---

title: "Lab 4"

author: "Asher Katz"

output: pdf_document

date: 03/13/2022

---

Load up the famous iris dataset. We are going to do a different prediction problem. Imagine the only input x is Species and you are trying to predict y which is Petal.Length. A reasonable prediction is the average petal length within each Species. Prove that this is the OLS model by fitting an appropriate `lm` and then using the predict function to verify.

```{r}

data(iris)

#We are going to do a different prediction problem. Imagine the only input x is Species and you are trying to predict y which is Petal.Length. A reasonable prediction is the average petal length within each Species. Prove that this is the OLS model by fitting an appropriate `lm` and then using the predict function to verify.

x = iris$Species

y = iris$Petal.Length

AvgPL = lm(y~x)

table(x)

predict(AvgPL, newdata = data.frame(Species = c("setosa", "versicolor", "virginica") ) )# we're predicting for petal length

mean(iris$Petal.Length[iris$Species == "setosa"])

mean(iris$Petal.Length[iris$Species == "versicolor"])

mean(iris$Petal.Length[iris$Species == "virginica"])

```

Construct the design matrix with an intercept, X without using `model.matrix`.

```{r}

X = cbind(1,iris$Species == "setosa", iris$Species == "versicolor") # those are BOOL so we get a 1 if true. We are putting setosa in the 2nd col as the reference variable

```

Find the hat matrix H for this regression.

```{r}

H = X %*% (solve( t(X )%*% X )) %*% t(X) # solve() takes the inverse

```

Verify this hat matrix is symmetric using the `expect_equal` function in the package `testthat`.

```{r}

pacman::p_load(testthat)

expect_equal( t(H), H)

```



Verify this hat matrix is idempotent using the `expect_equal` function in the package `testthat`.



```{r}

expect_equal(H, H%*%H)

# H^2 = H

```



Using the `diag` function, find the trace of the hat matrix.



```{r}

sum(diag(H)) # 3 cause its

#the trace of an orthogonal projection matrix is its rank

```



It turns out the trace of a hat matrix is the same as its rank! But we don't have time to prove these interesting and useful facts..

Using the hat matrix, compute the yhat vector and using the projection onto the residual space, compute the e vector and verify they are orthogonal to each other.



```{r}

yhat = H %*% y

evec = y - yhat

e = (diag(nrow(iris)) - H) %*% y

```



Compute SST, SSR and SSE and R^2 and then show that SST = SSR + SSE.



```{r}

ybar = mean(y)

SST = sum( (y - ybar)^2 ) #t(y-ybar) * (y-ybar)

SSR = sum( (yhat - ybar)^2 )#t(yhat-ybar) * (yhat-ybar)

SSE = sum( (evec)^2 )  #t(e) * e = SSE

Rsq = SSR/SST



expect_equal(SSR + SSE, SST)
```

```

Find the angle theta between y - ybar 1 and yhat - ybar 1 and then verify that its cosine squared is the same as the R^2 from the previous problem.

```{r}
u = y - ybar

v = yhat - ybar

norm = function(d){

  sqrt( sum(d^2) )

}

theta = acos( norm( t(u)%*%v ) /( norm(SST)*norm(SSR) ) )

#angle between two vectors
```

Project the y vector onto each column of the X matrix and test if the sum of these projections is the same as yhat.

```{r}
proj1 = ((X[,1] %*% t(X[,1])) / as.numeric(t(X[,1]) %*% X[,1])) %*% y

proj2 = ((X[,2] %*% t(X[,2])) / as.numeric(t(X[,2]) %*% X[,2])) %*% y

proj3 = ((X[,3] %*% t(X[,3])) / as.numeric(t(X[,3]) %*% X[,3])) %*% y

expect_equal(proj1 + proj2 + proj3, yhat)

```

Construct the design matrix without an intercept, X, without using `model.matrix`.

```{r}
X = cbind(iris$Species == "virginica", iris$Species == "setosa", iris$Species == "versicolor") # those are BOOL so we get a 1 if true

colSums(X)
```

Find the OLS estimates using this design matrix. It should be the sample averages of the petal lengths within species.

```{r}
b = (solve( t(X) %*% X )) %*% t(X) %*% y #probably a y issue

X_model = lm(y ~ X)
```
```

```

Verify the hat matrix constructed from this design matrix is the same as the hat matrix constructed from the design matrix with the intercept. (Fact: orthogonal projection matrices are unique).

```{r}
H_prime = X%*%(solve( t(X) %*% X) %*%t(X) )
expect_equal(H_prime, H)
```

```

Project the y vector onto each column of the X matrix and test if the sum of these projections is the same as yhat.

```{r}

proj1 = ((X[,1] %*% t(X[,1])) / as.numeric(t(X[,1]) %*% X[,1])) %*% y
proj2 = ((X[,2] %*% t(X[,2])) / as.numeric(t(X[,2]) %*% X[,2])) %*% y
proj3 = ((X[,3] %*% t(X[,3])) / as.numeric(t(X[,3]) %*% X[,3])) %*% y
expect_equal(proj1 + proj2 + proj3, yhat)

```

Convert this design matrix into Q, an orthonormal matrix.

```{r}
#v1vec = x1vec
#v2vec= x2vec - projv1vec(x2vec)
#v3vec = x3vec - projv1vec(x3vec) - projv2vec(x3vec)
#q1vec = v1vec/ ||v1vec||
#q2vec= v2vec/ ||v2vec||
#q3vec= v3vec/ ||v3vec||
v_1 = X[,1]
v_2 = X[,2] - (v_1%*% t(v_1) / norm(v_1)) %*% X[,2]
v_3 = X[,3]- (v_1%*% t(v_1) / norm(v_1)) %*% X[,3]- (v_2%*% t(v_2) / norm(v_2)) %*% X[,3]
q_1 = v_1 /norm(v_1)
q_2 = v_2 /norm(v_2)
q_3 = v_3 /norm(v_3)
Q = cbind(q_1, q_2, q_3)
```

```

Project the y vector onto each column of the Q matrix and test if the sum of these projections is the same as yhat.

```{r}

proj1 = ((Q[,1] %*% t(Q[,1])) / as.numeric(t(Q[,1]) %*% Q[,1])) %*% y

proj2 = ((Q[,2] %*% t(Q[,2])) / as.numeric(t(Q[,2]) %*% Q[,2])) %*% y

proj3 = ((Q[,3] %*% t(Q[,3])) / as.numeric(t(Q[,3]) %*% Q[,3])) %*% y

expect_equal(proj1 + proj2 + proj3, yhat)

```

Find the p=3 linear OLS estimates if Q is used as the design matrix using the `lm` method. Is the OLS solution the same as the OLS solution for X?

```{r}

Q_model = lm(y ~ Q, iris) # not the same

Q_model

```

Use the predict function and ensure that the predicted values are the same for both linear models: the one created with X  as its design matrix and the one created with Q as its design matrix.

```{r}

predict(X_model)

predict(Q_model)

expect_equal(predict(X_model), predict(Q_model))

```

Clear the workspace and load the Boston housing data and extract X and y. The dimensions are n = 506 and p = 13. Create a matrix that is (p + 1) x (p + 1) full of NA's. Label the columns the same columns as X. Do not label the rows. For the first row, find the OLS estimate of the y regressed on the first column only and put that in the first entry. For the second row, find the OLS estimates of the y regressed on the first and second columns of X only and put them in the first and second entries. For the third row, find the OLS estimates of the y regressed on the first, second and third columns of X only and put them in the first, second and third entries, etc. For the last row, fill it with the full OLS estimates.

```{r}

rm(list = ls()) #Clear the workspace

pacman::p_load(MASS)

data(Boston) #load the Boston housing data

#and extract X and y

X = cbind(1, as.matrix(Boston[,1:13]))
```
```

```
y = Boston[,14]



p_plus1 = ncol(X)

matrix_p_plus1 = matrix(NA, nrow = p_plus1, ncol = p_plus1)# Create a matrix that is (p + 1) x (p + 1)



colnames(matrix_p_plus1) = c(colnames(Boston[1:13]), "full OLS") #Label the columns the same columns as X.



  #For the first row, find the OLS estimate of the y regressed on the first column only and put that in the first entry.

#OLS NA NA NA NA...

  #For the second row, find the OLS estimates of the y regressed on the first and second columns of X only and put them in the first and second entries

#OLS OLS NA NA NA NA ...

# For the third row, find the OLS estimates of the y regressed on the first, second and third columns of X only and put them in the first, second and third entries,

#OLS OLS OLS NA NA NA NA NA ...etc.



for (i in 1:ncol(X)) {

  X_i = X[,1:i]

  matrix_p_plus1[i,1:i] = solve( t(X_i) %*% X_i ) %*% t(X_i) %*% y # creates a vector on row i with the proper X col entries

}

View(matrix_p_plus1)

```



Why are the estimates changing from row to row as you add in more predictors?



the estimates are changing from row to row because simply because each row is adding more features. The model adjusts it's predictions based on this new information.



Create a vector of length p+1 and compute the R^2 values for each of the above models.



```{r}

Rsq_vec = c(1:14)



for (i in 1:ncol(X)) {

  mod = lm(y ~ X[, 1:i])

  Rsq_vec[i] = summary(mod)$r.squared

}

Rsq_vec

```



Is R^2 monotonically increasing? Why?

R^2 is monotonically increasing because as the model predicts based on more features, it makes sense that the model will get better at explaining the variance.

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.

```{r}
n = 1
X = as.matrix(cbind(1, rnorm(2*n)))#Create a 2x2 matrix with the first column 1's and the next column iid normals
AbsAng = acos(t(X[,1]) %*% X[,2] / sqrt(sum(X[, 1]^2) * sum(X[, 2]^2))) * 180 / pi
X
AbsAng
```

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

```{r}
Nsim = 1e5
n = 1
AbsAngs = list()
for(i in 1:Nsim){
X = as.matrix(cbind(1, rnorm(2*n)))#Create a 2x2 matrix with the first column 1's and the next column iid normals
AbsAng = acos(t(X[,1]) %*% X[,2] / sqrt(sum(X[, 1]^2) * sum(X[, 2]^2))) * 180 / pi
AbsAngs[i] = AbsAng
}
mean(as.numeric(AbsAngs))
```

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.

```{r}
Nsim = 1e5
n = c(10, 50, 100, 200, 1000)
AbsAngs = list()
for (j in n){
for(i in 1:Nsim){
```

```
X = as.matrix(cbind(1, rnorm(2*j)))#Create a 2x2 matrix with the first column 1's and the next column iid normals

AbsAng = acos(t(X[,1]) %*% X[,2] / sqrt(sum(X[, 1]^2) * sum(X[, 2]^2))) * 180 / pi

AbsAngs[i] = AbsAng

}

print(mean(as.numeric(AbsAngs)))

}
```

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

It's converging towards 90 degrees. This makes sense because we're caluculating absolute difference from 90 degrees and the more data we have, the norms will cancel each other out, less is subtracted or added to 90 degrees.