PS7

ShubeiWang

11/12/2018

1

We can calculate the sample standard error of the 1000 estimated regression coefficients and compare to the average of the 1000 estimated standard errors. If they're close enough we can say the statistical method properly characterize the uncertainty of the estimated regression coefficient.

2

(a)

There are 1×10^9 observations and 8×10^9 predictors, so the total memory is $(8+1) \times 10^9 \times 8$ bytes = 72GB.

(b)

Firstly we can use a 10000×8 matrix(denoted by $\hat{X}$) to store the 10000 unique combinations of covariates(by rows). That would take up $8 \times 10000 \times 8$ bytes = 0.64MB.

Then we can use a vector v of length n to store the indices. $v[i] = k$ means that the i_{th} row in X is the same as the k_{th} row in $\hat{X}$. Since each index is an integer, that would take up 4×10^9 bytes = 4GB.

We still need 8GB for the storage of n observations. To sum up the total memory is 12GB+0.64MB, which is much less than the memory used in (a).

(c)

If we run lm(), glm() on the data, a least-square linear model will be fitted to each column of X seperately so we'll need the whole data set. Thus the work in (b) will be useless.

(d)

The notations are the same as in (b).

Firstly we need to solve $A = X^\top X$. This is a symmetric matrix so we only need to find the upper triangle entries. Denote $X = (X_1, X_2, ...)$. Note that $X_i = \hat{X}[v, i]$, we can use $A_{ij} =< X_i, X_j >$ to compute A. Then we can compute $(X^\top X)^{-1} = A^{-1}$. After that we need to compute $(X^\top X)^{-1} X^\top Y$. We can start with $X^\top Y$ to reduce the computational complexity.

The pseudo-code is as below:

```
p <- 8
A <- matrix(data = NA, nrow=p, ncol=p)

# compute the upper triangle
for (i in 1:p){
```

```r
  for (j in i:p){
    A[i,j] <- crossprod(X_hat[v, i], X_hat[v, j])
  }
}

# get A
A <-forceSymmetrix(A)

# compute the inverse of A
A_inv <- solve(A)

B <- matrix(data = NA, nrow=p, ncol=1)

# compute t(X)*Y
for (k in 1:p){
 B[k, 1] <- crossprod(X_hat[v, i], Y)
}

# final result
result <- A %*% B
```

3

Since Σ is positive definite it admits eigen value decomposition $\Sigma = U\Lambda U^T$,hence

$$X^T\Sigma^{-1}X\hat{\beta} = X^TU\Lambda^{-1}U^TY = X^TU\Lambda^{-1/2}\Lambda^{-1/2}U^TY$$

denote

$$\tilde{X} = \Lambda^{-1/2}U^TX, \tilde{Y} = \Lambda^{-1/2}U^TY$$

then using QR decomposition $X = QR$we have

$$\tilde{X}^T\tilde{X}\hat{\beta} = \tilde{X}^T\tilde{Y}$$

$$\Rightarrow R^TQ^TQR\hat{\beta} = R^TQ^T\tilde{Y}$$

$$\Rightarrow R\hat{\beta} = Q^T\tilde{Y}$$

since R is upper-triangular we can then backsolve $\hat{\beta}$.

```r
## R code for the above procedure

gls <- function(x,y,sigma){
  #eigen value decomposition
  eigen <- eigen(sigma)
  u <- eigen$vectors
  lambdas <- eigen$values
  inv_lambdas <- lambdas^(-1/2)

  #transform X and Y
  tilde_x <- inv_lambdas*t(u)%*%x
  tilde_y <- inv_lambdas*t(u)%*%y

  #QR decomposition
  x.qr <- qr(tilde_x)
  Q <- qr.Q(x.qr)
  R <- qr.R(x.qr)

  #calculate beta
  beta <- backsolve(R, t(Q)%*%tilde_y)
}
```

4

(a)

To get the upper triangular matrix the first step is to zero out the first column of A except for the first row. Then zero out the second column of A except for the first two rows and then proceed like that until all entries below the diagonal are zeroed out. The number of computations is

$$\sum_{i=1}^{n-1}(i+1)i = \sum_{i=1}^{n-1} i^2 + \sum_{i=1}^{n-1} i = \frac{n(n+1)}{2} + \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + n^2 + \frac{2n}{3}$$

To get I^*, The number of computations is

$$n \times [(n-1) + (n-2) + ... + 1] = \frac{n^2(n+1)}{2} = \frac{n^3}{2} + \frac{n^2}{2}$$

In total, the number of computations is $\frac{5n^3}{6} + \frac{3n^2}{2} + \frac{2n}{3}$. The computational complexity is $O(n^3)$.

(b)

The number of computations for solving for Z can be calculated using backsolve$(U, I*)$. The number of operations is $\frac{n^3}{2}$ and the computational complexity is $O(n^3)$.

(c)

Z is a $n \times n$ matrix. b is a vector of length n. Using matrix multiplication we know the number of operations is n^2 and the computational complexity is $O(n^2)$.

5

(a)

```r
W <- matrix(rnorm(5000*5000), ncol=5000, nrow=5000)
X <- crossprod(W,W)
y <- rnorm(5000)

system.time(solve(X)%*%y)
system.time(LU <- solve(X,y))
system.time({
  U <- chol(X)
  Cholesky <- backsolve(U, backsolve(U, y, transpose = TRUE))
})
```

method(a):

time: 106.652

order of computations: According to problem 4, to solve X we need $\frac{5n^3}{6} + \frac{n^3}{2} + O(n^2)$ operations. To compute $X^{-1}y$, we need n^2 operations. In total, we need $\frac{4n^3}{3} + O(n^2)$ operations.

method(b):

time: 24.697

order of computations: We are using the LU decomposition when using solve(X,y). LU involves $n^3/3 + O(n^2)$ computations in total.

method(c):

time: 23.270

order of computations: Cholesky involves $n^3/6 + O(n^2)$ computations.

The comparison between the timing of method (a) and (b) seems to agree with the order of computations we discussed in class. But the results of (b) and (c) are very close, which is not consistent with what I would expect.

(b)

```
## compute the condition number
eigen <- eigen (X)
condition <- max(eigen$values)/min(eigen$values)
print(condition)
## 210704896

## see if result of (b),(c) are the same numerically
identical(LU, Cholesky)
## FALSE

## see how many digits agree
print(formatC(LU[1], 20, format = 'f'))
print(formatC(Cholesky[1], 20, format = 'f'))
##-88.32920150226011912764
##-88.32920203592964014661
```

From the results we can see that condition numbers $\approx 2 \times 10^8$, so we have accuracy of order 10^{8-16}. That means we have 8 digits of accuracy, which agrees with the result.