Lab 5

Meihe Liu

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.

```
X <-matrix(1:1, nrow = 2, ncol = 2)
X[,2] = rnorm(2)

norm_vec = function(v){
  sqrt(sum(v^2))
}
cos_theta = t(X[,1]) %*% X[,2] / (norm_vec(X[,1]) * norm_vec(X[,2]))
acos(cos_theta)*180/pi
```

```
##          [,1]
## [1,] 153.8494
```

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] 44.9223
```

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] 44.974389 23.127778 15.359184  6.536125  4.599986  3.252277  2.042841
## [8]  1.447508
```

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

The absolute angle difference from 90 is converging to 0. This make sense b/c in a 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
X = cbind(1,rnorm(n))
y=rnorm(n)
H=X %*% solve(t(X) %*% X) %*% t(X)
yhat=H %*% y
ybar=mean(y)
SSR=sum((yhat-ybar)^2)
SST=sum((y-ybar)^2)
Rsq = (SSR/SST)
Rsq
```

```
## [1] 0.07474755
```

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)
  yhat=H %*% y
  ybar=mean(y)
  SSR=sum((yhat-ybar)^2)
  SST=sum((y-ybar)^2)
  Rsq_s[j] = (SSR/SST)
}
Rsq_s
```

```
##  [1] 0.07639760 0.07640032 0.08170874 0.08363622 0.08368172 0.08925738
##  [7] 0.11344572 0.13131039 0.14579344 0.15173161 0.17612588 0.20926158
## [13] 0.21179273 0.21419054 0.21433991 0.21641814 0.21703958 0.24631055
## [19] 0.24994160 0.25163540 0.27015836 0.28014941 0.28358590 0.38065719
## [25] 0.38529221 0.38620849 0.38624203 0.38782662 0.38801030 0.38931708
## [31] 0.43800240 0.44059893 0.44223994 0.44327634 0.44723600 0.44941722
```

```
## [37] 0.48487883 0.49477872 0.49816965 0.51233424 0.51412891 0.51788709
## [43] 0.51789003 0.52032307 0.52042566 0.52506309 0.52532784 0.53058607
## [49] 0.54499307 0.54691514 0.55090714 0.58221400 0.60280679 0.60952730
## [55] 0.60953814 0.61551188 0.62685630 0.62697433 0.63306089 0.64030520
## [61] 0.64612802 0.64670950 0.64706623 0.67786629 0.67805288 0.74093590
## [67] 0.80233749 0.80973984 0.81956491 0.81965843 0.82730816 0.83311408
## [73] 0.83388768 0.84349822 0.85606222 0.85607688 0.86009159 0.86553206
## [79] 0.86693608 0.86993630 0.91559005 0.92671466 0.93802362 0.94123727
## [85] 0.94268921 0.94398469 0.94427922 0.95284997 0.95294183 0.96152300
## [91] 0.97266433 0.97638368 0.97677765 0.97680186 0.97713114 0.98204856
## [97] 0.98376549 1.00000000
```

```
diff(Rsq_s)
```

```
##  [1] 2.726878e-06 5.308417e-03 1.927483e-03 4.550110e-05 5.575660e-03
##  [6] 2.418834e-02 1.786467e-02 1.448304e-02 5.938168e-03 2.439427e-02
## [11] 3.313570e-02 2.531154e-03 2.397803e-03 1.493710e-04 2.078231e-03
## [16] 6.214392e-04 2.927097e-02 3.631050e-03 1.693807e-03 1.852296e-02
## [21] 9.991052e-03 3.436491e-03 9.707129e-02 4.635019e-03 9.162821e-04
## [26] 3.353196e-05 1.584592e-03 1.836792e-04 1.306787e-03 4.868532e-02
## [31] 2.596524e-03 1.641008e-03 1.036401e-03 3.959665e-03 2.181222e-03
## [36] 3.546161e-02 9.899890e-03 3.390926e-03 1.416459e-02 1.794663e-03
## [41] 3.758181e-03 2.944172e-06 2.433039e-03 1.025906e-04 4.637434e-03
## [46] 2.647471e-04 5.258232e-03 1.440699e-02 1.922074e-03 3.991995e-03
## [51] 3.130687e-02 2.059279e-02 6.720504e-03 1.084640e-05 5.973731e-03
## [56] 1.134442e-02 1.180289e-04 6.086558e-03 7.244311e-03 5.822821e-03
## [61] 5.814827e-04 3.567296e-04 3.080006e-02 1.865845e-04 6.288302e-02
## [66] 6.140159e-02 7.402351e-03 9.825071e-03 9.352215e-05 7.649732e-03
## [71] 5.805916e-03 7.736009e-04 9.610541e-03 1.256400e-02 1.465814e-05
## [76] 4.014713e-03 5.440465e-03 1.404019e-03 3.000223e-03 4.565375e-02
## [81] 1.112460e-02 1.130897e-02 3.213649e-03 1.451934e-03 1.295484e-03
## [86] 2.945309e-04 8.570747e-03 9.186306e-05 8.581171e-03 1.114132e-02
## [91] 3.719348e-03 3.939755e-04 2.421297e-05 3.292735e-04 4.917423e-03
## [96] 1.716933e-03 1.623451e-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)
#H=X %*% solve(t(X) %*% X) %*% t(X)
#H[1:10,1:10]
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)
  yhat=H %*% y
```

```
  ybar=mean(y)
  SSR=sum((yhat-ybar)^2)
  SST=sum((y-ybar)^2)
  Rsq = (SSR/SST)
  Rsq
```

Why does this make sense? line 107 fails. rank dificient matrix is not inverible.

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
```

```
#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
```

```
#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
```

```
#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)
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?

It's expected to be different form 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.

```
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
  }
}
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?

{r}
```
Q_from_Rs_builtin = qr.Q(qr(X))
expect_equal(Q,Q_from_Rs_builtin)
```

Is this expected? Why did this happen?

Yes. many orthonormal basis of

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.

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

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
}
H = Q %*% solve(t(Q) %*% Q) %*% t(Q)
expect_equal(H %*% y, 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 = nrow(X)
n_test = round(n * 1 / K)
n_train = n - n_test

test_vec = sample(1:n,n_test)
train_vec = setdiff(1:n, test_vec)

X_train = X[train_vec, ]
y_train = y[train_vec]
X_test = X[test_vec, ]
y_test = y[test_vec]

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_mod = lm(y_train ~.+0, data.frame(X_train))
summary(ols_mod)$sigma #RMSE
```

```
## [1] 5.019707
```

```
sd(ols_mod$residuals) #insample
```

```
## [1] 4.938284
```

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

```
K = 5
Nsim = 1000
n_test = round(n * 1 /K)
n_train = n - n_test
oosSSE_arr = array(NA, dim = Nsim)
s_e_arr = array(NA, dim = Nsim)
RMSE_arr = array(NA, dim = Nsim)
for(i in 1:Nsim){
  test_vec = sample(1:n, 1/K*n)
  train_vec = setdiff(1:n, test_vec)

  X_train = X[train_vec, ]
  y_train = y[train_vec]
  X_test = X[test_vec, ]
  y_test = y[test_vec]

  mod = lm(y_train ~.+0, data.frame(X_train))
  yhat_test = predict(mod, data.frame(X_test))
  oosSSE_arr[i] = sd(y_test - yhat_test)
  s_e_arr[i] = sd(mod$residuals)
  RMSE_arr[i] = summary(mod)$sigma
}
```

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`.

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

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

```
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`.

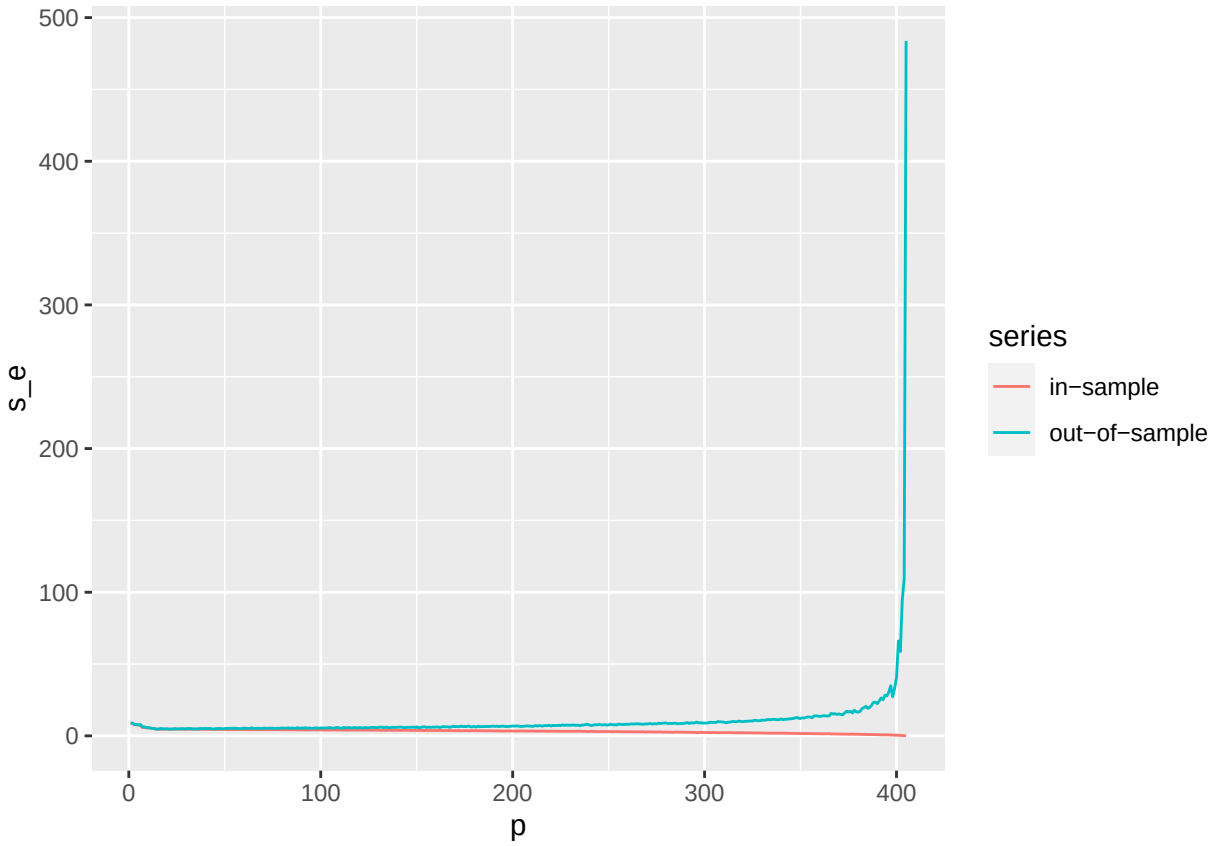
```
K = 5
Nsim = 10
n_test = round(n * 1 /K)
n_train = n - n_test
s_e_by_p = array(NA, dim = ncol(X_with_junk))
ooss_e_by_p = array(NA, dim = ncol(X_with_junk))
for(j in 1:ncol(X_with_junk)){
  oosSSE_arr = array(NA, dim = Nsim)
  s_e_arr = array(NA, dim = Nsim)
  for(i in 1:Nsim){
    test_vec = sample(1:n, 1/K*n)
    train_vec = setdiff(1:n, test_vec)

    X_train = X_with_junk[train_vec, 1:j , drop = FALSE]
    y_train = y[train_vec]
    X_test = X_with_junk[test_vec, 1:j , drop = FALSE]
    y_test = y[test_vec]

    mod = lm(y_train ~ .+0, data.frame(X_train))
    yhat_test = predict(mod, data.frame(X_test))
    oosSSE_arr[i] = sd(y_test - yhat_test)
    s_e_arr[i] = sd(mod$residuals)
  }
 s_e_by_p[j] = mean(s_e_arr)
 ooss_e_by_p[j] = mean(oosSSE_arr)
}
```

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. The model is overfitting so the out-of-sample error is exponentially growing as p is getting larger while the in-sample error is approaching zero as p is getting larger.