Lab 5

Amy Butler

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,] 0.7370102

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

##          [,1]
## [1,] 47.47735
```

#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] 45.14763
```

#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] 44.975187 23.206373 15.352776  6.512916  4.596174  3.254262  2.046538
## [8]  1.443217
```

#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 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)
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] 0.0006872124
```

#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] 0.0007389075 0.0060698005 0.0124403810 0.0156202066 0.0173855688
##  [6] 0.0174309724 0.0198342760 0.0439464716 0.0441725038 0.0442132082
## [11] 0.0669590225 0.0736459674 0.0865487111 0.0865558153 0.0871539189
## [16] 0.1585263306 0.1615464305 0.1684894368 0.2586822230 0.2735748259
## [21] 0.2740060139 0.2821029857 0.2949488743 0.2953542491 0.2963664553
## [26] 0.2964021237 0.2964853823 0.2984531653 0.3179491299 0.3180605332
## [31] 0.3318574173 0.3410024326 0.3423887451 0.3424038273 0.3467535305
## [36] 0.3622493473 0.3624735695 0.3630034965 0.3731853577 0.3749901323
## [41] 0.3754271601 0.3878920151 0.3960800544 0.4170297343 0.4172912501
## [46] 0.4225410067 0.4420243375 0.4420751456 0.4441167122 0.4475500025
## [51] 0.5254704342 0.5302888506 0.5307700046 0.5985103039 0.5985953201
## [56] 0.5986833179 0.6106209167 0.6157035436 0.6159066897 0.6188085355
## [61] 0.6224860466 0.6522302115 0.6726604738 0.6866731187 0.6866739608
## [66] 0.6886858756 0.6914217282 0.7168473184 0.7198930407 0.7391515964
## [71] 0.7391534880 0.7593055052 0.7781554545 0.7885030204 0.7928726143
## [76] 0.7946673467 0.7999369637 0.8040434041 0.8040632051 0.8460649367
## [81] 0.8506447594 0.8855042311 0.8907876074 0.8918577060 0.8936762775
## [86] 0.8957954296 0.8961222460 0.9203886513 0.9645464397 0.9684620063
## [91] 0.9767741647 0.9805354289 0.9842280686 0.9856367686 0.9920010926
## [96] 0.9934796406 0.9967833070 1.0000000000
```

```
diff(Rsq_s)
```

```
##  [1] 5.330893e-03 6.370580e-03 3.179826e-03 1.765362e-03 4.540365e-05
##  [6] 2.403304e-03 2.411220e-02 2.260321e-04 4.070448e-05 2.274581e-02
## [11] 6.686945e-03 1.290274e-02 7.104163e-06 5.981036e-04 7.137241e-02
## [16] 3.020100e-03 6.943006e-03 9.019279e-02 1.489260e-02 4.311880e-04
## [21] 8.096972e-03 1.284589e-02 4.053749e-04 1.012206e-03 3.566841e-05
## [26] 8.325859e-05 1.967783e-03 1.949596e-02 1.114033e-04 1.379688e-02
## [31] 9.145015e-03 1.386312e-03 1.508220e-05 4.349703e-03 1.549582e-02
## [36] 2.242222e-04 5.299270e-04 1.018186e-02 1.804775e-03 4.370278e-04
## [41] 1.246485e-02 8.188039e-03 2.094968e-02 2.615158e-04 5.249757e-03
## [46] 1.948333e-02 5.080814e-05 2.041567e-03 3.433290e-03 7.792043e-02
## [51] 4.818416e-03 4.811540e-04 6.774030e-02 8.501619e-05 8.799779e-05
## [56] 1.193760e-02 5.082627e-03 2.031462e-04 2.901846e-03 3.677511e-03
## [61] 2.974416e-02 2.043026e-02 1.401264e-02 8.421569e-07 2.011915e-03
## [66] 2.735853e-03 2.542559e-02 3.045722e-03 1.925856e-02 1.891583e-06
## [71] 2.015202e-02 1.884995e-02 1.034757e-02 4.369594e-03 1.794732e-03
## [76] 5.269617e-03 4.106440e-03 1.980101e-05 4.200173e-02 4.579823e-03
## [81] 3.485947e-02 5.283376e-03 1.070099e-03 1.818572e-03 2.119152e-03
## [86] 3.268165e-04 2.426641e-02 4.415779e-02 3.915567e-03 8.312158e-03
## [91] 3.761264e-03 3.692640e-03 1.408700e-03 6.364324e-03 1.478548e-03
## [96] 3.303666e-03 3.216693e-03
```

#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?

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

#Why does this make sense?

It fails because X-transpose-X is rank deficient making it impossible to invert.

#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: This makes sense.
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: The dot product is zero so the two vectors are perpendicular.
The parallel vector should be zero and the perpendicular vector should be
itself.
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

#prediction:
result$a_parallel + result$a_perpendicular

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

#prediction: tHIS should be reconstructed the original vector.
result$a_parallel / c(1, 3, 5 ,7)*37

##          [,1]
## [1,] 33.47619
## [2,] 33.47619
## [3,] 33.47619
## [4,] 33.47619

#prediction: The scalar.
```

#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`.

```
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 is expected to be different from 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]
  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)) expect_equal(Q, Q_from_Rs_builtin)

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

```
Q_from_Rs_builtin = qr.Q(qr(X))
```

#Is this expected? Why did this happen?

There are infinite orthonormal bases of any column space. The projection will be exactly the same.

#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.

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

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_test = round(n * 1 / K)
n_train = n - n_test
test_ind = sample(1:n, n_test)
train_ind= setdiff(1:n, test_ind)
X_test = X[test_ind,]
X_train = X[train_ind]
Y_test = y[test_ind]
Y_train = y[train_ind]
```

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))
sd(mod$residuals)

## [1] 9.2823

y_hat = predict(mod, data.frame(X_test))

## Warning: 'newdata' had 101 rows but variables found have 405 rows

e = Y_test - y_hat

## Warning in Y_test - y_hat: longer object length is not a multiple of
shorter
## object length

oos_SE = sd(e)
oos_SE

## [1] 8.751189
```

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

Nsim = 1000 sum = 0 for (i in 1:Nsim) { test_indices = sample(1 : n, n_test) train_indices = setdiff(1 : n, test_indices) X_train = X[train_indices,] y_train = y[train_indices] X_test = X[test_indices,] y_test = y[test_indices] ols_mod = lm(y_train ~ .+0, data.frame(X_train)) s_e = sd(ols_mod$residuals) y_oos = predict(ols_mod, data.frame(X_test)) residuals = y_test - y_oos ooss_e = sd(residuals) sum = sum + abs(s_e - ooss_e) } avg_diff = sum / Nsim avg_diff

#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`.

test_indices = sample(1 : n, n_test) train_indices = setdiff(1 : n, test_indices) s_e_by_p = rep(NA, ncol(X_with_junk)) ooss_e_by_p = rep(NA, ncol(X_with_junk)) sum_by_p = 0 oos_sum_by_p = 0 Nsim = 100 for (i in 1 : Nsim) { for (j in 1 : ncol(X_with_junk)) { X_train = X_with_junk[train_indices, 1 : j, drop = FALSE] y_train = y[train_indices] X_test = X_with_junk[test_indices, 1 : j, drop = FALSE] y_test = y[test_indices] in_mod = lm(y_train ~ .+0, data.frame(X_train)) oos_y = predict(in_mod, data.frame(X_test)) s_e_by_p[j] = sd(in_mod$residuals) ooss_e_by_p[j] = sd(y_test - oos_y) } sum_by_p = sum_by_p + sum(s_e_by_p) oos_sum_by_p = oos_sum_by_p + sum(ooss_e_by_p) } sum_by_p / (ncol(X_with_junk) * Nsim) oos_sum_by_p / (ncol(X_with_junk) * Nsim)

#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 this shape is expected because we added more features.