Lab 5

Chrstian Guaraca

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.

```
norm_vec = function(v){
  sqrt(sum(v^2))
}

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

```
##           [,1]
## [1,] 0.5805796
```

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

```
##          [,1]
## [1,] 35.49132
```

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)
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] 45.05304
```

Create a 2xn 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(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] 44.972381 23.143318 15.351490  6.549569  4.591453  3.249282  2.052595
## [8]  1.443598
```

What is this absolute angle converging to? Why does this make sense?

The absolute angle difference from ninety is converging to zero. This makes sense because in a high dimensional space random direction in orthogonal.

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_bar)^2)
SST = sum((y - y_bar)^2)

Rsq = (SSR/SST)
Rsq
```

```
## [1] 0.004275145
```

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_bar)^2)
  SST = sum((y - y_bar)^2)

  Rsq_s[j] = (SSR/SST)
}

Rsq_s
```

```
##  [1] 0.00775192 0.01746362 0.02405209 0.02453839 0.04851215 0.05173581
##  [7] 0.05894017 0.06500114 0.07116284 0.07118733 0.08988040 0.12278883
## [13] 0.12417316 0.12776834 0.12825139 0.13161695 0.13981581 0.13998578
## [19] 0.14688776 0.15208749 0.15388471 0.15587389 0.16445722 0.16976154
## [25] 0.18231910 0.18265055 0.18276601 0.18277724 0.19166294 0.19294322
```

```
## [31] 0.19604400 0.20214750 0.20281157 0.20358462 0.20421913 0.20577710
## [37] 0.21257579 0.23772817 0.25221218 0.25792099 0.25837066 0.28083867
## [43] 0.29258622 0.30058509 0.30196726 0.30314829 0.30370084 0.33014053
## [49] 0.33373594 0.38194773 0.40361395 0.40739596 0.40744270 0.41590525
## [55] 0.42725687 0.42772600 0.43160779 0.52772334 0.54175698 0.54180344
## [61] 0.54923267 0.55097269 0.60918172 0.61111344 0.61112501 0.62752711
## [67] 0.69782951 0.71789253 0.71867757 0.74364909 0.76162930 0.76608763
## [73] 0.77576225 0.78251127 0.79539301 0.83081209 0.83128618 0.83513436
## [79] 0.85447642 0.87537431 0.89792213 0.92577475 0.94529893 0.94551830
## [85] 0.94552380 0.94555645 0.94994691 0.95035052 0.95044337 0.95224548
## [91] 0.95276833 0.95356517 0.95850283 0.96233324 0.97261639 0.97355147
## [97] 0.97514410 1.00000000
```

```
diff(Rsq_s)
```

```
##  [1] 9.711703e-03 6.588463e-03 4.863035e-04 2.397376e-02 3.223668e-03
##  [6] 7.204355e-03 6.060975e-03 6.161694e-03 2.448786e-05 1.869307e-02
## [11] 3.290843e-02 1.384322e-03 3.595180e-03 4.830487e-04 3.365570e-03
## [16] 8.198851e-03 1.699694e-04 6.901983e-03 5.199732e-03 1.797221e-03
## [21] 1.989181e-03 8.583324e-03 5.304321e-03 1.255756e-02 3.314534e-04
## [26] 1.154586e-04 1.123407e-05 8.885694e-03 1.280281e-03 3.100783e-03
## [31] 6.103503e-03 6.640640e-04 7.730511e-04 6.345089e-04 1.557975e-03
## [36] 6.798688e-03 2.515237e-02 1.448401e-02 5.708808e-03 4.496720e-04
## [41] 2.246801e-02 1.174755e-02 7.998870e-03 1.382172e-03 1.181028e-03
## [46] 5.525500e-04 2.643969e-02 3.595409e-03 4.821180e-02 2.166621e-02
## [51] 3.782011e-03 4.674312e-05 8.462549e-03 1.135162e-02 4.691266e-04
## [56] 3.881792e-03 9.611554e-02 1.403364e-02 4.645825e-05 7.429235e-03
## [61] 1.740016e-03 5.820903e-02 1.931724e-03 1.157142e-05 1.640210e-02
## [66] 7.030239e-02 2.006303e-02 7.850382e-04 2.497152e-02 1.798022e-02
## [71] 4.458321e-03 9.674626e-03 6.749019e-03 1.288174e-02 3.541908e-02
## [76] 4.740818e-04 3.848186e-03 1.934206e-02 2.089789e-02 2.254782e-02
## [81] 2.785262e-02 1.952418e-02 2.193681e-04 5.499523e-06 3.264697e-05
## [86] 4.390459e-03 4.036122e-04 9.284743e-05 1.802111e-03 5.228487e-04
## [91] 7.968414e-04 4.937662e-03 3.830414e-03 1.028314e-02 9.350814e-04
## [96] 1.592630e-03 2.485590e-02
```

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 -4.401202e-13  8.745227e-13  8.875956e-13 -2.905454e-13
##  [2,] -3.240463e-14  1.000000e+00 -6.326051e-13 -6.102896e-13 -5.284662e-13
##  [3,] -3.856929e-13 -6.305234e-13  1.000000e+00 -4.018019e-13 -1.571746e-14
##  [4,]  3.867115e-13  3.391176e-13 -2.280398e-13  1.000000e+00 -7.948087e-13
##  [5,] -5.034584e-13 -1.966094e-12  3.037570e-13 -4.010126e-13  1.000000e+00
##  [6,]  5.639066e-14  4.358319e-13  6.137313e-13 -6.054324e-13  1.809386e-13
##  [7,]  7.945415e-13 -4.241885e-13  1.289524e-13  1.659228e-13 -1.033673e-12
##  [8,] -2.932168e-13 -8.543166e-14 -7.960299e-13 -7.956968e-13 -2.384759e-13
##  [9,]  7.181755e-13  1.505351e-12  1.147971e-13  3.579914e-13  1.131040e-12
```

```
## [10,] -5.990737e-13  4.767922e-13  4.915096e-13  2.317868e-13  5.559442e-14
##                [,6]          [,7]          [,8]          [,9]         [,10]
##  [1,]  1.557088e-13  4.548584e-13  1.259715e-12  9.319628e-14 -1.324496e-13
##  [2,] -9.214851e-15  5.655476e-13 -1.088463e-12 -2.595077e-13  8.177903e-13
##  [3,] -1.082546e-13  7.065615e-14 -6.312104e-13 -1.599696e-13  1.312266e-13
##  [4,] -4.263256e-14 -4.904965e-13 -1.528222e-13 -2.354228e-13  1.955103e-13
##  [5,]  6.807888e-13 -9.934276e-13 -7.494005e-13  6.454698e-13 -5.750955e-13
##  [6,]  1.000000e+00 -1.351141e-13  2.192968e-13  2.297450e-13 -3.109180e-13
##  [7,]  8.370526e-13  1.000000e+00  6.811218e-13  3.525409e-13  4.507505e-13
##  [8,]  8.847367e-13  1.361133e-13  1.000000e+00  5.254061e-13  8.479883e-13
##  [9,] -6.766809e-14  2.903788e-13  2.353673e-14  1.000000e+00 -1.228573e-12
## [10,] -7.390338e-13 -3.668177e-13 -2.901707e-13 -2.658932e-13  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_bar)^2)
  SST = sum((y - y_bar)^2)

  Rsq = (SSR/SST)

  Rsq
  #suppose to fail
```

Why does this make sense?

This makes sense because you cannot invert a rank deficcent matrix.

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.

```
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))
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
```

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

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

```r
#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)
head(X)
```

```
##   (Intercept)    crim zn indus chas   nox    rm  age    dis rad tax ptratio
## 1           1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3
## 2           1 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8
## 3           1 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8
## 4           1 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7
## 5           1 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7
## 6           1 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7
##    black lstat
## 1 396.90  4.98
## 2 396.90  9.14
## 3 392.83  4.03
## 4 394.63  2.94
## 5 396.90  5.33
## 6 394.12  5.21
```

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?

This is expected to be different than one

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]# - orthogonal_projection(X[,j], V[,j-1])$a_parallel
  for(k in 1:(j-1)){
    V[,j] = V[,j] - orthogonal_projection(X[,j], V[,k])$a_parallel
  }
}

V[,7] %*% V[,9]
```

```
##               [,1]
## [1,] -2.140346e-11
```

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, diag(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) #error expected as well
```

Is this expected? Why did this happen?

There are many orthonormal basis of column space.

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.

```
?unname
y_projection = (Q %*% t(Q) %*% y)
y_1 = lm(y ~ X)$fitted.values
expect_equal(c(unname(y_projection)), unname(y_1))
```

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

```
yhat_naive =
```

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_test = round(n * 1 / K)
n_train = n - n_test

test_set = sample(1:n, n_test)
train_set = setdiff(1:n, test_set)

X_train = X[train_set, ]
y_train = y[train_set]
X_test = X[test_set, ]
y_test = y[test_set]

dim(X_train)
```

```
## [1] 405  14
```

```
dim(X_test)
```

```
## [1] 101  14
```

```
length(y_train)
```

```
## [1] 405
```

```
length(y_test)
```

```
## [1] 101
```

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.

```
OLS_model = lm(y_train ~ .+0, data.frame(X_train))
s_e = sd(OLS_model$residuals)
s_e
```

```
## [1] 4.582256
```

```
y_2 = predict(OLS_model, data.frame(X_test))

oos = y_test - y_2
oos_s_e = sd(oos)
oos_s_e
```

```
## [1] 5.162357
```

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

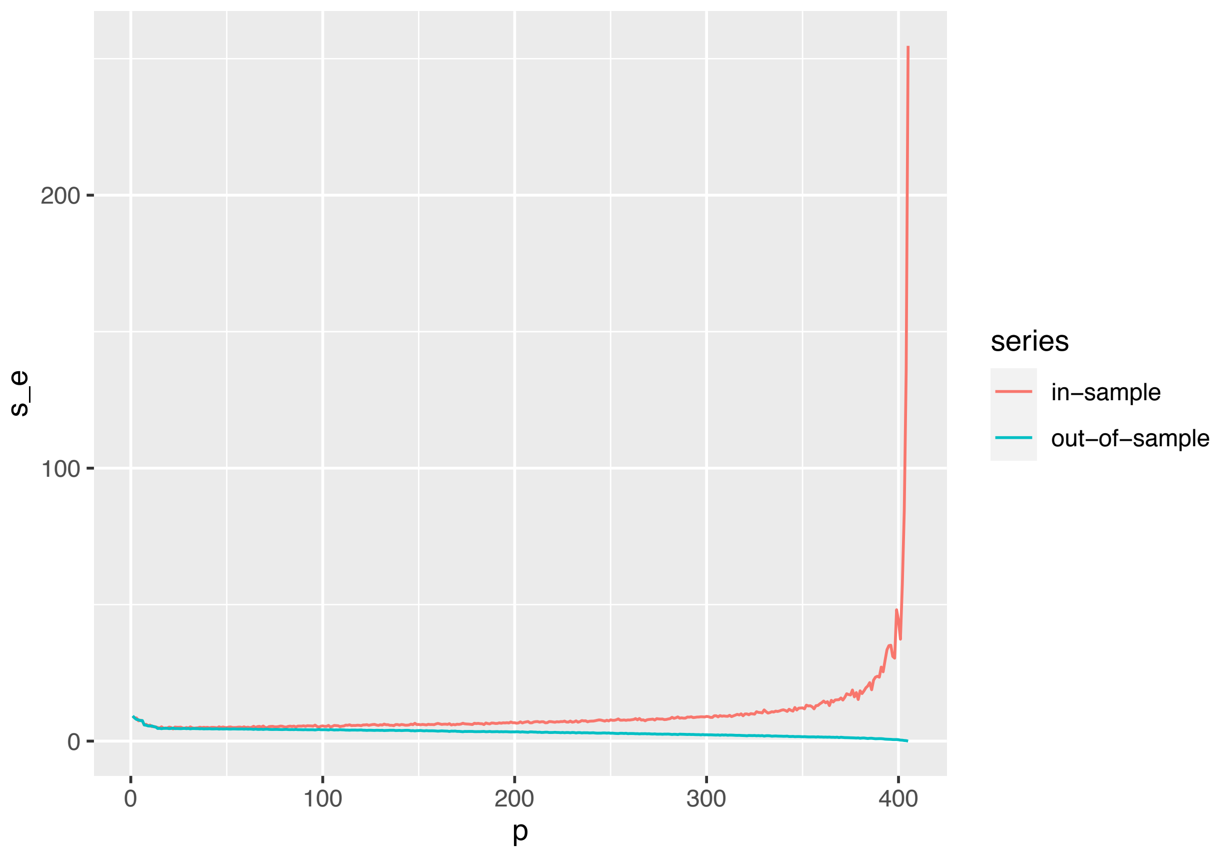
```
Nsim = 1000
average_difference = c()
sum = 0
K = 5
for(i in 1:Nsim){
  test_set = sample(1:n, n_test)
train_set = setdiff(1:n, test_set)

X_train = X[train_set, ]
y_train = y[train_set]
X_test = X[test_set, ]
y_test = y[test_set]

OLS_model = lm(y_train ~ ., data.frame(X_train))
s_e = sd(OLS_model$residuals)

y_2 = predict(OLS_model, data.frame(X_test))

oos = y_test - y_2
oos_s_e = sd(oos)
sum = sum + abs(s_e - oos_s_e)
}
```

```
## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
```

```
## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning in predict.lm(OLS_model, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading
```

You can graph them here:

```
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 = oos_s_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.

#Yes this shape is expected since the number of features increase for the in sample thus making the predictions less accurate.