Package 'semmcmc'

May 13, 2021

Title Bayesian Structural Equation Modeling in Multiple Omics Data Integration

Version 0.0.6

Date 2021-04-20

Description Provides Markov Chain Monte Carlo (MCMC) routine for the structural equation modelling described in Maity et. al. (2020) <doi:10.1093/bioinformatics/btaa286>. This MCMC sampler is useful when one attempts to perform an integrative survival analysis for multiple platforms of the Omics data where the response is time to event and the predictors are different omics expressions for different platforms.

Depends R (>= 4.0)

Imports MASS, msm

License GPL-3

Encoding UTF-8

LazyData FALSE

RoxygenNote 7.1.1

NeedsCompilation no

Author Arnab Maity [aut, cre],
Sang Chan Lee [aut],
Bani K. Mallick [aut],
Samsiddhi Bhattacharjee [aut],
Nidhan K. Biswas [aut]

Maintainer Arnab Maity <arnab.maity@pfizer.com>

Repository CRAN

Date/Publication 2021-05-13 04:20:03 UTC

R topics documented:

mcmc *mcmc function*

Description

MCMC routine for the strucatural equation model

Usage

```
mcmc(ct, u1, u2, X, nburnin = 1000, nmc = 2000, nthin = 1)
```

Arguments

ct	survival response, a $n*2$ matrix with first column as response and second column as right censored indicator, 1 is event time and 0 is right censored.
u1	Matix of predictors from the first platform, dimension $n * q_1$
u2	Matix of predictors from the first platform, dimension $n * q_2$
X	Matrix of covariates, dimension $n * p$.
nburnin	number of burnin samples
nmc	number of markov chain samples
nthin	thinning parameter. Default is 1 (no thinning)

Value

pMean.beta.t

pMean.beta.t

pMean.alpha.t

pMean.alpha.t

pMean.phi.t

pMean.phi.t

pMean.alpha.u1

pMean.alpha.u2

pMean.alpha.u2

pMean.phi.u1

pMean.eta1

pMean.eta2

pMean.sigma.t.square

pMean.sigma.u1.square

pMean.sigma.u2.square

alpha.t.samples

phi.t.samples
beta1.t.samples

beta2.t.samples

beta.t.samples
alpha.u1.samples

alpha.u2.samples

phi.u1.samples

phi.u2.samples

eta1.samples

eta2.samples
sigma.t.square.samples

sigma.u1.square.samples

sigma.u2.square.samples

pMean.logt.hat

DIC

WAIC

References

Maity, A. K., Lee, S. C., Mallick, B. K., & Sarkar, T. R. (2020). Bayesian structural equation modeling in multiple omics data integration with application to circadian genes. Bioinformatics, 36(13), 3951-3958.

Examples

```
require(MASS)
# for random number from multivariate normal distribution

n <- 100  # number of individuals
p <- 5    # number of variables
q1 <- 20    # dimension of the response
q2 <- 20    # dimension of the response

ngrid   <- 1000
nburnin <- 100
nmc     <- 200
nthin   <- 5
```

```
niter   <- nburnin + nmc
effsamp <- (niter - nburnin)/nthin

alpha.tt  <- runif(n = 1, min = -1, max = 1)  # intercept term
alpha.u1t <- runif(n = 1, min = -1, max = 1)  # intercept term
alpha.u2t <- runif(n = 1, min = -1, max = 1)  # intercept term
beta.tt   <- runif(n = p, min = -1, max = 1)  # regression parameter
gamma1.t  <- runif(n = q1, min = -1, max = 1)
gamma2.t  <- runif(n = q2, min = -1, max = 1)
phi.tt    <- 1
phi.u1t   <- 1
phi.u2t   <- 1

sigma.tt    <- 1
sigma.u1t   <- 1
sigma.u2t   <- 1
sigma.etat1 <- 1
sigma.etat2 <- 1

x <- mvrnorm(n = n, mu = rep(0, p), Sigma = diag(p))



eta2 <- rnorm(n = 1, mean = 0, sd = sigma.etat2)
eta1 <- rnorm(n = 1, mean = eta2, sd = sigma.etat1)
logt <- rnorm(n = n, mean = alpha.tt + x %*% beta.tt + eta1 * phi.tt,
sd = sigma.tt)
u1   <- matrix(rnorm(n = n * q1, mean = alpha.u1t + eta1 * phi.u1t,
sd = sigma.u1t), nrow = n, ncol = q1)
u2   <- matrix(rnorm(n = n * q2, mean = alpha.u2t + eta2 * phi.u2t,
sd = sigma.u2t), nrow = n, ncol = q2)
logt <- rnorm(n = n, mean = alpha.tt + x %*% beta.tt + u1 %*% gamma1.t +
u2 %*% gamma2.t, sd = sigma.tt)

# Survival time generation
T <- exp(logt)   # AFT model
C <- rgamma(n, shape = 1, rate = 1)  # 50% censor
time <- pmin(T, C)  # observed time is min of censored and true
status = time == T   # set to 1 if event is observed
1 - sum(status)/length(T)   # censoring rate
censor.rate <- 1 - sum(status)/length(T)    # censoring rate
censor.rate
summary(C)
summary(T)
ct <- as.matrix(cbind(time = time, status = status))  # censored time
logt.grid <- seq(from = min(logt) - 1, to = max(logt) + 1, length.out = ngrid)

index1 <- which(ct[, 2] == 1)  # which are NOT censored
ct1    <- ct[index1, ]

posterior.fit.sem <- mcmc(ct, u1, u2, x, nburnin = nburnin,
nmc = nmc, nthin = nthin)
```

```
pMean.beta.t   <- posterior.fit.sem$pMean.beta.t
pMean.alpha.t  <- posterior.fit.sem$pMean.alpha.t
pMean.phi.t    <- posterior.fit.sem$pMean.phi.t
pMean.alpha.u1 <- posterior.fit.sem$pMean.alpha.u1
pMean.alpha.u2 <- posterior.fit.sem$pMean.alpha.u2
pMean.phi.u1   <- posterior.fit.sem$pMean.phi.u1
pMean.phi.u2   <- posterior.fit.sem$pMean.phi.u2
pMean.eta1     <- posterior.fit.sem$pMean.eta1
pMean.eta2     <- posterior.fit.sem$pMean.eta2
pMean.logt.hat <- posterior.fit.sem$posterior.fit.sem

pMean.sigma.t.square  <- posterior.fit.sem$pMean.sigma.t.square
pMean.sigma.u1.square <- posterior.fit.sem$pMean.sigma.u1.square
pMean.sigma.u2.square <- posterior.fit.sem$pMean.sigma.u2.square
pMean.logt.hat        <- posterior.fit.sem$pMean.logt.hat

DIC.sem  <- posterior.fit.sem$DIC
WAIC.sem <- posterior.fit.sem$WAIC
mse.sem  <- mean(pMean.logt.hat[index1] - log(ct1[, 1]))^2

alpha.t.samples         <- posterior.fit.sem$alpha.t.samples
beta1.t.samples         <- posterior.fit.sem$beta1.t.samples
beta2.t.samples         <- posterior.fit.sem$beta2.t.samples
beta.t.samples          <- posterior.fit.sem$beta.t.samples
phi.t.samples           <- posterior.fit.sem$phi.t.samples
alpha.u1.samples        <- posterior.fit.sem$alpha.u1.samples
alpha.u2.samples        <- posterior.fit.sem$alpha.u2.samples
phi.u1.samples          <- posterior.fit.sem$phi.u1.samples
phi.u2.samples          <- posterior.fit.sem$phi.u2.samples
sigma.t.square.samples  <- posterior.fit.sem$sigma.t.square.samples
sigma.u1.square.samples <- posterior.fit.sem$sigma.u1.square.samples
sigma.u2.square.samples <- posterior.fit.sem$sigma.u2.square.samples
eta1.samples            <- posterior.fit.sem$eta1.samples
eta2.samples            <- posterior.fit.sem$eta2.samples

inv.cpo <- matrix(0, nrow = effsamp, ncol = n)
# this will store inverse cpo values
log.cpo <- rep(0, n)                        # this will store log cpo
for(iter in 1:effsamp)  # Post burn in
{
  inv.cpo[iter, ] <- 1/(dnorm(ct[, 1], mean = alpha.t.samples[iter] +
  x %*% beta.t.samples[, iter] +
                              + eta1.samples[iter] * phi.t.samples[iter],
                          sd = sqrt(sigma.t.square.samples[iter]))^ct[, 2] *
                        pnorm(ct[, 1], mean = alpha.t.samples[iter] +
                        x %*% beta.t.samples[, iter] +
                                + eta1.samples[iter] * phi.t.samples[iter],
                              sd = sqrt(sigma.t.square.samples[iter]),
                              lower.tail = FALSE)^(1 - ct[, 2]))
}                   # End of iter loop
for (i in 1:n){
  log.cpo[i]   <- -log(mean(inv.cpo[, i]))
  # You average invcpo[i] over the iterations,
```

```
  # then take 1/average and then take log.
  # Hence the negative sign in the log
}
lpml.sem <- sum(log.cpo)



DIC.sem
WAIC.sem
mse.sem
```

Index