Multivariate Bayesian Regression

Charles Zheng

06/25/2015

Setup

Suppose we have a $n \times p_Y$ response matrix Y and a $n \times p_X$ covariate matrix X. We have the model

$$Y = XB + E$$

where B is a $p_X \times p_Y$ coefficient matrix and E is a matrix of noise. Write

$$E = \begin{pmatrix} e_1^T \\ e_2^T \\ \dots \\ e_n^T \end{pmatrix}$$

Assume the rows of E are independent, and each row is distributed

$$e_i \sim N(0, \Sigma_e)$$

Let us assume a prior distribution on the coefficients,

$$B_{ij} \sim N(0, \sigma^2_{B,j})$$

and where B_{ij} are independent. Note that each column $j = 1, ..., p_Y$ has a different prior coviarance $\sigma^2_{B,j}$. This reflects our prior knowledge that some column of Y may have more signal than other columns.

Bayes' rule gives us the posterior mean and covariance of the coefficients B_{ij}. First, we introduce the vectorized notation $\vec{B}$ to denote the $p_X p_Y \times 1$ vector obtained by stacking the columns of B, and similarly $\vec{Y}$ to denote the $p_Y n \times 1$ vector obtained by stacking the columns of Y. Also recall the definition of Kronecker product,

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots \\ a_{21}B & a_{22}B & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

Also define

$$\Omega_e = \Sigma_e^{-1}$$

$$\Omega_b = \text{diag}\left(\frac{1}{\sigma^2_{b,1}}, ..., \frac{1}{\sigma^2_{b,p_Y}}\right)$$

The posterior mean and covariance are given as follows.

$$\text{E}[\vec{B}|Y] = (\Omega_e \otimes X^T X + \Omega_b \otimes I_{p_X})^{-1}((I \otimes X)^T (\Omega_e \otimes I)\vec{y})$$

$$\text{Cov}(\vec{B}|Y) == (\Omega_e \otimes X^T X + \Omega_b \otimes I_{p_X})^{-1}$$

Simulated Example

Setup the parameters and generate the data.

```r
library(magrittr)
library(pracma, warn.conflicts = FALSE)
library(MASS)
f2 <- function(x) sum(x^2)
solvediag <- function(D, b) 1/diag(D) * b
n <- 20
pX <- 60 # number of X-features: assume larger than n
pY <- 60 # number of Y-responses
Sigma_b <- 0.01 * diag(abs(rnorm(pY))) # size of each random coefficent
# noise covariance
Sigma_e <- 10 * (1/10/pY) * randn(10 * pY, pY) %>% {t(.) %*% .}
X <- randn(n, pX)
# Generate coefficents
B <- mvrnorm(n = pX, mu = rep(0, pY), Sigma = Sigma_b)
Y <- X %*% B + mvrnorm(n = n, mu = rep(0, pY), Sigma = Sigma_e)
```

Compute the posterior mean and covariance, also timing the computation. In the timings, we assume $X^T X$ and Ω_e are pre-computed.

```r
Omega_e <- solve(Sigma_e)
Omega_b <- diag(1/diag(Sigma_b))
yVec <- as.numeric(Y)
xtx <- t(X) %*% X
```

Computation of the posterior covariance.

```r
tt <- proc.time()
post_cov <- solve(Omega_e %x% xtx + Omega_b %x% eye(pY))
proc.time() -tt
```

```
##    user  system elapsed
##  22.592   0.248  23.161
```

Computation of the posterior mean.

```r
tt <- proc.time()
post_mu <- solve(Omega_e %x% xtx + Omega_b %x% eye(pY),
                 t(eye(pY) %x% X) %*% (Omega_e %x% eye(n)) %*% yVec)
proc.time() -tt
```

```
##    user  system elapsed
##  10.618   0.279  11.088
```

Computation

In the following, we assume that $p_X > n$.

Naively, computing the posterior covariance takes $O(p_X^3 p_Y^3)$ operations and computing the posterior mean takes $O(p_X^2 p_Y^2)$ operations.

However, by diagonalizing the covariance matrix and taking advantage of the properties of Kronecker products, we can make the computation much more efficient.

Using simultaneous diagonalization, find V_e, D_e such that

$$\Omega_e = V_e D_e V_e^T$$

and

$$\Omega_b = V_e V_e^T$$

Note that V_e is not orthogonal. The procedure for finding V_e, D_e is well-known and also given in the code.

Furthermore, define

$$\tilde{X} = \begin{pmatrix} X \\ 0 \end{pmatrix}$$

so that $\tilde{X}$ is an $p_X \times p_X$ matrix. Then take the SVD

$$\tilde{X} = U_X D_X V_X^T$$

so that D_X and V_X are both $p_X \times p_X$. We have

$$V_X V_X^T = V_X V_X^T = I_{p_X}$$

and

$$V_X D_X^2 V_X^T = X^T X.$$

Now we can rewrite the expression

$$\begin{aligned}\text{Cov}(\vec{B}|Y) &== (\Omega_e \otimes X^T X + \Omega_b \otimes I_{p_X})^{-1} \\ &= ((V_e D_e V_e^T) \otimes (V_X D_X^2 V_X^T) + (V_e V_e^T) \otimes (V_X V_X^T))^{-1} \\ &= [(V_e \otimes V_X)(D_e \otimes D_X^2 + I_{p_X p_Y})(V_e^T \otimes V_X^T)]^{-1} \\ &= (V_e^{-1} \otimes V_X^T)^T (D_e \otimes D_X^2 + I_{p_X p_Y})^{-1} (V_e^{-1} \otimes V_X^T)\end{aligned}$$

In this form the expression is much each to compute since

- Only a diagonal matrix needs be inverted, and yields a diagonal matrix
- Multiplying the tranpose of a Kronecker product with a diagonal with itself is easy to compute.

To see the second point, consider computing the product

$$C = (A \otimes B)^T D (A \otimes B)$$

where A is $a_1 \times a_2$, B is $b_1 \times b_2$, and D is diagonal with

$$D = \begin{pmatrix} D_1 & 0 & 0 & 0 & ... \\ 0 & D_2 & 0 & 0 & ... \\ 0 & 0 & D_3 & 0 & ... \\ ... & ... & ... & ... & ... \end{pmatrix}$$

such that each $D_1, ..., D_{a_1}$ is $b_1 \times b_1$.

For $i, j = 1..., a_2$, let $C_{[ij]}$ denote $b_2 \times b_2$ blocks such that

$$C = \begin{pmatrix} C_{[11]} & C_{[12]} & ... \\ C_{[21]} & C_{[22]} & ... \\ ... & ... & ... \end{pmatrix}$$

Now we can write

$$C_{[ij]} = \sum_{k=1}^{a_1} a_{ki}a_{kj}B^T D_k B = B^T \left(\sum_{k=1}^{a_1} a_{ki}a_{kj} D_k \right) B.$$

Thus to compute each block of C, it takes $O(a_1 b_1)$ operations to compute the weighted sum of $D_1, ..., D_{a_1}$, and then $O(b_2^1 b_1)$ to multiply on the left with B^T and on the right with B. Therefore, we see that it takes $O(a_2^2 b_2^2 b_1)$ operations to compute C.

Applying to our problem, we see that the cost to evaluate the posterior covariance using this method is $O(p_Y^2 p_X^3)$, which is a saves a power of p_Y compared to the naive approach.

Example (cont.)

Computing V_e, D_e, V_X, D_X and also V_e^{-1}.

```
homega_b <- diag(1/sqrt(diag(Sigma_b)))
hsigma_b <- diag(sqrt(diag(Sigma_b)))
res <- eigen(hsigma_b %*% Omega_e %*% hsigma_b)
D_e <- diag(res$values)
V_e <- homega_b %*% res$vectors
iV_e <- solve(V_e)
resX <- svd(rbind(X, zeros(pX - n, pX)))
V_x <- resX$v
D_x <- diag(resX$d)
```

Function for computing Kronecker times diagonal times transposed Kromnecker, and demo.

```
# computes t(A %x% B) %*% diag(d) %*% (A %*% B)
tkron_d_kron <- function(A, B, d) {
  a1 <- dim(A)[1]; a2 <- dim(A)[2]; b1 <- dim(B)[1]; b2 <- dim(B)[2]
  dmat <- matrix(d, b1, a1) # columns of dmat are diag(D1), diag(D2), ...
  AtA <- t(A) %*% A
  C <- zeros(a2 * b2)
  for (i in 1:a2) {
    for (j in i:a2) {
      dtemp <- as.numeric(dmat %*% (A[, i] * A[, j]))
      Cij <- t(B) %*% (dtemp * B)
      C[(i-1) * b2 + (1:b2), (j-1) * b2 + (1:b2)] <- Cij
      C[(j-1) * b2 + (1:b2), (i-1) * b2 + (1:b2)] <- t(Cij)
    }
  }
  C
}
A <- randn(10, 20)
B <- randn(20, 30)
d <- rnorm(10 * 20)
C <- t(A %x% B) %*% diag(d) %*% (A %x% B)
C2 <- tkron_d_kron(A, B, d)
f2(C - C2)
```

```
## [1] 7.134721e-24
```

Compute the posterior covariance and time it

```r
tt <- proc.time()
# invert diagonal matrix
d <- 1/(diag(D_e) %x% diag(D_x)^2 + 1)
post_cov2 <- tkron_d_kron(iV_e, t(V_x), d)
proc.time() -tt
```

```
##    user  system elapsed
##   0.861   0.168   1.037
```

Check that the answer matches

```r
f2(post_cov2 - post_cov)
```

```
## [1] 1.669464e-29
```