STAA 577 HW1

Enqun Wang

March 21, 2016

Enter the following matrix into R

```
set.seed(2016)
A = matrix(rnorm(4*3), nrow=4, ncol=3)
```

1. R question: Suppose we want to get the column mean for each column of the matrix A. Do this with

(a) Hard coding (that is, write $((A[1,1] + A[2,1] + ...)/4, ...)$

```
mean.col1 = (A[1, 1] + A[2, 1] + A[3, 1] + A[4, 1])/4
mean.col2 = (A[1, 2] + A[2, 2] + A[3, 2] + A[4, 2])/4
mean.col3 = (A[1, 3] + A[2, 3] + A[3, 3] + A[4, 3])/4

mean.col = data.frame(col = 1:3, mean = c(mean.col1, mean.col2, mean.col3))

require(knitr)
kable(mean.col, align = "l")
```

col	mean
1	0.0816821
2	-1.1306825
3	-0.0408484

(b) For loop(s)

```
mean.col = c()
for(i in 1:3){
  S = 0
  for(j in 1:4){
    S = S + A[j, i]
  }
  mean.col[i] = S/4
}
kable(data.frame(col = 1:3, mean = mean.col), align = "l")
```

col	mean
1	0.0816821
2	-1.1306825
3	-0.0408484

(c) The apply (or related) function

```r
kable(data.frame(col = 1:3, mean = apply(A, 2, mean)), align = "l")
```

col	mean
1	0.0816821
2	-1.1306825
3	-0.0408484

Note: the intention of this question is just to make sure everyone has these key tools in their R toolbox

---

2. Many statistical methods can be computed/analyzed using the SVD. Let's look at solving least squares problems as they are fundamental to modern data analysis.

(a) We will first explore the *overdetermined* case using the above A. Write $A = UDV^T$ (that is, form `svd.out = svd(A)`). Type `svd.out` into the `R` interpreter. What does it produce?

```r
svd.out = svd(A)
svd.out
```

```
## $d
## [1] 3.0903144 1.1602891 0.6265673
##
## $u
##              [,1]       [,2]       [,3]
## [1,] -0.955298367  0.1219184  0.2693087
## [2,] -0.004938162 -0.8860204  0.3871030
## [3,] -0.233353526 -0.1536838 -0.7499325
## [4,] -0.181457368 -0.4201013 -0.4639238
##
## $v
##             [,1]        [,2]       [,3]
## [1,]  0.26801322 -0.96062232 0.07330531
## [2,]  0.96124568  0.27173799 0.04653180
## [3,] -0.06461933  0.05799328 0.99622343
```

Suppose we wish to solve for $\hat{x} = argmin_x \|Ax - b\|_2^2 = (A^TA)^{-1}A^Tb$ for $b = (1, 1, 1, 1)^T$

As an aside, to show this, note that

$$\begin{aligned}
\|Ax - b\|_2^2 &= x^TA^TAx + b^Tb - 2x^TA^Tb \\
\Rightarrow \nabla_x &= 2A^TA\hat{x} - 2A^Tb \doteq 0 \\
\Rightarrow \hat{x} &= (A^TA)^{-1}A^Tb
\end{aligned}$$

How can I solve this using the SVD? Here, let's follow the steps:

i. Form U^Tb
ii. Solve $Dw = U^Tb$
iii. Form $\hat{x} = Vw$

Produce this $\hat{x}$ in R via this method. Note that in this particular case, all the singular values in D are nonzero and hence $\hat{x} = VD^{-1}U^Tb$.

```
b = matrix(c(1, 1, 1, 1), byrow = F)
U = t(svd.out$u)
D = diag(svd.out$d)
V = svd.out$v

Utb = U %*% b
w = solve(D) %*% Utb
x.hat = V %*% w
x.hat
```

```
##            [,1]
## [1,]  0.9231868
## [2,] -0.7824396
## [3,] -0.9244375
```

(b) Suppose instead we have observations under the model $Y = \mathbb{X}\beta + \epsilon$, where $Y = b$ and $\mathbb{X} = A$. Using the R function `lm` and `predict`, what is the least squares solution $\hat{\beta}$ and the fitted values $\hat{Y}$ for Y using the least squares solution?(Remember to not use an intercept)

```
Y = b; X = A

fit = lm(Y ~ 0 + X)
fit$coefficients # beta.hat
```

```
##         X1         X2         X3
##  0.9231868 -0.7824396 -0.9244375
```

```
predict(fit) # Y.hat
```

```
##         1         2         3         4
## 1.0003428 0.9763970 0.9445295 1.0701723
```

Derive the formula for $\hat{Y} = \mathbb{X}\hat{\beta}$ in terms of the SVD and the response Y.

$$\begin{gathered}
\hat{Y} = \mathbb{X}\hat{\beta} \\
\Rightarrow \hat{Y} = A(A^TA)^{-1}A^TY \\
\Rightarrow \hat{Y} = UDV^T((UDV^T)^TUDV^T)^{-1}(UDV^T)^TY \\
\Rightarrow \hat{Y} = Y
\end{gathered}$$

How does the produced coefficient vector $\hat{\beta}$ compare the $\hat{x}$?

- The produced coefficient vector $\hat{\beta}$ is the same as $\hat{x}$.

---

3. Now, let's look at a new A

```r
set.seed(2016)
A = matrix(rnorm(4*3), ncol=4, nrow=3)
```

and $b = (1,1,1)^T$. This is an example of an *underdetermined* system.

(a) What do(es) the corresponding $\hat{x}$ look like using the SVD? What do(es) the $\hat{\beta}$ look like using `lm`?

```r
svd.out = svd(A)
b = matrix(c(1, 1, 1), byrow = F)
U = t(svd.out$u)
D = diag(svd.out$d)
V = svd.out$v

Utb = U %*% b
w = solve(D) %*% Utb
x.hat = V %*% w
x.hat
```

```
##            [,1]
## [1,] -2.1801655
## [2,] -1.0839903
## [3,]  0.5752772
## [4,] -1.2759721
```

```r
Y = b; X = A
fit = lm(Y ~ 0 + X)
fit$coefficients # beta.hat
```

```
##        X1        X2        X3        X4
## -2.546250 -1.550859  1.138306        NA
```

(b) What do(es) the corresponding $A\hat{x}$ look like using the SVD? What do(es) the $\hat{Y} = \mathbb{X}\hat{\beta}$ look like using `predict`?

```r
A %*% x.hat
```

```
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    1
```

```r
predict(fit) # Y.hat
```

```
## 1 2 3
## 1 1 1
```

(c) Though this is just one simulated example and not a proof, your findings generalize to all situations when $p > n$. Summarize in words what these findings are.

- In the high dimensional regime ($p > n$), the solutions of $Y = \mathbb{X}\hat{\beta}$ are not unique, and it has an infinite number of solutions.

---

4. We will discuss more about R's memory, but suffice it to say that it has about 2 Gb of workable space. This can readily be exhausted. A tool for fitting least squares with R under memory constraints is `biglm`. For this problem, we will generate an object that is still able to be placed in memory, but this is for comparison's sake only. Do the following:

```
n = 2000
p = 500
set.seed(2016)
X = matrix(rnorm(n*p), nrow=n, ncol=p)

format(object.size(X),units='auto') #memory used by X
```

```
## [1] "7.6 Mb"
```

```
Xdf = data.frame(X)

Y = X %*% rnorm(p) + rnorm(n)

write.table(X[1:500,],file='Xchunk1.txt',sep=',',row.names=F,col.names=names(Xdf))
write.table(X[501:1000,],file='Xchunk2.txt',sep=',',row.names=F,col.names=names(Xdf))
write.table(X[1001:1500,],file='Xchunk3.txt',sep=',',row.names=F,col.names=names(Xdf))
write.table(X[1501:2000,],file='Xchunk4.txt',sep=',',row.names=F,col.names=names(Xdf))

write.table(Y[1:500],file='Ychunk1.txt',sep=',',row.names=F,col.names=F)
write.table(Y[501:1000],file='Ychunk2.txt',sep=',',row.names=F,col.names=F)
write.table(Y[1001:1500],file='Ychunk3.txt',sep=',',row.names=F,col.names=F)
write.table(Y[1501:2000],file='Ychunk4.txt',sep=',',row.names=F,col.names=F)
```

(a) Report the first 6 entries in $\hat{\beta}$ using `lm` on all the data simultaneously.

```
fit = lm(Y ~ 0 + X)
round(head(fit$coefficients), 5) # beta.hat
```

```
##       X1       X2       X3       X4       X5       X6
## -0.97823 -0.39771 -0.71099 -0.62446  0.23622 -0.50964
```

(b) Alternatively, we can read in each chunk and update the solution using `biglm`. Here is the first part. Complete the procedure in the natural way on the remaining chunks.

```
require(biglm)

Xchunk = read.table(file='Xchunk1.txt',sep=',',header=T)
Ychunk = scan(file='Ychunk1.txt',sep=',')

form = as.formula(paste('Ychunk ~ -1 + ',paste(names(Xchunk),collapse=' + '),
  collapse=''))
out.biglm = biglm(formula = form,data=Xchunk)

Xchunk = read.table(file='Xchunk2.txt',sep=',',header=T)
```

```r
Ychunk = scan(file='Ychunk2.txt',sep=',')
out.biglm = update(out.biglm,moredata=Xchunk)

Xchunk = read.table(file='Xchunk3.txt',sep=',',header=T)
Ychunk = scan(file='Ychunk3.txt',sep=',')
out.biglm = update(out.biglm,moredata=Xchunk)

Xchunk = read.table(file='Xchunk4.txt',sep=',',header=T)
Ychunk = scan(file='Ychunk4.txt',sep=',')
out.biglm = update(out.biglm,moredata=Xchunk)

round(head(out.biglm$qr$thetab), 5)
```

```
## [1] -1.09942 -0.35895 -0.52637 -1.04163  0.37634  0.48184
```

Compare the first 6 entries in $\hat{\beta}$ formed by this method with the entries in (a).

- The two methods generate slightly different results.