Lab 6

Sakib Salim

11:59PM March 24, 2019

Load the Boston Housing data and create the vector y and the design matrix X.

```
data(Boston, package = "MASS")
y = Boston$medv
X = rep(1, nrow(Boston))
X = as.matrix(cbind(X, Boston[, 1 : 13]))
```

Find the OLS estimate and OLS predictions without using `lm`.

```
b = solve(t(X) %*% X) %*% t(X) %*% y
yhat = X %*% b
```

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){
  a_parallel = (v %*% t(v) %*% a) / ( sum(v^2) )
  a_perpendicular = a - a_parallel
  list("a_parallel" = a_parallel, "a_perpendicular" = a_perpendicular)
}
```

Try to project onto the column space of X by projecting y on each vector of X individually and adding up the projections. You can use the function `orthogonal_projection`.

```
sumOrthProj = rep(0, nrow(X))
for (j in 1 : ncol(X)){
  sumOrthProj = sumOrthProj + orthogonal_projection(y, X[, j])$a_parallel
  }
head(sumOrthProj)
```

```
##          [,1]
## [1,] 177.3425
## [2,] 185.6013
## [3,] 177.7175
## [4,] 171.7247
## [5,] 177.3255
## [6,] 175.5639
```

How much double counting occurred? Measure the magnitude relative to the true LS orthogonal projection.

```
doublecount = sumOrthProj / yhat
head(doublecount)
```

```
##       [,1]
## 1 5.910661
## 2 7.416470
## 3 5.813919
## 4 6.002884
## 5 6.345853
## 6 6.951296
```

Convert X into Q where Q has the same column space as X but has orthogonal columns. You can use the function `orthogonal_projection`. This is essentially gram-schmidt.

```
Q = matrix(NA, nrow = nrow(X), ncol = ncol(X))
Q[, 1] = X[, 1]
for(j in 2 : ncol(X)){
  Q[ , j] = X[ , j]

  for(j0 in 1 : (j - 1)){
    Q[ , j] = Q[ , j] - (orthogonal_projection(X[ , j], Q[ , j0])$a_parallel)
  }
}
pacman::p_load(Matrix)
rankMatrix(Q)[1]
```

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

Make Q's columns orthonormal.

```
for (j in 1 : ncol(Q)){
  Q[ ,j] = Q[ , j] / sqrt(sum(Q[ ,j]^2))
}
```

Verify Q^T is the inverse of Q.

```
#TO-DO
head(t(Q) %*% Q)
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.000000e+00 -1.170938e-16  7.329207e-17 -3.932090e-15  3.044440e-16
## [2,] -1.170938e-16  1.000000e+00  1.566672e-17  6.763727e-17  4.510281e-17
## [3,]  7.329207e-17  1.566672e-17  1.000000e+00 -5.826231e-17  3.794708e-19
## [4,] -3.932090e-15  6.763727e-17 -5.826231e-17  1.000000e+00  1.051744e-15
## [5,]  3.044440e-16  4.510281e-17  3.794708e-19  1.051744e-15  1.000000e+00
## [6,]  7.548107e-15  6.550750e-16  5.526721e-17  3.046028e-14 -2.882202e-15
##               [,6]          [,7]          [,8]          [,9]         [,10]
## [1,]  7.548107e-15 -1.379756e-14 -2.475017e-15 -1.269384e-15  1.098514e-15
## [2,]  6.550750e-16  1.082847e-15 -7.361733e-17  7.773730e-16 -2.138047e-16
## [3,]  5.526721e-17 -2.208520e-16  5.084908e-17  2.385245e-18 -9.540979e-18
## [4,]  3.046028e-14  3.826098e-14 -2.164291e-15  3.891581e-14 -6.627464e-15
## [5,] -2.882202e-15 -2.479679e-15 -1.329232e-16 -2.511229e-15  3.783866e-17
## [6,]  1.000000e+00 -6.696465e-14 -4.081119e-14 -3.385638e-14 -2.339584e-14
##              [,11]         [,12]         [,13]         [,14]
## [1,]  1.463239e-15 -1.382228e-14  1.006416e-14 -6.628812e-15
## [2,]  7.455516e-16  1.229485e-15  2.636644e-16  8.515324e-16
## [3,]  4.065758e-17  2.602085e-17 -5.095750e-17 -1.021318e-16
## [4,]  2.017742e-14  6.014552e-14  2.289555e-14  2.996148e-14
## [5,] -2.035237e-15 -4.422678e-15 -1.515213e-15 -2.182933e-15
## [6,] -6.567132e-14 -8.574749e-14 -3.213684e-14 -6.870822e-14
```

Project Y onto Q and verify it is the same as the OLS fit.

```
head(cbind(Q %*% t(Q) %*% y, yhat))
```

```
##       [,1]     [,2]
## 1 30.00384 30.00384
## 2 25.02556 25.02556
## 3 30.56760 30.56760
## 4 28.60704 28.60704
## 5 27.94352 27.94352
## 6 25.25628 25.25628
```

Project Y onto the columns of Q one by one and verify it sums to be the projection onto the whole space.

```
#TO-DO
proj_col_Q = rep(0,ncol(Q))
for (j in 1 : ncol(Q)){
  proj_col_Q = proj_col_Q + orthogonal_projection(y, Q[, j])$a_parallel
}
```

```
## Warning in proj_col_Q + orthogonal_projection(y, Q[, j])$a_parallel: longer
## object length is not a multiple of shorter object length
```

```
proj_Q = Q %*% t(Q) %*% y
pacman::p_load(testthat)
expect_equal(proj_col_Q, proj_Q)
```

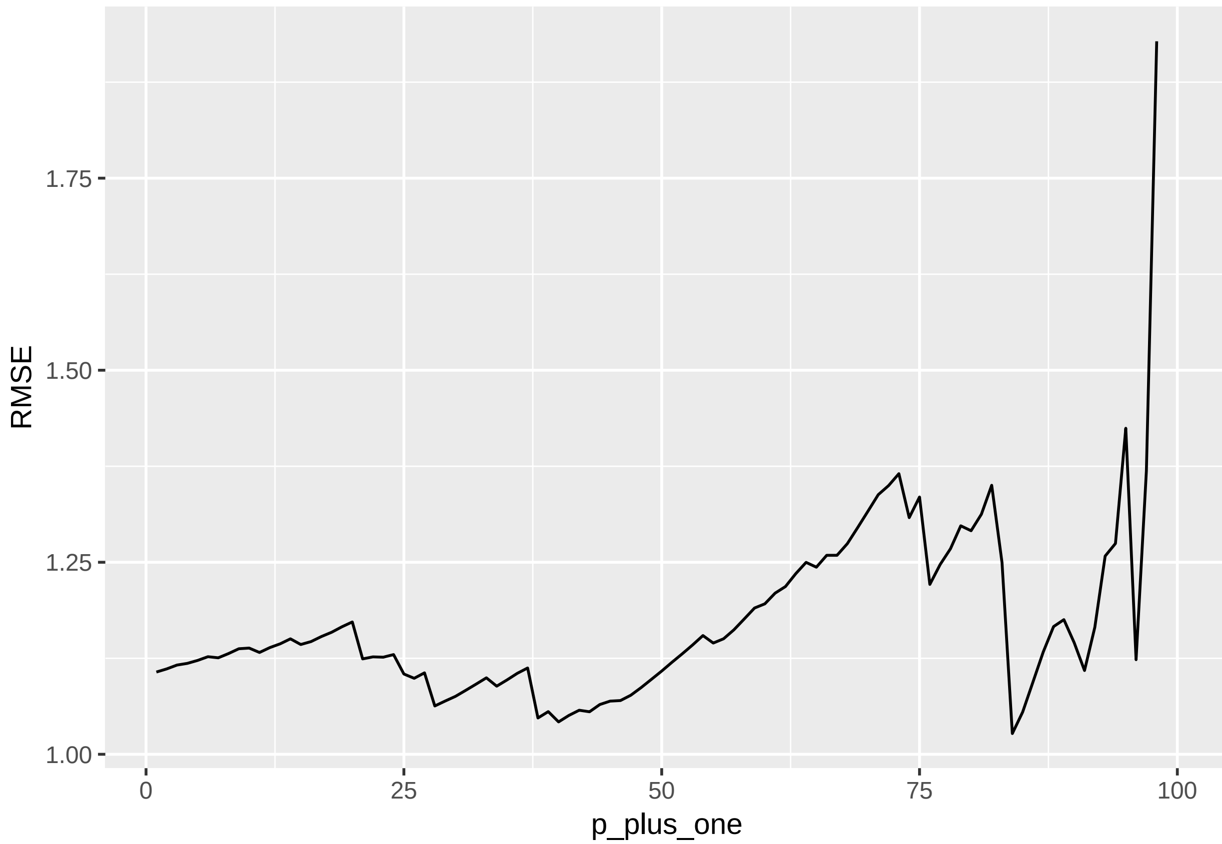
Verify the OLS fit squared length is the sum of squared lengths of each of the orthogonal projections.

```
#TO-DO
sum_sq_length_col_Q = 0
for (j in 1 : ncol(Q)){
  col_proj = orthogonal_projection(y, Q[, j])$a_parallel
  sum_sq_length_col_Q = sum_sq_length_col_Q + sum(col_proj^2)
}
OLS_sq_length = sum(proj_Q^2)
expect_equal(sum_sq_length_col_Q, OLS_sq_length)
```

Rewrite the "The monotonicity of SSR" demo from the lec06 notes but instead do it for RMSE. Comment every line in detail. Write about what the plots means.

```
#TO-DO
n = 100
y = rnorm(n)
Rmses = array(NA, n) #array to store RMSE values

#Create data matrix
X = matrix(NA, nrow = n, ncol = 0)

#for every new p, add a random column
for (p_plus_one in 1 : n){          #loop through eventual columns of X
  X = cbind(X, rnorm(n))            #add a random column
  Rmses[p_plus_one] = summary(lm(y ~ X))$sigma  #find RMSE
}
pacman::p_load(ggplot2)
base = ggplot(data.frame(p_plus_one = 1 : n, RMSE = Rmses))
base + geom_line(aes(x = p_plus_one, y = RMSE))
```

```
## Warning: Removed 2 rows containing missing values (geom_path).
```

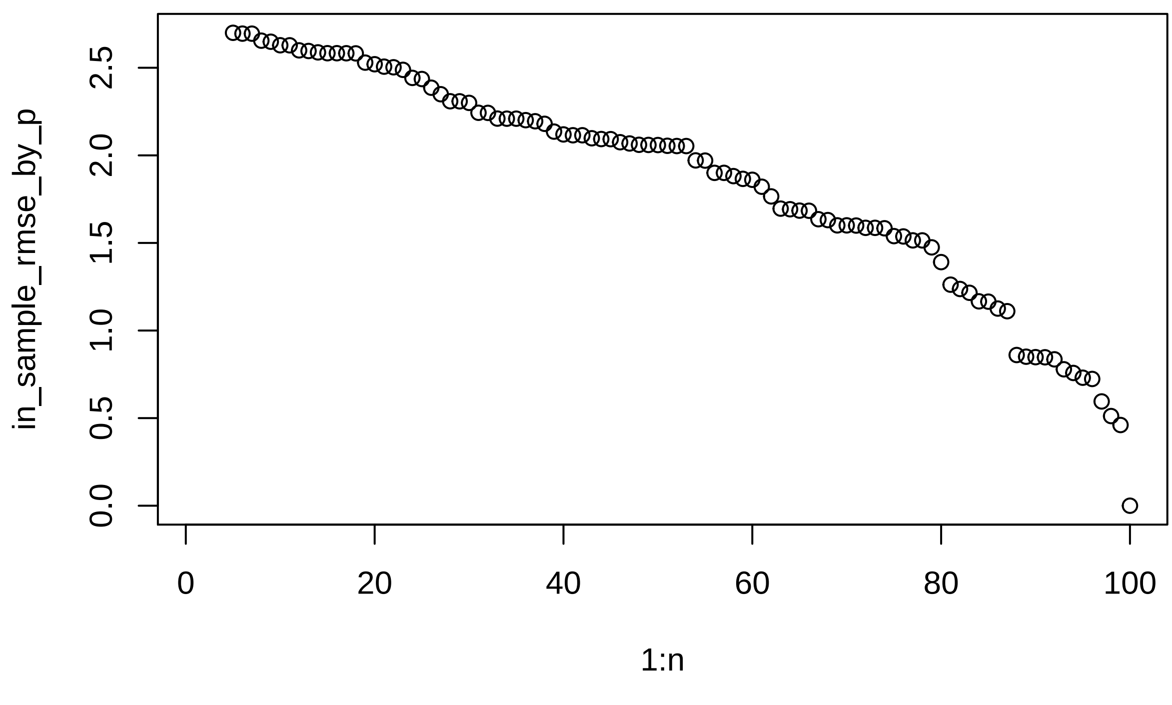



```
#The RMSE value decreases as we add columns to our data.
#Eventually R^2 approaches 1 which in turn makes RMSE = 0.
```

Rewrite the "Overfitting" demo from the lec06 notes. Comment every line in detail. Write about what the plots means.

```
#TO-DO
bbeta = c(1, 2, 3, 4)    #the real betas

#build training data
n = 100
X = cbind(1, rnorm(n), rnorm(n), rnorm(n)) #Intercept plus 3 random columns
y = X %*% bbeta + rnorm(n, 0, 0.3) #ma

#build test data
n_star = 100
X_star = cbind(1, rnorm(n), rnorm(n), rnorm(n_star))
y_star = X_star %*% bbeta + rnorm(n, 0, 0.3)

#store the betas each time you model on
#a design matrix with p+1 columns
all_betas = matrix(NA, n, n)
all_betas[4, 1 : 4] = coef(lm(y ~ 0 + X)) #fourth row of beta matrix are the
#beta values from when we had 4 columns in X
in_sample_rmse_by_p = array(NA, n)        #Store In Sample RMSE
```

```r
for (j in 5 : n){
  X = cbind(X, rnorm(n))                   #add a random column to X
  lm_mod = lm(y ~ 0 + X)                   #new linear model w/o intercept

  all_betas[j, 1 : j] = coef(lm_mod)       #add betas to the beta matrix in the
  #appropriate column (for j columns add them to the jth column of beta matrix)

  y_hat = X %*% all_betas[j, 1 : j]                     #produce a prediction our
  #new model(using new betas)

  in_sample_rmse_by_p[j] = sqrt(sum((y - y_hat)^2))   #calculate RMSE for the
  #prediction of "each new linear model"
}
plot(1 : n, in_sample_rmse_by_p)
```

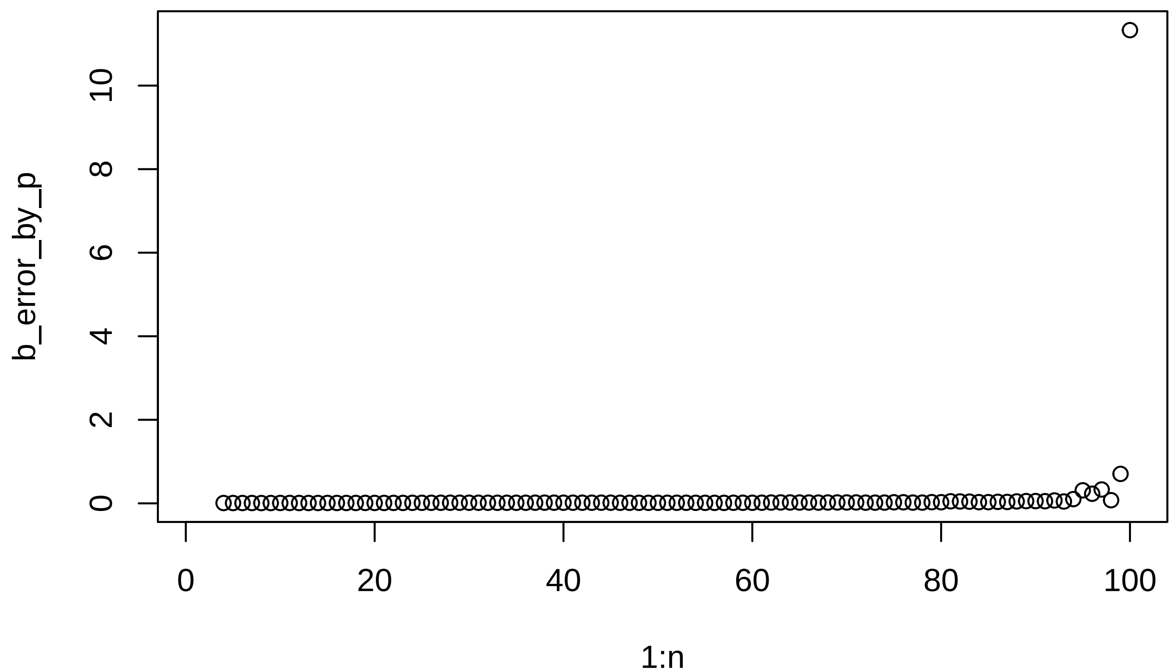



```r
#RMSE decreases as the columns of
#our design matrix approach n
#not honest, since we are just testing on given data,
#so eventually our model passes through every point in our data

head(all_betas[4 : n, 1 : 4]) #take the first four betas for each of our models
```

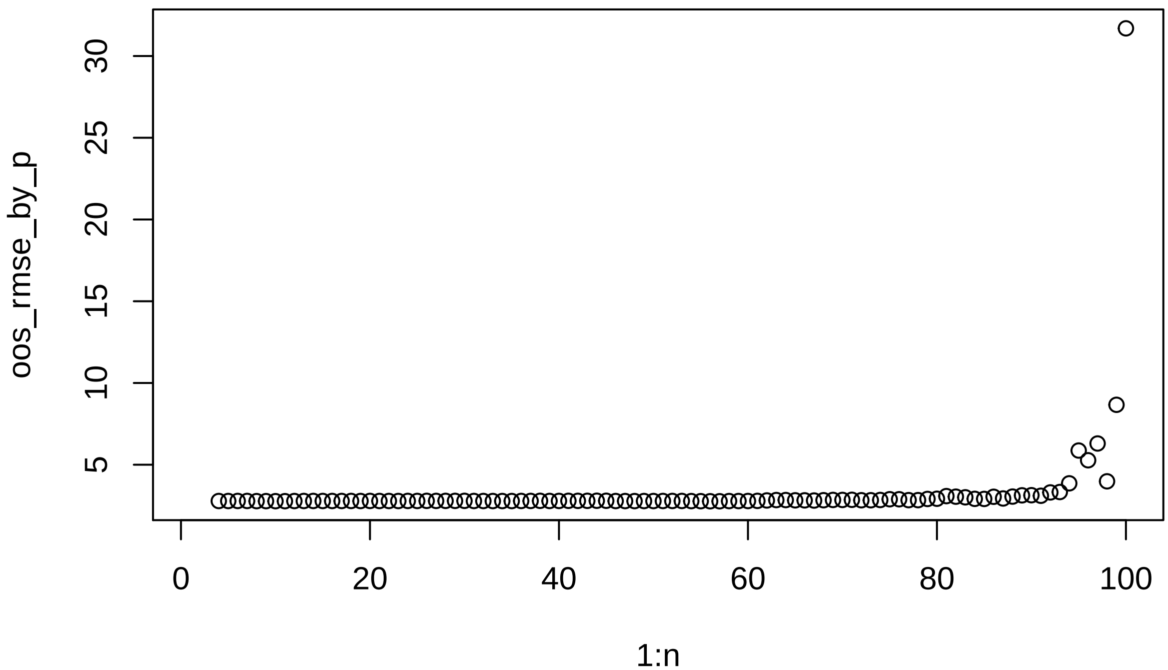
```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 1.022091 1.979610 3.002089 3.918065
## [2,] 1.022689 1.979520 3.002909 3.917379
## [3,] 1.022429 1.978806 3.005931 3.915418
```

```
## [4,] 1.022409 1.978495 3.006211 3.915244
## [5,] 1.025181 1.983374 3.002062 3.919133
## [6,] 1.026859 1.982744 3.002184 3.917314
```

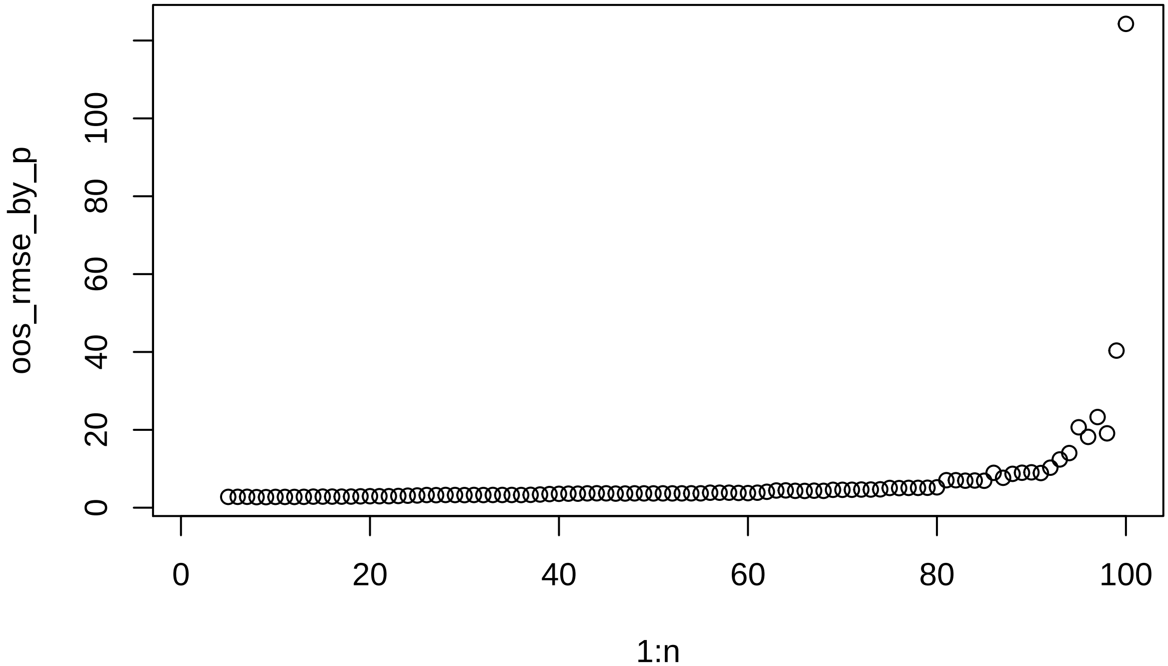
```r
b_error_by_p = rowSums((all_betas[, 1 : 4] - matrix(rep(bbeta, n), nrow = n, byrow = TRUE))^2)

#compare the model betas to the real betas
plot(1 : n, b_error_by_p)
```




```r
#our betas degenerate as the number of columns approaches n

#look at out of sample error in the case of only the first four features
oos_rmse_by_p = array(NA, n)    #store oosRMSE
for (j in 4 : n){
  y_hat_star = X_star %*% all_betas[j, 1 : 4]
  #predict on the data using the betas from "first four columns of Xstar"
  oos_rmse_by_p[j] = sqrt(sum((y_star - y_hat_star)^2)) #calculate RMSE
}
plot(1 : n, oos_rmse_by_p)
```




```
#look at out of sample error
#in the case of the random features too
oos_rmse_by_p = array(NA, n)
for (j in 5 : n){
  X_star = cbind(X_star, rnorm(n))              #again add a random column
  y_hat_star = X_star %*% all_betas[j, 1 : j] #prediction
  oos_rmse_by_p[j] = sqrt(sum((y_star - y_hat_star)^2)) #oosRMSE for when you add random columns
}
plot(1 : n, oos_rmse_by_p)
```




```
#oosRMSE actually tests how well our model
#works on data outside of the given data
#the predicting power of our model
#significantly decreases as we increase the columns of X
```