Stat243 ps7

Winnie Gao

11/13/2018

Problem 1

Suppose I have a statistical method that estimates a regression coefficient and its standard error. I develop a simulation study and have m = 1000 simulated datasets that each give me an estimate of the coefficent and its standard error. How would I determine if the statistical method properly characterizes the uncertainty of the estimated regression coefficient? Note your answer could be as simple as a sentence or two describing what quantities to consider.

Ans: We should take the simulation variance of coefficient values, so that we can get the Monte Carlo Simulation error. We need to check if the sample standard deviation of all coefficient estimates is close to the average of the 1000 standard errors. If that's the case, then we can determine the statistical method properly characterizes the uncertainty of the estimated regression coefficient.

Problem 2

Suppose I have a very large dataset, with n = 1x10^9, so I have n observations and an n x p matrix X of predictors, where p = 8.

(a) Ordinarily, how much memory would the dataset take up?

Ans: We need 72GB to store the dataset. Because we have (8+1)*10^9 numbers in total and each one takes 8 bytes, so we need 72GB.

(b) Now suppose that there are only 10000 unique combinations of the p covariates. Given what you know about data structures in R, how could you store the data to use up much less memory? How much memory would be used by your solution?

Ans: To store matrix X of predictors, we can create a new matrix which has 10000 rows and each row stands for a unique combination of the p covariates. This matrix takes about 0.64 MB (10000*64/1000000). And we store a new vector which indicates that index location of each row from the original matrix takes. Suppose for the first row, the original matrix has values [1,2,3,4,5,6,7,8], and it's stored as 3rd row in the compressed matrix, so in the index location matrix, first row has value of 4. Since we have only 10000 different index location, and we can store it in binary format and we only need 14 bits to store 10000 numbers (2^14=16384). Therefore, we need 14 bits times 10^9 in memory, which is 1.75GB. In total, we need 9.75064GB (n observations included).

(c) Now suppose you need to run lm(), glm(), etc. on this data. Why would all of your work in part (b) go to waste?

Ans: When we run lm() and glm() function, they will perform linear regression model in columnwise way. They have to extract the information for each column, therefore, they still need to form the original n*p matrix X of predictors.

(d) If you need to find the OLS, $(X^TX)^{-1}X^TY$ estimator here, how could you code this up (please provide pseudo-code; you don't need to write any R code) so that you do not need to use up the full memory and can take advantage of your data structure(s).

Ans: First of all, since X^TX is a $8*8$ symmetric matrix, then we only need to calculate the upper triangular part. And we let M denotes the result of X^TX. Suppose X^* is the compressed matrix which has 10000 rows and each row stands for a unique combination of the p covariates, so X^* is matrix of shape $10000*8$. Suppose n is the 10^9*1 vector that stores the index location information. Therefore, M[1,1] element is the inner product of X[,1] (first column of X) and X[,1], M[1,2] element is the inner product of X[,1] and X[,2] (second column of X), etc. To get X[,1], we can use $X^*[n,1]$. Similarly, we can use X^* and n to extract the information about each column and use for loop to fill up each element in the upper triangular part of X^TX.

The for loop can be:

```
for (i in 1:8){
for (j in i:8){
$M[i,j] = X^*[n,i]$ %*% $X^*[n,j]$
}
}
```

Then we use forceSymmetric function to fill the lower triangular part of matrix M.

For X^TY part, the result should be a $8*1$ vector and i-th element should be the inner product of X[,i] (i-th column of X) and Y. We can use $X^*[n,1]$ to get X[,i] and then get the matrix for X^TY.

Finally, we can take the inverse of X^TX then multiplies X^TY to find the OLS estimator.

Problem 3

Suppose I need to compute the generalized least squares estimator, $\hat{\beta} = (X^T\Sigma^{-1}X)^{-1}X^T\Sigma^{-1}Y$, for X $n*p$, $\Sigma n*n$ and assume that n > p. Assume n could be of order several thousand and p of order in the hundreds. First write out in pseudo-code how you would do this in an efficient way - i.e., the particular linear algebra steps and the order of operations. Then write efficient R code in the form of a function, gls(), to do this - you can rely on the various high-level functions for matrix decompositions and solving systems of equations, but you should not use any code that already exists for doing generalized least squares.

Ans: Since for linear system, $X^T\Sigma^{-1}X\hat{\beta} = X^T\Sigma^{-1}Y$, and Σ is square and symmetric, therefore we can use eigendecomposition on it. So $\Sigma = U\Lambda U^T$ where U is orthogonal and Λ is diagonal matrix of eigenvalues. Therefore,

$$X^T(U\Lambda U^T)^{-1}X\hat{\beta} = X^T(U\Lambda U^T)^{-1}Y$$
$$X^TU\Lambda^{-1}U^TX\hat{\beta} = X^TU\Lambda^{-1}U^TY$$

Since $\Lambda^{-1/2}\Lambda^{-1/2} = \Lambda^{-1}$, then

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

If we let X^* denotes $\Lambda^{-1/2}U^TX$ and Y^* denotes $\Lambda^{-1/2}U^TY$. Then we can get,

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

Then we can use QR decomposition to find $\hat{\beta}$.

$$R^TQ^TQR\hat{\beta} = R^TQ^TY^*$$
$$R\hat{\beta} = Q^TY^*$$

```r
gls <- function(X, Y, Sigma){
  eigen_decom = eigen(Sigma)
  U = eigen_decom$vectors
  Lamb = eigen_decom$values
  Lamb_inv_root = Lamb^(-1/2)

  X_new = Lamb_inv_root*t(U)%*%X
  Y_new = Lamb_inv_root*t(U)%*%Y

  X.qr = qr(X_new)
  Q = qr.Q(X.qr)
  R = qr.R(X.qr)

  beta_hat = backsolve(R, t(Q)%*%Y_new)
}
```

Problem 4

We've seen how to use Gaussian elimination (i.e., the LU decomposition) to solve Ax = b and that we can do the solution in n^3/3 operations (plus lower-order terms). Now let's consider matrix-vector multiplication. Count the number of computations for

(a) transforming AZ = I to $UZ = I^*$ (where I^* is no longer a diagonal matrix),

(b) for solving for Z given $UZ = I^*$, and

(c) for calculating x = Zb.

Then compare the total cost to the n^3/3 cost of what we saw in class.

Ans:

For step (a), we need to find the upper triangular matrix, which is U. For first column, we need to compute n(n-1) times (we have n-1 rows in total, for each row, we need to compute one division and n-1 times multiplication); for second column, we need (n-1)(n-2) times computation (we have n-2 rows in total, for each row, we need to compute one division and n-2 times multiplication), etc. In general, it takes $O(\frac{n^3}{3})$ operations. Then we need to find I^* ($LI^* = I$) using forwardsolve, which takes n*(1+2+...+n-1+n)=$\frac{n^2(n+1)}{2}$ operations. Therefore, for step (a), it takes $O(\frac{n^3}{3} + \frac{n^3}{2})$ operations.

For step(b), we can use backsolve to get Z, which takes n*(n+n-1+...+2+1)=$\frac{n^2(n+1)}{2}$ operations. Therefore, step (b) takes $O(\frac{n^3}{2})$ operations.

For step(c), we do the multiplication of Z and b, which takes O(n^2) operations.

In total, instead of using Gaussian elimination, we have to do the solution in $O(\frac{4n^3}{3} + n^2)$ operations. So the cost is much higher than n^3/3.

Problem 5

Compare the speed of $b = X^{-1}y$ using: (a)solve(X)%*%y, (b) solve(X,y), and (c) Cholesky decomposition followed by solving triangular systems. Do this for a matrix of size 5000x5000 using a single thread, using a matrix X constructed from W^TW where the elements of the nxn matrix W are generated independently using rnorm().

(a) How do the timing and relative ordering amongst methods compare to the order of computations we discussed in class, the notes, and above in problem 4.

```r
library(RhpcBLASctl)
n=5000
set.seed(11)
blas_set_num_threads(1)
X = crossprod(matrix(rnorm(n^2),n))
y = rnorm(n)
system.time({
  out1 <- solve(X)%*%y
})
```

```
##    user  system elapsed
## 119.668   1.312 126.574
```

```r
system.time({
  out2 <- solve(X,y)
})
```

```
##    user  system elapsed
##  25.192   0.088  25.333
```

```r
system.time({
  U <- chol(X)
  out3 <- backsolve(U, backsolve(U, y, transpose = TRUE))
})
```

```
##    user  system elapsed
##  23.288   0.078  23.415
```

Ans: According to the result, method(a) takes the most time and method(b) and method(c) take the similar amount of time. For method(a), we will need to do $O(4n^3/3)$ to get the inverse of matrix X (derived from problem 4) and then do the multiplication which is $O(n^2)$. For method(b), which uses LU decomposition, it involves $\frac{n^3}{3} + O(n^2)$ operations. Cholesky method involves about $\frac{n^3}{6} + O(n^2)$ of computation.

(b) Are the results for b the same numerically for methods (b) and (c) (up to machine precision)? Comment on how many digits in the elements of b agree, and relate this to the condition number of the calculation.

```r
formatC(out2[1:5],20)
```

```
## [1] "6.7664331035951539306"  "34.539487720833228934"
## [3] "0.22145319371639973771" "-14.418213385603248611"
## [5] "-13.545476481615558839"
```

```r
formatC(out3[1:5],20)
```

```
## [1] "6.7664331049651469385"  "34.539487725328314127"
## [3] "0.22145319232753674088" "-14.418213387811476878"
## [5] "-13.545476483003055179"
```

```
e <- eigen(X)
cond_X <- e$values[1]/e$values[5000]
cond_X
```

```
## [1] 18255237
```

Ans: As we can see, the results for b for methods(b) and (c) are not the same numerically (up to machine precision). 9 digits in the elements of b agree. condition number of X is approximately 10^7. Therefore, we have accuracy of order 10^(7-16) which is 10^(-9). Therefore, we have lost 7 digits of accuracy.