Lab 4

Marin Azhar

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
```

```
#this mean the ols predicts within range
#TO-DO
```

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

```
#what is a design matrix : matrix that stacks the training data row by row
#X = cbind(1, iris$Species)
#head(X)

#will not give good regression b/c it sees it as a continuous variable the it will coheres the facts to
#we only need two cols the 3rd will be the refrance
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.

```
#hat_matrix= in notes do matrix mult..
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))
#you dont need a tolerance.
```

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

```
#TO-DO

#may need a tolerance if H is very large
expect_equal(H, H%*%H)
```

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

```
#TO-DO

sum(diag(H))
```

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

```
#sum of trace is 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-SKIP
```

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.

```
#TO-DO
y=iris$Petal.Length
y_hat = H%*%iris$Petal.Length
#Y_hat is the predictions of y_bars.. we are suppose to see 50 y_bars for each Species
e =(diag(nrow(iris))-H)%*%y
#147 cols

# no discernible pattern,  very complicated projection....
#table(y_hat)
t(e)%*%y_hat #orthogonal
```

```
##              [,1]
## [1,] -2.2915e-13
```

```
Matrix::rankMatrix(e)
```

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

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

```
#TO-DO
SEE =t(e)%*%e
y_bar = mean(y)
SST = t(y - y_bar)%*%(y - y_bar)
Rsq =1 - (SEE/SST)
Rsq
```

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

```
SSR = t(y_hat - y_bar)%*%(y_hat - y_bar)
expect_equal(SEE+SSR, SST)
var(y) #if you say y_bar for eevry error??
```

```
## [1] 3.116278
```

```
var(e) #tremendous reduction on errors
```

```
##          [,1]
## [1,] 0.182702
```

```
#triangle formula
```

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.

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

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

```
#rsq was large so ethsa shold be small
```

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

```
#TO-DO
#lec 10...3/3

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

#expect_equal(proj1 +proj2 +proj3, y_hat)
#you can only add the projection only if it's orthogonal
```

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

```
newmatrix= cbind(iris$Species == "setosa", as.numeric(iris$Species == "versicolor"), as.numeric(iris$Sp
y= iris$Petal.Length

head(newmatrix)
```

```
##      [,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
```

```
#TO-DO
```

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

```
#TO-DO
#OLS IS Y_HAT
#solve is the inverse
H2 =newmatrix%*%solve(t(newmatrix)%*%newmatrix)%*%t(newmatrix)

y_hat2 =H2%*%y

unique(y_hat2)
```

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

```
unique(y_hat)
```

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

```
#actual means
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).

```
#TO-DO

pacman::p_load(testthat)
expect_equal(H, H2)
```

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

```
#TO-DO
qrx= qr(newmatrix)
Q= qr.Q(qrx)
R= qr.R(qrx)


y_proj_X1 =X[,1]%*%t(X[,1])%*%y
y_proj_X2 =X[,2]%*%t(X[,2])%*%y
y_proj_X3 =X[,3]%*%t(X[,3])%*%y
sum_y_proj_on_x= y_proj_X1+ y_proj_X2+ y_proj_X2
#expect_equal(sum_y_proj_on_x, y_hat)
#the projection will not add up unless it's orthogonal and the matrix are not
dim(R)
```

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

```
dim(Q)
```

```
## [1] 150   3
```

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

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

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

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

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

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

Convert this design matrix into Q, an orthonormal matrix.

```
#TO-DO
Qorth= qr(Q)
```

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

```
#TO-DO

y_proj_Q1 =Q[,1]%*%t(Q[,1])%*%y
y_proj_Q2 =Q[,2]%*%t(Q[,2])%*%y
y_proj_Q3 =Q[,3]%*%t(Q[,3])%*%y

sum_y_proj_on_q = y_proj_Q1+y_proj_Q2+ y_proj_Q3

expect_equal(y_hat, sum_y_proj_on_q)
```

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?

```
#TO-DO
mod_x =lm(y~ X, iris)
mod_q =lm(Petal.Length ~ 0 + Q, iris)
# the zero takes out the intercept the q is the ...dumb species
mod_q
```

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

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.

```
#TO-DO
predict(mod_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
```

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

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

```
## Warning in predict.lm(mod_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.

```
#office hrs
rm(list=ls())
Boston = MASS::Boston
intercept = rep(1, nrow(Boston))
X = cbind(intercept, Boston[, 1:13])
y = Boston[,14]
```

```r
matrix1 = matrix(data = NA, nrow = 14, ncol = 14)
colnames(matrix1) = c(colnames(X))

for (i in 1:ncol(matrix1)){
  b=array(NA, dim = ncol(matrix1))
  X_new = X[, 1:i]
  X_new = as.matrix(X_new)
  XTX_inv = solve(t(X_new) %*% X_new)
  b[1:i] = XTX_inv %*% t(X_new) %*% y
  matrix1[i, ] <- b
}
matrix1
```

```
##         intercept       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
```

```
#the B's are the weights
```

Why are the estimates changing from row to row as you add in more predictors? the model is trying fit the new data #TO-DO

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

```
#TO-DO
p_plus_one =14
R_sq_values = array(dim = p_plus_one)
ybar = mean(y)
SST = sum((y - ybar)^2)
for(i in 1:nrow(matrix1)){
  b = c(matrix1[i,1:i],rep(0, nrow(matrix1)-i))
  yhat = X_new%*% b
  SSR = sum((yhat - ybar)^2)
  Rsq = SSR / SST
  R_sq_values[i] = Rsq
}

R_sq_values
```

```
##  [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?

#TO-DO it is increasing because we are overfilling thus we are now spanning the entire col space