Lab 5

Nasif Khan

11:59PM March 18, 2021

Create a 2x2 matrix with the first column 1's and the next column iid normals. Find the absolute value of the angle (in degrees, not radians) between the two columns in absolute difference from 90 degrees.

```
X = matrix(1:1,nrow=2,ncol=2)
X[,2] = rnorm(2)
norm_vec= function(v){
  sqrt(sum(v^2))
}

X <- matrix(1:1,nrow = 2,ncol = 2)
X[,2]= rnorm(2)
#cos_theta= norm_vec(X[,1])
cos_theta = t(X[,1])%*%X[,2]/ (norm_vec(X[,1])* norm_vec(X[,2]))
cos_theta
```

```
##            [,1]
## [1,] -0.5256062
```

```
abs(90- acos(cos_theta)*180/pi)
```

```
##          [,1]
## [1,] 31.70906
```

Repeat this exercise `Nsim = 1e5` times and report the average absolute angle.

```
Nsim = 1e5
angles = array(NA,Nsim)
for(i in 1:Nsim){
X = matrix(1:1,nrow=2,ncol=2)
X[,2] = rnorm(2)
#norm_vec= function(v){
 # sqrt(sum(v^2))
X <- matrix(1:1,nrow = 2,ncol = 2)
X[,2]= rnorm(2)
cos_theta = t(X[,1])%*%X[,2]/ (norm_vec(X[,1])* norm_vec(X[,2]))
angles [i] = abs(90-acos(cos_theta)*180/pi )
}
mean(angles)
```

```
## [1] 44.86853
```

Create a n x 2 matrix with the first column 1's and the next column iid normals. Find the absolute value of the angle (in degrees, not radians) between the two columns. For n = 10, 50, 100, 200, 500, 1000, report the average absolute angle over `Nsim = 1e5` simulations.

```
N_s= c(1, 2, 5, 10, 50, 100, 200, 500, 1000)
Nsim = 1e5
angles = matrix(NA,nrow=Nsim,ncol=length(N_s))
for(j in 1:length(N_s)) {
   for(i in 1:Nsim){
  X = matrix(1,nrow=N_s[j],ncol=2)
    X[,2] = rnorm(N_s[j])
cos_theta = t(X[,1])%*%X[,2] / (norm_vec(X[,1])* norm_vec(X[,2]))
angles [i,j] = abs(90-acos(cos_theta)*180/pi )
   }
}
colMeans(angles)
```

```
## [1] 90.000000 45.081325 23.257531 15.354151  6.553249  4.600035  3.245032
## [8]  2.050290  1.449562
```

What is this absolute angle difference from 90 degrees converging to? Why does this make sense?

#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
X= cbind(1,rnorm(n))
y = rnorm(n)

H = X %*% solve((t(X) %*% X)) %*%  t(X)
y_hat= H%*% y
y_bar= mean(y)

SSR= sum(y_hat-y_hat)^2
SST= sum(y-y_bar)^2

Rsq = (SSR/SST)
Rsq
```

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

```
head(X)
```

```
##      [,1]          [,2]
## [1,]    1 -0.0758830178
## [2,]    1 -0.0003453358
## [3,]    1  0.5355878490
## [4,]    1  1.4278221392
## [5,]    1  0.0239627069
## [6,]    1  0.2505114253
```

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??

```
Rsq_s = array(NA, dim =n-2)

for (j in 1:(n-2)){
  X = cbind(X,rnorm(n))
  H = X %*% solve((t(X) %*% X)) %*%  t(X)
  y_hat= H%*% y
  y_bar= mean(y)

  SSR= sum(y_hat-y_hat)^2
  SST= sum(y-y_bar)^2

Rsq_s [j] = (SSR / SST)
}
Rsq_s
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [77] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
  diff(Rsq_s)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [77] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Test that the projection matrix onto this X is the same as I_n. You may have to vectorize the matrices in the `expect_equal` function for the test to work.

```
pacman::p_load(testthat)
dim(X)
```

```
## [1] 100 100
```

```
H = X %*% solve((t(X) %*% X)) %*% t(X)
H[1:10,1:10]
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  1.000000e+00 -1.043610e-14 -3.418793e-14  5.596652e-15 -6.951384e-14
##  [2,] -3.332664e-14  1.000000e+00 -2.455675e-14 -1.525882e-14  5.401755e-14
##  [3,] -4.122397e-14 -1.601497e-14  1.000000e+00  7.004076e-14  3.953782e-14
##  [4,] -2.808864e-14 -1.970646e-14 -2.495226e-14  1.000000e+00 -3.230749e-14
##  [5,]  2.054606e-14 -1.267736e-14  2.772088e-14  4.300336e-14  1.000000e+00
##  [6,] -2.896121e-14  1.524822e-15  5.496081e-14  1.757852e-14  3.175411e-14
##  [7,]  5.678791e-14  2.242651e-14 -2.817191e-15  1.485530e-14  3.658185e-14
##  [8,]  6.114553e-14  1.515454e-14  8.288509e-15 -1.804481e-14 -3.690104e-14
##  [9,] -8.007484e-14  3.405609e-14  3.859413e-14  6.472167e-14  9.087175e-14
## [10,] -6.875056e-14 -1.212086e-13  6.175616e-14 -4.151670e-14 -1.187939e-13
##                [,6]          [,7]          [,8]          [,9]         [,10]
```

```
##  [1,]  3.217912e-15 -1.012385e-14 -6.550316e-15 -3.360159e-14  9.697104e-15
##  [2,] -4.319570e-14 -9.400987e-14  1.456352e-13 -3.352657e-14  4.704483e-14
##  [3,]  6.870546e-14  7.315676e-14 -4.814205e-14 -5.620504e-15  1.672967e-14
##  [4,]  1.082467e-15 -2.421674e-14 -4.729550e-14 -2.370326e-14  5.250661e-14
##  [5,] -2.330428e-14  5.311376e-14 -3.043399e-14  1.898481e-14  1.655620e-14
##  [6,]  1.000000e+00  3.091060e-14 -4.888451e-15 -8.355621e-14 -1.878294e-14
##  [7,]  1.137111e-14  1.000000e+00 -6.994405e-15 -1.353084e-15 -1.378758e-14
##  [8,] -5.308818e-14 -4.510281e-17  1.000000e+00 -1.602884e-15 -5.776629e-16
##  [9,] -2.863682e-14  5.578871e-15  2.051137e-14  1.000000e+00  6.288026e-14
## [10,] -7.766357e-14  2.710332e-14  8.493206e-15  4.052314e-14  1.000000e+00
```

```
I =diag(n)
expect_equal(H,I)
```

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

```
X = cbind(X,rnorm(n))
 #H = X %*% solve((t(X) %*% X)) %*% t(X)
 y_hat= H%*% y
 y_bar= mean(y)
 SSR= sum(y_hat-y_hat)^2
 SST= sum(y-y_bar)^2
 Rsq = (SSR / SST)
Rsq
```

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

Why does this make sense?

#TO-DO

Write a function spec'd as follows:

```
#' Orthogonal Projection
#'
#' Projects vector a onto v.
#'
#' @param a   the vector to project
#' @param v   the vector projected onto
#'
#' @returns   a list of two vectors, the orthogonal projection parallel to v named a_parallel,
#'            and the orthogonal error orthogonal to v called a_perpendicular
orthogonal_projection = function(a, v){
H = v %*% t(v) / norm_vec(v)^2
a_parallel =H %*% a
a_perpendicular = a - a_parallel
  list(a_parallel = a_parallel, a_perpendicular = a_perpendicular)
}
```

Provide predictions for each of these computations and then run them to make sure you're correct.

```r
orthogonal_projection(c(1,2,3,4), c(1,2,3,4))
```

```
## $a_parallel
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
##
## $a_perpendicular
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
```

```r
#prediction:
orthogonal_projection(c(1, 2, 3, 4), c(0, 2, 0, -1))
```

```
## $a_parallel
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
##
## $a_perpendicular
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
```

```r
#prediction:
result = orthogonal_projection(c(2, 6, 7, 3), c(1, 3, 5, 7) * 37)
t(result$a_parallel) %*% result$a_perpendicular
```

```
##               [,1]
## [1,] -3.552714e-15
```

```r
#prediction:
result$a_parallel + result$a_perpendicular
```

```
##      [,1]
## [1,]    2
## [2,]    6
## [3,]    7
## [4,]    3
```

```
#prediction:
result$a_parallel / c(1, 3, 5 ,7)
```

```
##           [,1]
## [1,] 0.9047619
## [2,] 0.9047619
## [3,] 0.9047619
## [4,] 0.9047619
```

```
#prediction:
```

Let's use the Boston Housing Data for the following exercises

```
y = MASS::Boston$medv
X = model.matrix(medv ~ ., MASS::Boston)
p_plus_one = ncol(X)
n = nrow(X)
```

Using your function `orthogonal_projection` orthogonally project onto the column space of X by projecting y on each vector of X individually and adding up the projections and call the sum `yhat_naive`.

```
yhat_naive = rep(0,n)
  for(j in 1:p_plus_one){
    yhat_naive=yhat_naive+orthogonal_projection(y,X[,j])$a_parallel
  }
```

How much double counting occurred? Measure the magnitude relative to the true LS orthogonal projection.

```
yhat = X %*% solve(t(X) %*% X) %*% t(X) %*% y
sqrt(sum(yhat_naive^2)) / sqrt(sum(yhat^2))
```

```
## [1] 8.997118
```

Is this ratio expected? Why or why not? It's expected to be different.

Convert X into V where V has the same column space as X but has orthogonal columns. You can use the function `orthogonal_projection`. This is the Gram-Schmidt orthogonalization algorithm.

```
V = matrix(NA, nrow = n, ncol = p_plus_one)
V[ , 1] = X[ , 1]
for(j in 2:p_plus_one){
 V[,j] - X[,j]
   for(k in 1:(j-1)){
   V[,j]= V[,j] - orthogonal_projection(X[,j], V[ ,k])$a_parallel
 }
 }
```

Convert V into Q whose columns are the same except normalized

```
Q = matrix(NA, nrow = n, ncol = p_plus_one)
for (j in 1:p_plus_one) {
  Q[,j] = V[,j] / norm_vec(V[,j])
}
```

Verify Q^T Q is I_{p+1} i.e. Q is an orthonormal matrix.

```
#expect_equal(t(Q) %*% Q, (p_plus_one))
```

Is your Q the same as what results from R's built-in QR-decomposition function?

```
Q_from_Rs_builtin =  qr.Q(qr(X))
#expect_equal(Q,Q_from_Rs_builtin)
```

Is this expected? Why did this happen? It doesn't run because it is not equal.

Project y onto colsp[Q] and verify it is the same as the OLS fit. You may have to use the function `unname` to compare the vectors since they the entries will likely have different names.

```
#expect_equal(unname(Q %*% yhat_naive))
```

Project y onto colsp[Q] one by one and verify it sums to be the projection onto the whole space.

```
#yhat_naive =
#projection <- 0
#for (x in 1:p_plus_one){
#  projection = projection + orthogonal_projection(y, Q[ , j])$a_parallel
#}
```

Split the Boston Housing Data into a training set and a test set where the training set is 80% of the observations. Do so at random.

```
K=5
n = 0.8
n_test = round(n * 1 / K)
n_train = n - n_test
#n_model = orthogonal_projection(y, Q[,j])
random_sample = sample(1:n, n_train,FALSE )
```

Fit an OLS model. Find the s_e in sample and out of sample. Which one is greater? Note: we are now using s_e and not RMSE since RMSE has the n-(p + 1) in the denominator not n-1 which attempts to de-bias the error estimate by inflating the estimate when overfitting in high p. Again, we're just using `sd(e)`, the sample standard deviation of the residuals.

```
#TODO
```

Do these two exercises `Nsim = 1000` times and find the average difference between s_e and ooss_e.

```
#TODO
```

We'll now add random junk to the data so that `p_plus_one = n_train` and create a new data matrix `X_with_junk`.

```r
#X_with_junk = cbind(X, matrix(rnorm(n * (n_train - p_plus_one)), nrow = n))
#dim(X)
#dim(X_with_junk)
```

Repeat the exercise above measuring the average s_e and ooss_e but this time record these metrics by number of features used. That is, do it for the first column of `X_with_junk` (the intercept column), then do it for the first and second columns, then the first three columns, etc until you do it for all columns of `X_with_junk`. Save these in `s_e_by_p` and `ooss_e_by_p`.

```r
#TODO
```

You can graph them here:

#{r} #pacman::p_load(ggplot2) #ggplot(# #rbind(#   #data.frame(s_e = s_e_by_p, p = 1 : n_train, series = "in-sample"), #   #data.frame(s_e = ooss_e_by_p, p = 1 : n_train, series = "out-of-sample") #)) + # geom_line(aes(x = p, y = s_e, col = series))

Is this shape expected? Explain.

#TO-DO