Estiamting multivariate normal models by Stan II

Xulong Wang

October 21, 2015

```r
library(rstan)
```

```
## Warning: package 'rstan' was built under R version 3.1.3
```

```
## Loading required package: Rcpp
```

```
## Warning: package 'Rcpp' was built under R version 3.1.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

```
## rstan (Version 2.8.0, packaged: 2015-09-19 14:48:38 UTC, GitRev: 05c3d0058b6a)
## For execution on a local, multicore CPU with excess RAM we recommend calling
## rstan_options(auto_write = TRUE)
## options(mc.cores = parallel::detectCores())
```

```r
rm(list = ls())
setwd("~/Dropbox/Github/BDA")
```

Model

$$y = \mu + random + residual$$

$$random \sim multi_norm(0, \sigma * K)$$

$$residual\ normal(0, \epsilon * I)$$

```r
load("adsp.rdt")
mdata <- adsp$mdata[1:100, ]
Sigma <- adsp$kinship$autosome[1:100, 1:100]
Sigma[Sigma < 0] <- 0

model <- stan_model("mvn.stan")
```

```r
dat1 <- list(L = nrow(mdata), y = mdata$AD1, Sigma = Sigma, prior = 0)
dat2 <- list(L = nrow(mdata), y = mdata$AD1, Sigma = Sigma, prior = 2)
dat3 <- list(L = nrow(mdata), y = mdata$AD1, Sigma = Sigma, prior = 4)

opt <- optimizing(model, data = dat1)
```

```
## STAN OPTIMIZATION COMMAND (LBFGS)
## init = random
## save_iterations = 1
## init_alpha = 0.001
## tol_obj = 1e-12
## tol_grad = 1e-08
## tol_param = 1e-08
## tol_rel_obj = 10000
## tol_rel_grad = 1e+07
## history_size = 5
## seed = 1073873365
## initial log joint probability = -31626.2
## Optimization terminated normally:
##   Convergence detected: relative gradient magnitude is below tolerance
```

```
opt$par[c("mu", "sigma", "epsilon", "z[1]", "u[1]")]
```

```
##            mu         sigma       epsilon          z[1]          u[1]
##  6.171775e-01  2.220446e-16  3.381692e-01 -4.169842e-05 -9.258910e-21
```

```
opt <- optimizing(model, data = dat2)
```

```
## STAN OPTIMIZATION COMMAND (LBFGS)
## init = random
## save_iterations = 1
## init_alpha = 0.001
## tol_obj = 1e-12
## tol_grad = 1e-08
## tol_param = 1e-08
## tol_rel_obj = 10000
## tol_rel_grad = 1e+07
## history_size = 5
## seed = 2036377972
## initial log joint probability = -5955.44
## Optimization terminated normally:
##   Maximum number of iterations hit, may not be at an optima
```

```
opt$par[c("mu", "sigma", "epsilon", "z[1]", "u[1]")]
```

```
##            mu         sigma       epsilon          z[1]          u[1]
## -6.028590e-01  1.239173e+00  8.427790e-08  6.882487e-01  8.528590e-01
```

```
opt <- optimizing(model, data = dat3)
```

```
## STAN OPTIMIZATION COMMAND (LBFGS)
## init = random
## save_iterations = 1
## init_alpha = 0.001
## tol_obj = 1e-12
## tol_grad = 1e-08
## tol_param = 1e-08
```

```
## tol_rel_obj = 10000
## tol_rel_grad = 1e+07
## history_size = 5
## seed = 1525111553
## initial log joint probability = -300.569
## Optimization terminated normally:
##   Convergence detected: relative gradient magnitude is below tolerance
```

```
opt$par[c("mu", "sigma", "epsilon", "z[1]", "u[1]")]
```

```
##            mu         sigma       epsilon          z[1]          u[1]
##  6.354216e-01  3.489616e-06  3.385083e-01 -1.820659e-04 -6.353402e-10
```

1. Random effect was tiny
2. Prior takes effect in optimizing()
3. As long as prior wasn't extremely ill, it does not affect the inference in noticable scale

```
fit <- sampling(model, chain = 2, data = dat1, iter = 600, warmup = 200)
```

```
##
## SAMPLING FOR MODEL 'mvn' NOW (CHAIN 1).
##
## Chain 1, Iteration:   1 / 600 [  0%]  (Warmup)
## Chain 1, Iteration:  60 / 600 [ 10%]  (Warmup)
## Chain 1, Iteration: 120 / 600 [ 20%]  (Warmup)
## Chain 1, Iteration: 180 / 600 [ 30%]  (Warmup)
## Chain 1, Iteration: 201 / 600 [ 33%]  (Sampling)
## Chain 1, Iteration: 260 / 600 [ 43%]  (Sampling)
## Chain 1, Iteration: 320 / 600 [ 53%]  (Sampling)
## Chain 1, Iteration: 380 / 600 [ 63%]  (Sampling)
## Chain 1, Iteration: 440 / 600 [ 73%]  (Sampling)
## Chain 1, Iteration: 500 / 600 [ 83%]  (Sampling)
## Chain 1, Iteration: 560 / 600 [ 93%]  (Sampling)
## Chain 1, Iteration: 600 / 600 [100%]  (Sampling)
## #  Elapsed Time: 5.49307 seconds (Warm-up)
## #                2.33773 seconds (Sampling)
## #                7.8308 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mvn' NOW (CHAIN 2).
##
## Chain 2, Iteration:   1 / 600 [  0%]  (Warmup)
## Chain 2, Iteration:  60 / 600 [ 10%]  (Warmup)
## Chain 2, Iteration: 120 / 600 [ 20%]  (Warmup)
## Chain 2, Iteration: 180 / 600 [ 30%]  (Warmup)
## Chain 2, Iteration: 201 / 600 [ 33%]  (Sampling)
## Chain 2, Iteration: 260 / 600 [ 43%]  (Sampling)
## Chain 2, Iteration: 320 / 600 [ 53%]  (Sampling)
## Chain 2, Iteration: 380 / 600 [ 63%]  (Sampling)
## Chain 2, Iteration: 440 / 600 [ 73%]  (Sampling)
## Chain 2, Iteration: 500 / 600 [ 83%]  (Sampling)
## Chain 2, Iteration: 560 / 600 [ 93%]  (Sampling)
```

```
## Chain 2, Iteration: 600 / 600 [100%]  (Sampling)
## #  Elapsed Time: 5.17309 seconds (Warm-up)
## #                2.08731 seconds (Sampling)
## #                7.2604 seconds (Total)
```

```r
print(fit, pars = c("mu", "sigma", "epsilon", "z[1]", "u[1]"))
```

```
## Inference for Stan model: mvn.
## 2 chains, each with iter=600; warmup=200; thin=1;
## post-warmup draws per chain=400, total post-warmup draws=800.
##
##          mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
## mu       0.62    0.01 0.06  0.50  0.59  0.62 0.66  0.74    33 1.06
## sigma    0.14    0.03 0.10  0.01  0.05  0.12 0.23  0.33    12 1.16
## epsilon  0.30    0.02 0.07  0.14  0.27  0.32 0.34  0.38    10 1.21
## z[1]    -0.50    0.08 1.00 -2.21 -1.20 -0.55 0.10  1.66   165 1.02
## u[1]    -0.10    0.04 0.18 -0.57 -0.19 -0.04 0.00  0.15    17 1.11
##
## Samples were drawn using NUTS(diag_e) at Wed Oct 21 14:24:12 2015.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```r
fit <- sampling(model, chain = 2, data = dat2, iter = 600, warmup = 200)
```

```
##
## SAMPLING FOR MODEL 'mvn' NOW (CHAIN 1).
##
## Chain 1, Iteration:   1 / 600 [  0%]  (Warmup)
## Chain 1, Iteration:  60 / 600 [ 10%]  (Warmup)
## Chain 1, Iteration: 120 / 600 [ 20%]  (Warmup)
## Chain 1, Iteration: 180 / 600 [ 30%]  (Warmup)
## Chain 1, Iteration: 201 / 600 [ 33%]  (Sampling)
## Chain 1, Iteration: 260 / 600 [ 43%]  (Sampling)
## Chain 1, Iteration: 320 / 600 [ 53%]  (Sampling)
## Chain 1, Iteration: 380 / 600 [ 63%]  (Sampling)
## Chain 1, Iteration: 440 / 600 [ 73%]  (Sampling)
## Chain 1, Iteration: 500 / 600 [ 83%]  (Sampling)
## Chain 1, Iteration: 560 / 600 [ 93%]  (Sampling)
## Chain 1, Iteration: 600 / 600 [100%]  (Sampling)
## #  Elapsed Time: 5.40878 seconds (Warm-up)
## #                2.02728 seconds (Sampling)
## #                7.43606 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mvn' NOW (CHAIN 2).
##
## Chain 2, Iteration:   1 / 600 [  0%]  (Warmup)
## Chain 2, Iteration:  60 / 600 [ 10%]  (Warmup)
## Chain 2, Iteration: 120 / 600 [ 20%]  (Warmup)
## Chain 2, Iteration: 180 / 600 [ 30%]  (Warmup)
## Chain 2, Iteration: 201 / 600 [ 33%]  (Sampling)
## Chain 2, Iteration: 260 / 600 [ 43%]  (Sampling)
```

```
## Chain 2, Iteration: 320 / 600 [ 53%]  (Sampling)
## Chain 2, Iteration: 380 / 600 [ 63%]  (Sampling)
## Chain 2, Iteration: 440 / 600 [ 73%]  (Sampling)
## Chain 2, Iteration: 500 / 600 [ 83%]  (Sampling)
## Chain 2, Iteration: 560 / 600 [ 93%]  (Sampling)
## Chain 2, Iteration: 600 / 600 [100%]  (Sampling)
## #  Elapsed Time: 5.18572 seconds (Warm-up)
## #                3.42262 seconds (Sampling)
## #                8.60834 seconds (Total)
```

```
print(fit, pars = c("mu", "sigma", "epsilon", "z[1]", "u[1]"))
```

```
## Inference for Stan model: mvn.
## 2 chains, each with iter=600; warmup=200; thin=1;
## post-warmup draws per chain=400, total post-warmup draws=800.
##
##          mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## mu       0.65    0.00 0.06  0.54  0.61  0.65  0.68  0.76   210 1.02
## sigma    0.21    0.02 0.09  0.02  0.16  0.22  0.28  0.35    27 1.08
## epsilon  0.27    0.01 0.06  0.15  0.23  0.28  0.32  0.37    19 1.14
## z[1]    -0.78    0.05 0.85 -2.28 -1.34 -0.87 -0.30  1.19   307 1.02
## u[1]    -0.19    0.03 0.19 -0.58 -0.33 -0.18 -0.04  0.15    48 1.06
##
## Samples were drawn using NUTS(diag_e) at Wed Oct 21 14:24:40 2015.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

1. Random effect was also small, but much larger than the optimizing() result
2. σ clearly higher and comparable with ϵ in sampling()

Run sampling() on selected markers

```
sample = extract(fit)

C2 <- sample$C2
C3 <- chol(Sigma)

t(C3)[1:5, 1:5]
```

```
##            SRR1057414 SRR1060488 SRR1060491 SRR1057411 SRR1057423
## SRR1057414     1.0000 0.00000000 0.00000000  0.0000000   0.000000
## SRR1060488     0.2672 0.96364110 0.00000000  0.0000000   0.000000
## SRR1060491     0.2604 0.21825669 0.94050830  0.0000000   0.000000
## SRR1057411     0.0472 0.03527056 0.02563623  0.9979333   0.000000
## SRR1057423     0.0444 0.03137716 0.02369991  0.2459985   0.967454
```

```
C2[1, 1:5, 1:5]
```

```
##        [,1]       [,2]       [,3]      [,4]     [,5]
## [1,] 1.0000 0.00000000 0.00000000 0.0000000 0.000000
## [2,] 0.2672 0.96364110 0.00000000 0.0000000 0.000000
## [3,] 0.2604 0.21825669 0.94050830 0.0000000 0.000000
## [4,] 0.0472 0.03527056 0.02563623 0.9979333 0.000000
## [5,] 0.0444 0.03137716 0.02369991 0.2459985 0.967454
```

$$C_{stan} = (chol(K))^T$$

```
u = sample$u
u2 = sample$u2

all(u == u2)
```

```
## [1] TRUE
```

```
z = sample$z
my_u <- sample$sigma[1] * t(chol(Sigma)) %*% z[1, ]

u[1, 1:10]
```

```
##  [1]  0.13637858 -0.05616676  0.21407174  0.06176288 -0.06706891
##  [6] -0.08975032 -0.13796216 -0.04598234  0.01188251  0.11765792
```

```
my_u[1:10, ]
```

```
##  SRR1057414  SRR1060488  SRR1060491  SRR1057411  SRR1057423  SRR1057420
##  0.13637858 -0.05616676  0.21407174  0.06176288 -0.06706891 -0.08975032
##  SRR1057432  SRR1104759  SRR1104762  SRR1104774
## -0.13796216 -0.04598234  0.01188251  0.11765792
```

Stan use same mechanism for "generated quantities" and "transformed parameters": direct computing, no estimation

```
cov = cov(u)

cor(Sigma[1, ], cov(u)[1, ])
```

```
## [1] 0.3609019
```

```
cor(Sigma[2, ], cov(u)[2, ])
```

```
## [1] 0.4967355
```

```
cor(Sigma[3, ], cov(u)[3, ])
```

```
## [1] 0.3619015
```

1. Covariances of the random samples are not Sigma.
2. We permit the random effect to be estimated again for each model
3. We let covariates and response information flowing back to help estimating the random effect