -19.028
```

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.

```
predict(Model)

##     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.

```
rm(list=ls())
Boston=MASS::Boston
X=cbind(1, as.matrix(Boston[,1:13]))
y=Boston[,14]
p1=ncol(X)
matrixp1=matrix(NA, nrow=p1, ncol=p1)
for(j in 1:ncol(X)){
  Xj=X[,1:j]
  matrixp1[j,1:j]=solve(t(Xj)%*%Xj)%*%t(Xj)%*%y
}
```

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

Because the predictions are getting better and better with more data.

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

```
vector=c(1:14)
for(i in 1:ncol(X)){
  model=lm(y~X[,1:ncol(X)])
  vector[i]=summary(model)$r.squared
}
```

Is R^2 monotonically increasing? Why?

R-squared is monotonically increasing because with more data there will be a better explanation.