Lab 5

Jacob Minkin

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 (1:1,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,] -0.9200737
```

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

```
##          [,1]
## [1,] 66.93686
```

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]))
  cos_theta
  angles[i] = abs(90- acos(cos_theta)*180/pi)
}
mean (angles)
```

```
## [1] 44.97355
```

Create a nx2 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(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 (1:1,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]))
```

```
    cos_theta
    angles[i,j] = abs(90- acos(cos_theta)*180/pi)
  }
}
colMeans (angles)
```

```
## [1] 15.346798  6.557522  4.603278  3.249991  2.050273  1.444905
```

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

The absolute angle difference from 90 is converging to zero degrees. It makes sense because in high dimensional space random directions are 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
y = rnorm(n)
X = cbind(1,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] 0.009298645
```

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] 0.02740782 0.03639190 0.04183453 0.04775491 0.04909295 0.06904063
##  [7] 0.10284375 0.10303012 0.11015030 0.11030090 0.12694696 0.12728067
## [13] 0.12748243 0.12752218 0.12805870 0.13134256 0.17342548 0.17365495
## [19] 0.17716760 0.18477440 0.18833452 0.20698817 0.20822179 0.20927946
## [25] 0.21047280 0.21057400 0.21070205 0.21609736 0.22093394 0.23976783
## [31] 0.24664096 0.26635361 0.26682700 0.28069798 0.28173260 0.28185528
```

```
## [37] 0.29231148 0.29256084 0.29536557 0.29689501 0.29695132 0.29696250
## [43] 0.29696511 0.29711383 0.29764979 0.31803823 0.31940743 0.31957700
## [49] 0.33793645 0.35817765 0.41390999 0.41929233 0.41945616 0.49195803
## [55] 0.53525818 0.53533413 0.53663479 0.53667819 0.56487877 0.56563991
## [61] 0.57932625 0.58100610 0.59135988 0.59215920 0.59298982 0.59436016
## [67] 0.59491737 0.62865731 0.65090568 0.65091346 0.66317481 0.66850307
## [73] 0.66954550 0.67595840 0.67614104 0.70251970 0.71102340 0.72684232
## [79] 0.73978045 0.73998347 0.74004050 0.80859133 0.80982030 0.80984566
## [85] 0.81005057 0.81534479 0.88820599 0.89193945 0.89371271 0.90217148
## [91] 0.91289710 0.93001324 0.93275749 0.94356912 0.96629834 0.96986977
## [97] 0.99994039 1.00000000
```

```
diff(rsq_s)
```

```
##  [1] 8.984088e-03 5.442625e-03 5.920380e-03 1.338042e-03 1.994768e-02
##  [6] 3.380311e-02 1.863732e-04 7.120181e-03 1.505989e-04 1.664606e-02
## [11] 3.337093e-04 2.017640e-04 3.975204e-05 5.365210e-04 3.283861e-03
## [16] 4.208291e-02 2.294701e-04 3.512653e-03 7.606800e-03 3.560121e-03
## [21] 1.865364e-02 1.233627e-03 1.057669e-03 1.193334e-03 1.012040e-04
## [26] 1.280507e-04 5.395305e-03 4.836581e-03 1.883389e-02 6.873135e-03
## [31] 1.971265e-02 4.733847e-04 1.387098e-02 1.034625e-03 1.226763e-04
## [36] 1.045620e-02 2.493637e-04 2.804725e-03 1.529441e-03 5.631477e-05
## [41] 1.117726e-05 2.604018e-06 1.487223e-04 5.359615e-04 2.038844e-02
## [46] 1.369198e-03 1.695689e-04 1.835945e-02 2.024121e-02 5.573233e-02
## [51] 5.382340e-03 1.638376e-04 7.250187e-02 4.330015e-02 7.595740e-05
## [56] 1.300659e-03 4.339395e-05 2.820058e-02 7.611365e-04 1.368635e-02
## [61] 1.679842e-03 1.035379e-02 7.993130e-04 8.306208e-04 1.370344e-03
## [66] 5.572088e-04 3.373994e-02 2.224836e-02 7.780528e-06 1.226135e-02
## [71] 5.328260e-03 1.042436e-03 6.412895e-03 1.826419e-04 2.637865e-02
## [76] 8.503706e-03 1.581891e-02 1.293814e-02 2.030136e-04 5.703538e-05
## [81] 6.855083e-02 1.228972e-03 2.535529e-05 2.049088e-04 5.294225e-03
## [86] 7.286120e-02 3.733462e-03 1.773254e-03 8.458775e-03 1.072562e-02
## [91] 1.711614e-02 2.744250e-03 1.081163e-02 2.272922e-02 3.571434e-03
## [96] 3.007061e-02 5.961435e-05
```

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[1:10,1:10]
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  1.000000e+00  6.922241e-14 -4.976575e-14  1.982026e-13  4.166112e-14
##  [2,] -3.915548e-13  1.000000e+00 -1.778785e-14  3.728615e-14 -2.643094e-13
##  [3,] -3.539807e-13 -8.049117e-15  1.000000e+00  7.085582e-13  4.728509e-14
##  [4,] -1.812994e-13 -1.006972e-13  6.110390e-14  1.000000e+00 -3.549383e-13
##  [5,]  1.627205e-13  8.811701e-14  3.849004e-14 -7.208123e-14  1.000000e+00
##  [6,] -2.533390e-14  1.931094e-13 -1.990144e-13  3.416295e-13 -8.461981e-15
##  [7,] -8.881784e-14 -2.714495e-14 -2.944867e-14  2.521316e-13 -2.446654e-14
##  [8,] -2.424172e-13  3.453210e-13  2.860212e-13  1.547928e-13 -2.902088e-13
##  [9,]  1.925682e-13 -1.590394e-13  2.582656e-13 -2.598199e-13  1.680253e-13
## [10,] -3.660683e-13  4.305584e-14 -1.347984e-13  5.156292e-14 -5.906334e-13
##                [,6]          [,7]          [,8]          [,9]         [,10]
```

```
##  [1,] -4.191092e-15  1.427014e-13 -3.715916e-13  5.700301e-15  3.238729e-14
##  [2,]  8.092138e-14  6.097033e-14 -1.237691e-13 -2.395315e-13 -4.465438e-14
##  [3,] -2.691319e-13 -2.794442e-13 -3.421430e-13 -1.249868e-13 -2.013875e-13
##  [4,]  1.181277e-13  8.019258e-14 -1.827982e-13 -1.922941e-13 -2.266763e-13
##  [5,] -9.910475e-14 -1.699607e-13  1.420947e-13 -2.867455e-14  1.897267e-14
##  [6,]  1.000000e+00 -5.753308e-14 -2.473022e-13  1.651890e-13  7.677366e-14
##  [7,]  9.495182e-14  1.000000e+00 -1.159073e-13 -1.531206e-13 -8.014422e-14
##  [8,]  1.583317e-13 -4.535738e-14  1.000000e+00  9.772565e-14  1.216700e-13
##  [9,] -4.649059e-16 -9.128982e-16 -5.945244e-14  1.000000e+00 -4.432912e-14
## [10,]  2.606249e-14 -3.105378e-13 -2.736700e-14  1.225244e-13  1.000000e+00
```

```r
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?

```r
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
```

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

Why does this make sense?

X^t X is not invertible because it is rank deficient

Write a function spec'd as follows:

```r
#' 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))
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

```r
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`.

```r
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.

```r
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 is expected to not be 1.

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.

```r
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)
```

Is this expected? Why did this happen?

YES. There are infinite orthonormal basis

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.

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

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

```
yhat_naive = rep(0,n)

for (j in 1:p_plus_one){
  yhat_naive = yhat_naive + orthogonal_projection(y,Q[,j])$a_parallel
}

expect_equal(yhat_naive, proj_y)
```

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

picked = sample(1:n, n_test)
X_test = X[picked,]; y_test = y[picked]
X_train = X[-picked,]; y_train = y[-picked]
```

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.

```
mod = lm(y_train~.+0,data.frame(X_train))
s_e = sd(mod$residuals)
y_pred = predict(mod, data.frame(X_test))
s_e_oos = sd(y_test - y_pred)

s_e
```

```
## [1] 4.428558
```

```r
s_e_oos
```

```
## [1] 5.635272
```

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

```r
Nsim = 100
diff_se = array(NA,Nsim)
for (i in 1:Nsim)
{
  picked = sample(1:n, n_test)
  X_test = X[picked,]; y_test = y[picked]
  X_train = X[-picked,]; y_train = y[-picked]

  mod = lm(y_train~.+0,data.frame(X_train))
  s_e = sd(mod$residuals)
  y_pred = predict(mod, data.frame(X_test))
  s_e_oos = sd(y_test - y_pred)

  diff_se [i]= s_e - s_e_oos
}
mean(diff_se)
```

```
## [1] -0.2383425
```

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)
```

```
## [1] 506  14
```

```r
dim(X_with_junk)
```

```
## [1] 506 405
```

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
Nsim = 100
ooss_e_by_p = array(NA,ncol(X_with_junk))
s_e_by_p = array(NA,ncol(X_with_junk))
diff_se = array(NA,ncol(X_with_junk))
for (j in 1:ncol(X_with_junk)){

  picked = sample(1:n, n_test)
  X_test = X_with_junk[picked,1:j,drop = FALSE]; y_test = y[picked]
  X_train = X_with_junk[-picked,1:j,drop =FALSE]; y_train = y[-picked]

  mod = lm(y_train~.+0,data.frame(X_train))
  s_e_by_p [j] = sd(mod$residuals)
  y_pred = predict(mod, data.frame(X_test))
  ooss_e_by_p[j] = sd(y_test - y_pred)

```

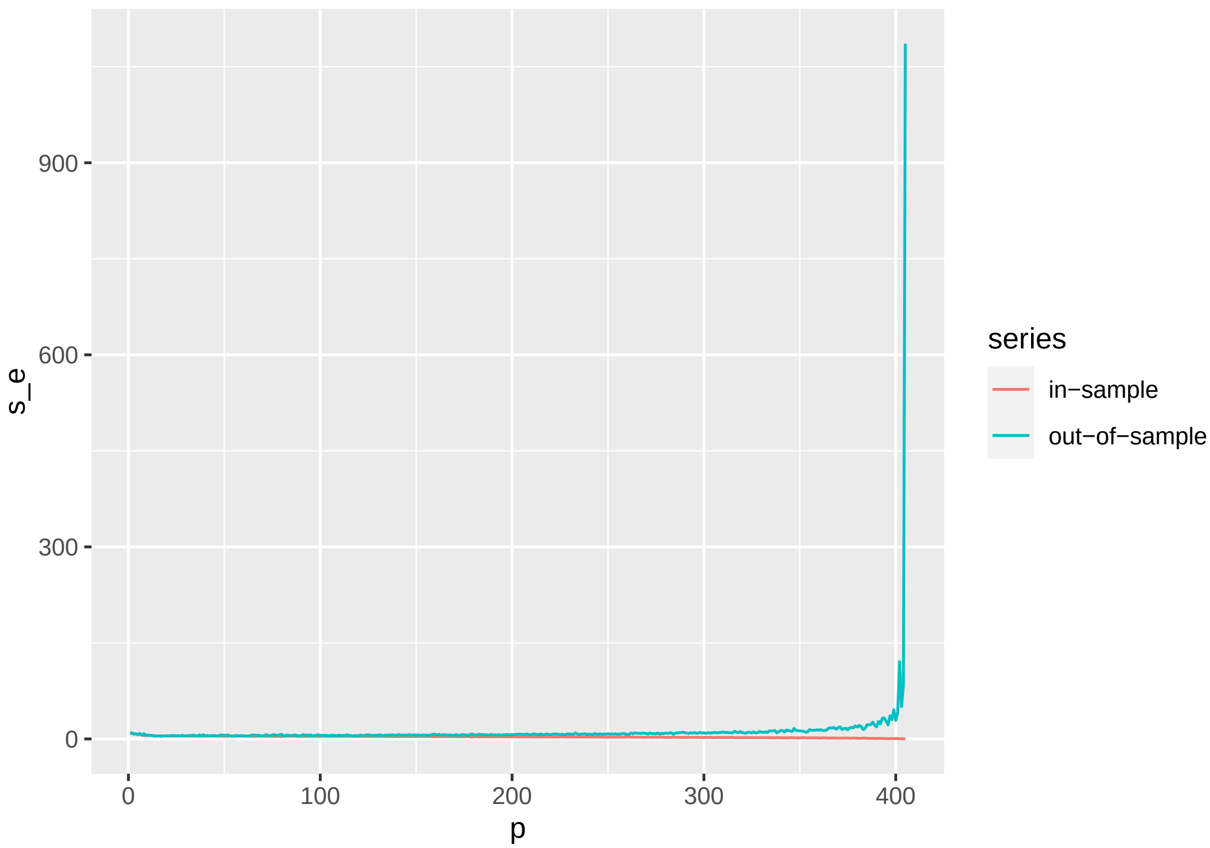
```
  diff_se [j]= s_e_by_p [j]-  ooss_e_by_p[j]

}
mean(diff_se)
```

```
## [1] -8.789525
```

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 = 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.

Yes. As you add more features you can better fit your in-sample model. This is artifical and really your model is more disconnected from reality. That is represented by your out of sample statistics.