STA 601 Homework 10b

Lingyun Shao

Nov. 24, 2018

10.5

Logistic regression variable selection: Consider a logistic regression model for predicting diabetes as a function of x_1 = number of pregnancies, x_2 = blood pressure, x_3 = body mass index, x_4 = diabetes pedigree and x_5 = age. Using the data in `azdiabetes.dat`, center and scale each of the $x-$variables by subtracting the sample average and dividing by the sample standard deviation for each variable. Consider a logistic regression model of the form $Pr(Y_i = 1|x_i, \beta, z) = e^{\theta_i}/(1 + e^{\theta_i})$ where

$$\theta_i = \beta_0 + \beta_1\gamma_1 x_{i,1} + \beta_2\gamma_2 x_{i,2} + \beta_3\gamma_3 x_{i,3} + \beta_4\gamma_4 x_{i,4} + \beta_5\gamma_5 x_{i,5}.$$

In this model, each γ_j is either 0 or 1, indicating whether or not variable j is a predictor of diabetes. For example, if it were the case that $\gamma = (1, 1, 0, 0, 0)$, then $\theta_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2}$. Obtain posterior distributions for β and γ, using independent prior distributions for the parameters, such that $\gamma_j \sim binary(1/2), \beta_0 \sim normal(0, 16)$ and $\beta_j \sim normal(0, 4)$ for each $j > 0$.

a) Implement a Metropolis-Hastings algorithm for approximating the posterior distribution of β and γ. Examine the sequences $\beta_j^{(s)}$ and $\beta_j^{(s)} \times \gamma_j^{(s)}$ for each j and discuss the mixing of the chain.

Based on the instruction and referring to Dellaportas et al 2000, it's easy for us to find the kernel of posterior for each parameter as the following:

Update β_0

$$p(\beta_0|\gamma, \beta, X, y) \propto p(\beta_0)p(y|\gamma, \beta_0, \beta, X)$$

Update $\beta_j, j > 0$

$$p(\beta_j|\gamma, \beta_0, \beta_{-j}, X, y) \propto \begin{cases} p(y|\gamma, \beta_0, \beta, X)p(\beta_j), & \gamma_j = 1 \\ p(\beta_j), & \gamma_j = 0 \end{cases}$$

Update γ

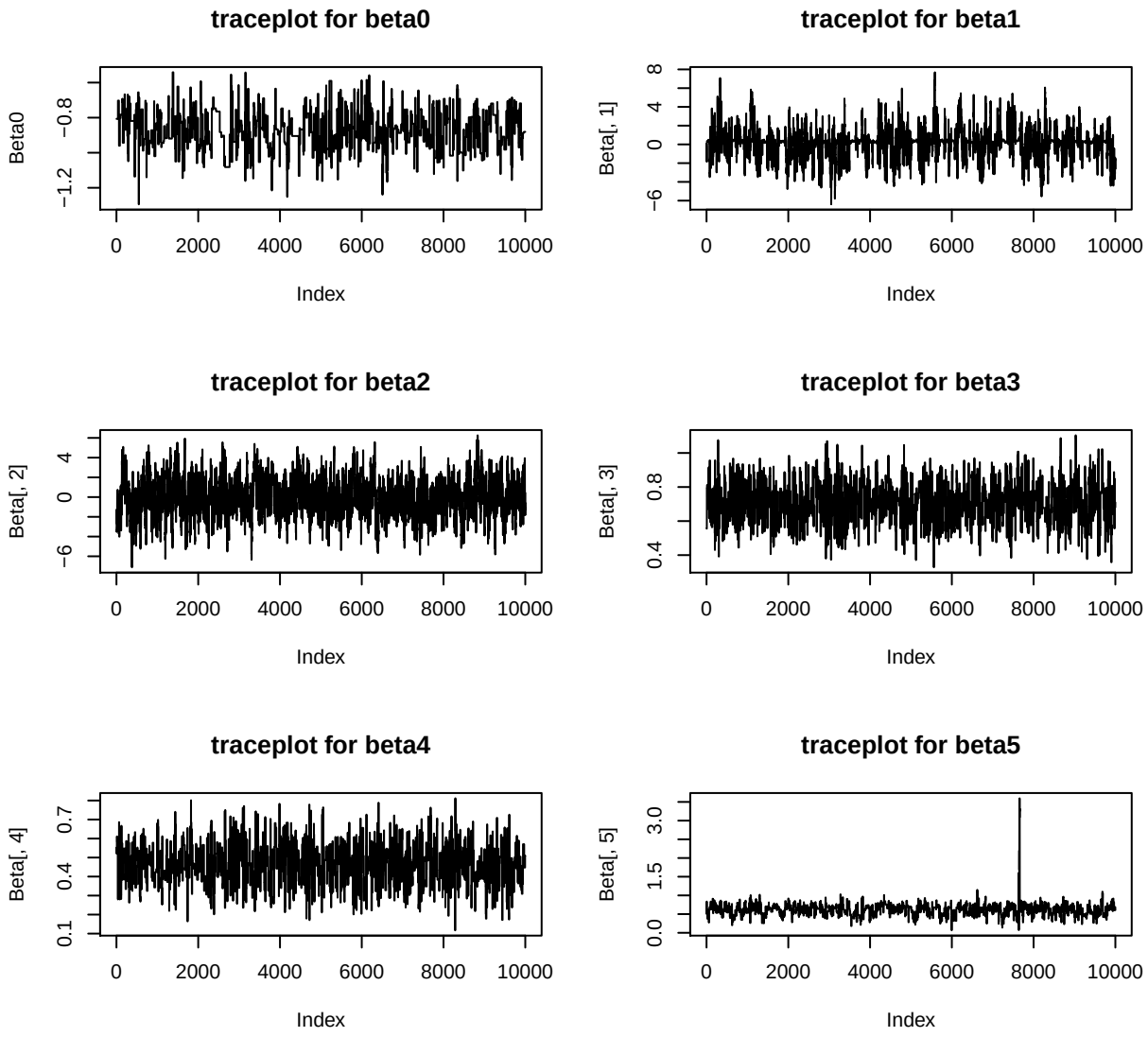
$$\begin{aligned} \gamma_j &\sim Bernoulli(\pi_j) \\ \pi_j &= \frac{a_j}{a_j + b_j} \\ a_j &= p(\gamma_j = 1|\gamma_{-j}, \beta_0, \beta, X, y) = p(y|\gamma_j = 1, \gamma_{-j}, \beta_0, \beta, X)Pr(\gamma_j = 1) \\ b_j &= p(\gamma_j = 0|\gamma_{-j}, \beta_0, \beta, X, y) = p(y|\gamma_j = 0, \gamma_{-j}, \beta_0, \beta, X)Pr(\gamma_j = 0) \end{aligned}$$

Based on these theoretical results, we can derive a sampler which uses Metropolis algorithm to sample β_0 and β_j, then directly sample γ_j from Binomial. The update functions are defined below.

```r
dia = read.table(file = 'http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/azdiabetes.dat',
                 header = TRUE, stringsAsFactors = FALSE)
y = (dia[,8]=='Yes')*1
X = dia[,c('npreg', 'bp', 'bmi', 'ped', 'age')] %>% as.matrix() %>% scale()

logit = function(p) {log(p/(1-p))}
expit = function(x) {exp(x)/(1+exp(x))}

# need this fix function for situation like:
# 0 = log(0^0) = 0*log(0) producing NaN and NA
fix = function(x) {
  ind = is.nan(x)|is.na(x)
  x[ind] = 0
  x
}

update_beta0 = function(gamma, beta, beta0) {
  beta0_p = rnorm(1, beta0, sd = 2) # propose beta0
  pi = expit(beta0 + as.matrix(X[,gamma==1]) %*% beta[gamma==1]) # pi_original
  pi_p = expit(beta0_p + as.matrix(X[,gamma==1]) %*% beta[gamma==1]) # pi_proposed
  log_a = sum(fix(y*log(pi_p))+fix((1-y)*log(1-pi_p))) + dnorm(beta0_p, mean=0, sd=4, log=TRUE) # log n
  log_b = sum(fix(y*log(pi))+fix((1-y)*log(1-pi))) + dnorm(beta0, mean=0, sd=4, log=TRUE) # log denomin
  log_r = log_a - log_b
  u = runif(1)
  beta0_new = ifelse(log(u)<log_r, beta0_p, beta0)
  return(beta0_new)
}

update_beta = function(gamma, beta, beta0) {
  p = length(beta)
  # for code simplicity, treat beta_gamma, beta_{-gamma} as the same
  for(j in 1:p) {
    beta_p = beta
    beta_p[j] = rnorm(1, beta[j], sd = 1) # proposal beta
    pi = expit(beta0 + as.matrix(X[,gamma==1]) %*% beta[gamma==1]) # pi_original
    pi_p = expit(beta0 + as.matrix(X[,gamma==1]) %*% beta_p[gamma==1]) # pi_proposed
    log_a = sum(fix(y*log(pi_p))+fix((1-y)*log(1-pi_p))) + dnorm(beta_p[j], mean=0, sd=2, log=TRUE) # l
    log_b = sum(fix(y*log(pi))+fix((1-y)*log(1-pi))) + dnorm(beta[j], mean=0, sd=2, log=TRUE) # log den
    log_r = log_a - log_b
    log_r
    u = runif(1)
    beta[j] = ifelse(log(u)<log_r, beta_p[j], beta[j])
  }
  return(beta)
}

update_gamma = function(gamma, beta, beta0) {
  # randomly choose update order
  p = length(gamma)
  for(j in sample(p)) {
    gamma_a = gamma_b = gamma
    gamma_a[j] = 1 # for numerator
    gamma_b[j] = 0 # for denominator
```

```
    pi_a = expit(beta0 + as.matrix(X[,gamma_a==1]) %*% beta[gamma_a==1]) # pi_original
    log_a = sum(fix(y*log(pi_a))+fix((1-y)*log(1-pi_a)))# log numerator
    pi_b = expit(beta0 + as.matrix(X[,gamma_b==1]) %*% beta[gamma_b==1]) # pi_original
    log_b = sum(fix(y*log(pi_b))+fix((1-y)*log(1-pi_b)))# log numerator
    log_odds = log_a - log_b
    u = runif(1)
    gamma[j] = ifelse(u < expit(log_odds), 1, 0)
  }
  return(gamma)
}
# initial values
p = 5
gamma = rep(1, p)
beta = rep(0, p)
beta0 = 1
S = 10500
B = 500 # Burn-in
Gamma = matrix(NA, nrow = S, ncol = p)
Beta = matrix(NA, nrow = S, ncol = p)
Beta0 = rep(NA, S)

# update parameters
for(i in 1:S) {
  beta0 = update_beta0(gamma, beta, beta0)
  beta = update_beta(gamma, beta, beta0)
  gamma = update_gamma(gamma, beta, beta0)
  Beta0[i] = beta0
  Beta[i,] = beta
  Gamma[i,] = gamma
  # print(i)
}

Beta0 = Beta0[-(1:B)]
Beta = Beta[-(1:B),]
Gamma = Gamma[-(1:B),]
```

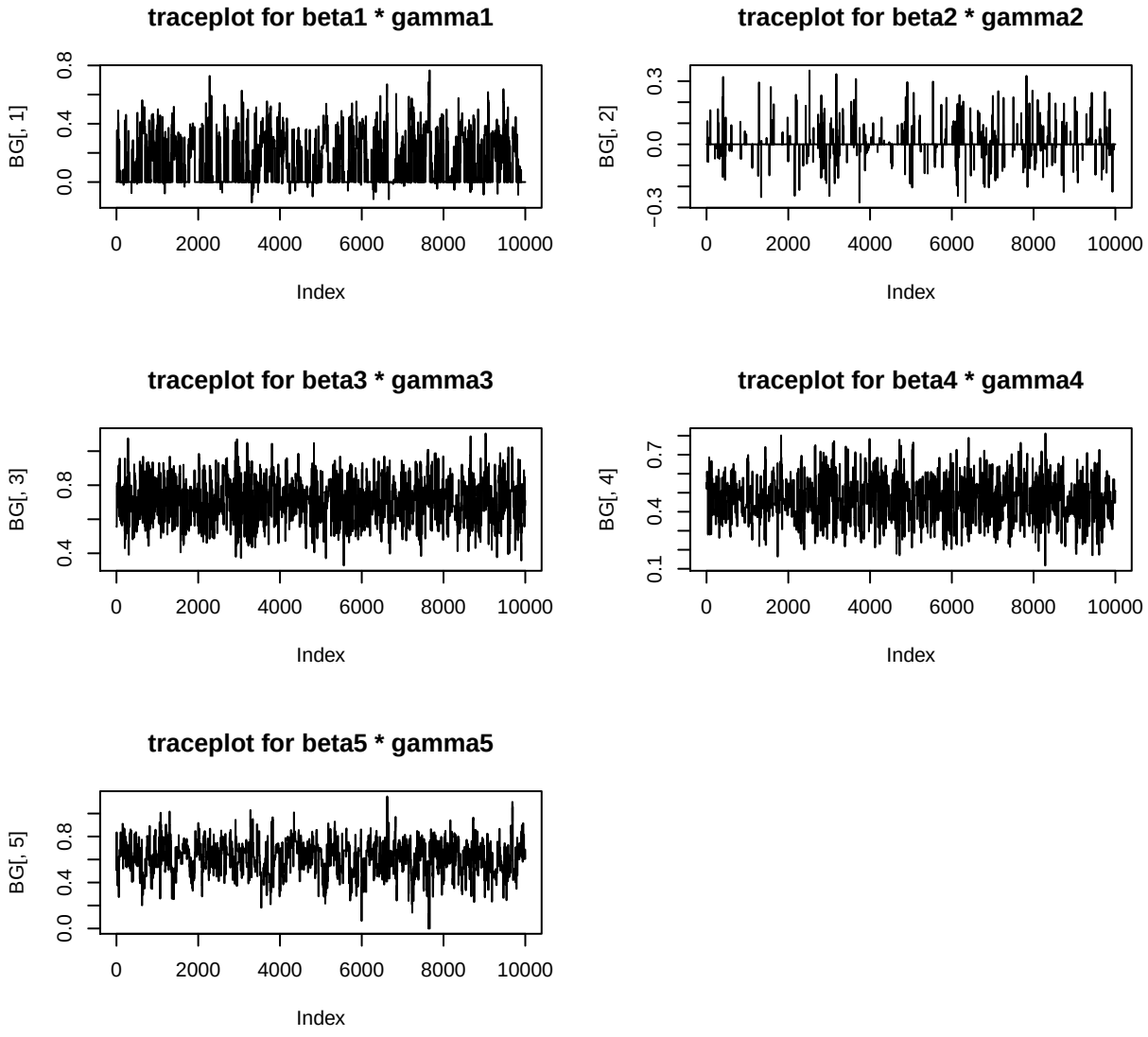
```
# stationary plot
# stationarity.plot = function(x,...){
#   S = length(x)
#   scan = 1:S
#   ng = min( round(S/100),10)
#   group = S*ceiling( ng*scan/S) /ng
#   boxplot(x~group,...) }

par(mfrow = c(3,2))
plot(Beta0, type = 'l', main = 'traceplot for beta0')
plot(Beta[,1], type = 'l', main = 'traceplot for beta1')
plot(Beta[,2], type = 'l', main = 'traceplot for beta2')
plot(Beta[,3], type = 'l', main = 'traceplot for beta3')
plot(Beta[,4], type = 'l', main = 'traceplot for beta4')
plot(Beta[,5], type = 'l', main = 'traceplot for beta5')
```




```
BG = Beta * Gamma
par(mfrow = c(3,2))
plot(BG[,1], type = 'l', main = 'traceplot for beta1 * gamma1')
plot(BG[,2], type = 'l', main = 'traceplot for beta2 * gamma2')
plot(BG[,3], type = 'l', main = 'traceplot for beta3 * gamma3')
plot(BG[,4], type = 'l', main = 'traceplot for beta4 * gamma4')
plot(BG[,5], type = 'l', main = 'traceplot for beta5 * gamma5')

res = rbind(apply(Beta, 2, effectiveSize),
            apply(BG, 2, effectiveSize))
rownames(res) = c('beta', 'beta \\* gamma')
res %>%
  kable(col.names = 1:5,
        caption = 'Effective Sample Size out of 10000')
```

Table 1: Effective Sample Size out of 10000

	1	2	3	4	5
beta	441.2977	441.1333	864.7364	950.5455	328.4921
beta * gamma	230.1314	4975.9489	864.7364	950.5455	337.4064



According to the traceplot and effective sample size, we may find that this Markov Chain have reached stationarity and have a fairly OK mixing.

The effective sample size for $\beta's$ are not very big because we used Metropolis algorithm and it is very likeli we get the same sample from time to time. Thus, the mixing of this chain is fairly satisfactory.

b) Approximate the posterior probability of the top five most frequently occurring values of γ. How good do you think the MCMC estimates of these posterior probabilities are?

```
top5 = do.call(paste0, as.data.frame(Gamma)) %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  .[1:5]
top5/10000
```

```
## .
##  00111  10111  01111  11111  10110
## 0.5240 0.4180 0.0312 0.0239 0.0029
```

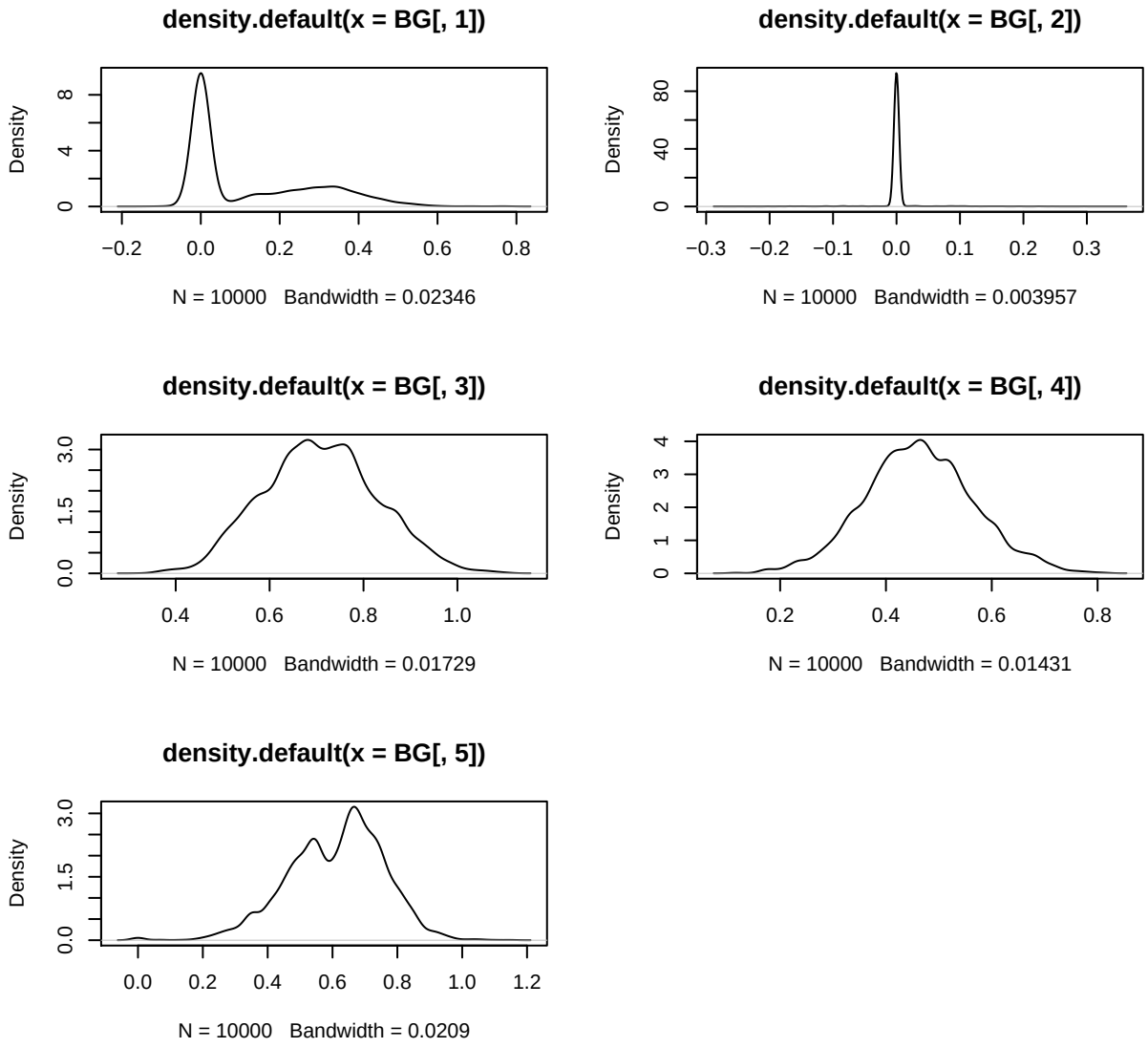
The posterior probabilities of the top five most frequently occuring γ are given above. We can find that our method is mostly regarding x_3, x_4, x_5 as important variables.

I think the MCMC estimates of these posterior probabilities look good to me since they end up with some similar important variables.

c) For each j, plot posterior densities and obtain posterior means for $\beta_j\gamma_j$. Also obtain $Pr(\gamma_j = 1|x, y)$

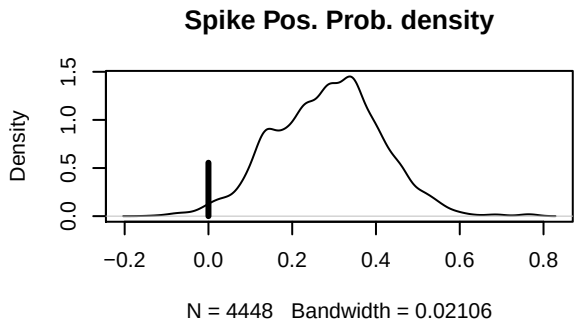
```
par(mfrow = c( 3,2))
plot(density(BG[,1]))
plot(density(BG[,2]))
plot(density(BG[,3]))
plot(density(BG[,4]))
plot(density(BG[,5]))
res = cbind(
  c(
    mean(BG[,1]),
    mean(BG[,2]),
    mean(BG[,3]),
    mean(BG[,4]),
    mean(BG[,5])
  ),
  c(
    mean(Gamma[,1]),
    mean(Gamma[,2]),
    mean(Gamma[,3]),
    mean(Gamma[,4]),
    mean(Gamma[,5])
  )
)
rownames(res) = paste('beta', 1:5, 'gamma', 1:5)
colnames(res) = c('posterior mean', 'Pr(gamma_j = 1|x,y)')

kable(res)
```

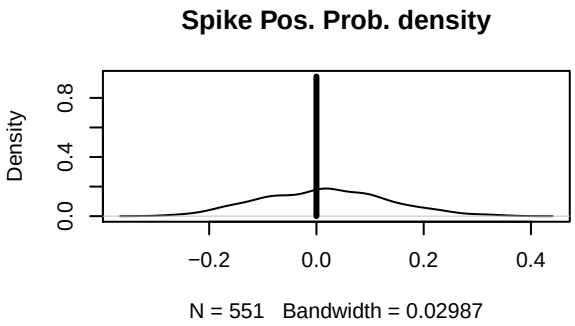
	posterior mean	Pr(gamma_j = 1|x,y)
beta 1 gamma 1	0.1263681	0.4448
beta 2 gamma 2	0.0008758	0.0551
beta 3 gamma 3	0.7078612	1.0000
beta 4 gamma 4	0.4625626	1.0000
beta 5 gamma 5	0.6108730	0.9971

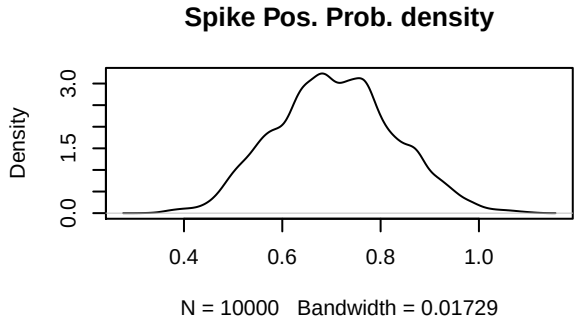


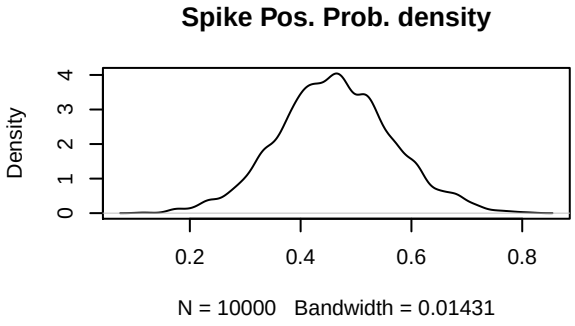
Another way of drawing posterior densities is to draw a probability mass at 0 and densities at other values:

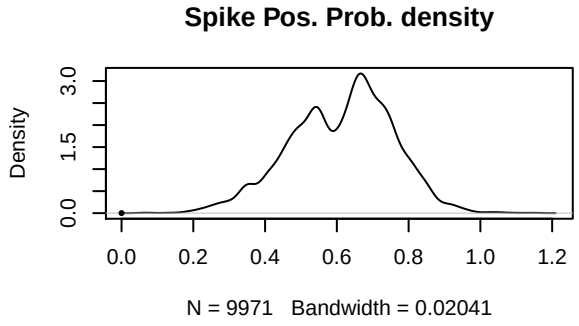
```
spike_dens = function(dt) {
  p0 = mean(dt == 0)
  dens = density(dt[dt!=0])
  dens$y = dens$y *(1-p0)
  lim = max(dens$y, p0)
  plot(dens, ylim = c(0,lim), main = 'Spike Pos. Prob. density')
  segments(0,0,0, p0, lwd = 3)
}
par(mfrow=c(3,2))
# for example, beta1 gamma1
spike_dens(BG[,1])
spike_dens(BG[,2])
spike_dens(BG[,3])
spike_dens(BG[,4])
spike_dens(BG[,5])
```











Mixture Prior Approach from George and McCullagh 1993

Implement the mixture prior approach I described from George and McCullagh 1993. rather than writing

$$\theta_i = \beta_0 + \beta_1\gamma_1 x_{i,1} + \beta_2\gamma_2 x_{i,2} + \beta_3\gamma_3 x_{i,3} + \beta_4\gamma_4 x_{i,4} + \beta_5\gamma_5 x_{i,5}.$$

write

$$\theta_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \beta_4 x_{i,4} + \beta_5 x_{i,5}.$$

and write the prior for $\beta_j|\gamma_j$ as what's written in the book if $\gamma_j = 1$ and as having $1/10$ of the variance if $\gamma_j = 0$. Compare the output of the two approaches.

Based on the instruction and referring to George and McCullagh 1993, it's easy for us to find the kernel of posterior for each parameter as the following:

Update β_0

$$p(\beta_0|\gamma, \beta, X, y) \propto p(\beta_0)p(y|\beta_0, \beta, X)$$

Update $\beta_j, j > 0$

$$p(\beta|\gamma, \beta_0, X, y) \propto p(y|\beta_0, \beta, X)p(\beta|\gamma)$$

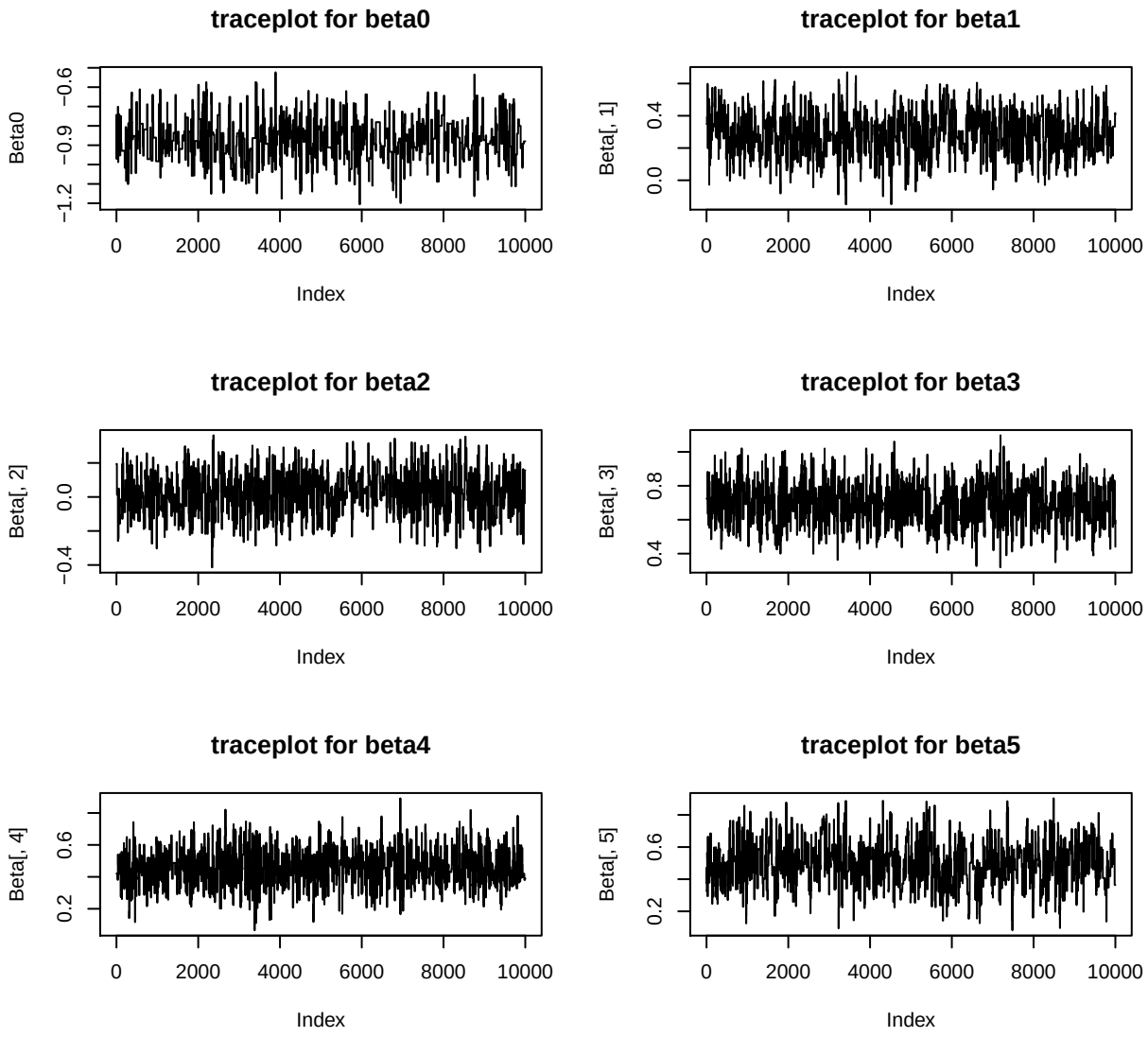
Update γ

$$\begin{aligned}
\gamma_j &\sim Bernoulli(\pi_j) \\
\pi_j &= \frac{a_j}{a_j + b_j} \\
a_j &= p(\gamma_j = 1|\gamma_{-j}, \beta_0, \beta, X, y) = p(\beta|\gamma_j = 1, \gamma_{-j})Pr(\gamma_j = 1) \\
b_j &= p(\gamma_j = 0|\gamma_{-j}, \beta_0, \beta, X, y) = p(\beta|\gamma_j = 0, \gamma_{-j})Pr(\gamma_j = 0)
\end{aligned}$$

Based on these theoretical results, the update functions are defined below.

```
update_beta0 = function(beta, beta0) {
  beta0_p = rnorm(1, beta0, sd = 2) # propose beta0
  pi = expit(beta0 + as.matrix(X) %*% beta) # pi_original
  pi_p = expit(beta0_p + as.matrix(X) %*% beta) # pi_proposed
  log_a = sum(fix(y*log(pi_p))+fix((1-y)*log(1-pi_p))) + dnorm(beta0_p, mean=0, sd=4, log=TRUE) # log n
  log_b = sum(fix(y*log(pi))+fix((1-y)*log(1-pi))) + dnorm(beta0, mean=0, sd=4, log=TRUE) # log denomin
  log_r = log_a - log_b
  u = runif(1)
  beta0_new = ifelse(log(u)<log_r, beta0_p, beta0)
  return(beta0_new)
}

update_beta = function(gamma, beta, beta0) {
  p = length(beta)
  # for code simplicity, treat beta_gamma, beta_{-gamma} as the same
  for(j in 1:p) {
    beta_p = beta
    sd.beta = (gamma[j]==1) * 2 + (gamma[j]!=1) * 2/sqrt(10)
    beta_p[j] = rnorm(1, beta[j], sd = 1) # proposal beta
    pi = expit(beta0 + as.matrix(X) %*% beta) # pi_original
    pi_p = expit(beta0 + as.matrix(X) %*% beta_p) # pi_proposed
    log_a = sum(fix(y*log(pi_p))+fix((1-y)*log(1-pi_p))) + dnorm(beta_p[j], mean=0, sd=sd.beta, log=TRU
    log_b = sum(fix(y*log(pi))+fix((1-y)*log(1-pi))) + dnorm(beta[j], mean=0, sd=sd.beta, log=TRUE) # l
    log_r = log_a - log_b
    log_r
    u = runif(1)
    beta[j] = ifelse(log(u)<log_r, beta_p[j], beta[j])
  }
  return(beta)
}
```

```r
update_gamma = function(gamma, beta) {
  # randomly choose update order
  p = length(gamma)
  for(j in sample(p)) {
    gamma_a = gamma_b = gamma
    gamma_a[j] = 1 # for numerator
    gamma_b[j] = 0 # for denominator
    log_a = sum(dnorm(beta, 0, sd = ((gamma_a==1) * 2 + (gamma_a!=1) * 2/sqrt(10)), log = TRUE))# log n
    log_b = sum(dnorm(beta, 0, sd = ((gamma_b==1) * 2 + (gamma_b!=1) * 2/sqrt(10)), log = TRUE))# log n
    log_odds = log_a - log_b
    u = runif(1)
    gamma[j] = ifelse(u < expit(log_odds), 1, 0)
  }
  return(gamma)
}

# initial values
p = 5
gamma = rep(1, p)
beta = rep(0, p)
beta0 = 1
S = 10500
B = 500 # Burn-in
Gamma = matrix(NA, nrow = S, ncol = p)
Beta = matrix(NA, nrow = S, ncol = p)
Beta0 = rep(NA, S)

# update parameters
for(i in 1:S) {
  beta0 = update_beta0(beta, beta0)
  beta = update_beta(gamma, beta, beta0)
  gamma = update_gamma(gamma, beta)
  Beta0[i] = beta0
  Beta[i,] = beta
  Gamma[i,] = gamma
  # print(i)
}

Beta0 = Beta0[-(1:B)]
Beta = Beta[-(1:B),]
Gamma = Gamma[-(1:B),]

par(mfrow = c(3,2))
plot(Beta0, type = 'l', main = 'traceplot for beta0')
plot(Beta[,1], type = 'l', main = 'traceplot for beta1')
plot(Beta[,2], type = 'l', main = 'traceplot for beta2')
plot(Beta[,3], type = 'l', main = 'traceplot for beta3')
plot(Beta[,4], type = 'l', main = 'traceplot for beta4')
plot(Beta[,5], type = 'l', main = 'traceplot for beta5')
```



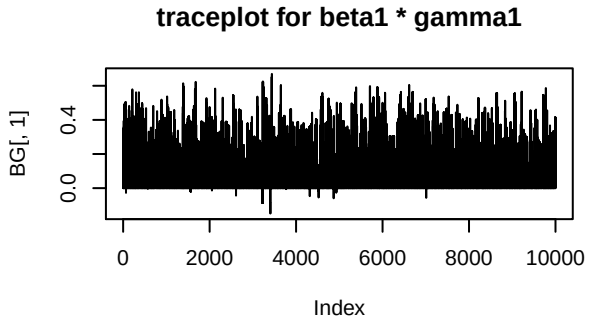

```
BG = Beta * Gamma
par(mfrow = c(3,2))
plot(BG[,1], type = 'l', main = 'traceplot for beta1 * gamma1')
plot(BG[,2], type = 'l', main = 'traceplot for beta2 * gamma2')
plot(BG[,3], type = 'l', main = 'traceplot for beta3 * gamma3')
plot(BG[,4], type = 'l', main = 'traceplot for beta4 * gamma4')
plot(BG[,5], type = 'l', main = 'traceplot for beta5 * gamma5')

res = rbind(apply(Beta, 2, effectiveSize),
            apply(BG, 2, effectiveSize))
rownames(res) = c('beta', 'beta \\* gamma')
res %>%
  kable(col.names = 1:5,
        caption = 'Effective Sample Size out of 10000')
```

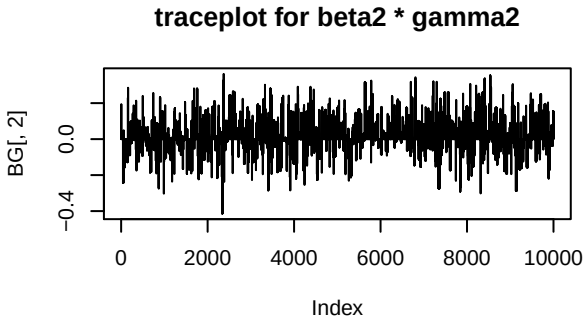
Table 3: Effective Sample Size out of 10000

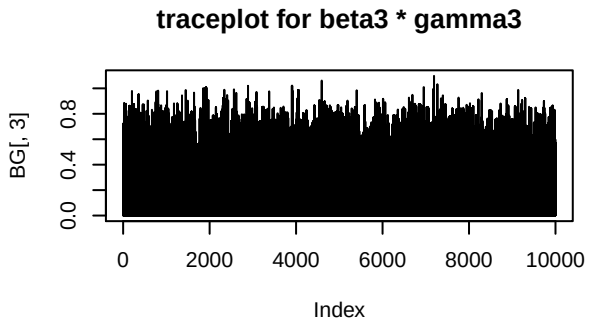
	1	2	3	4	5
beta	417.4036	526.804	637.098	841.6183	366.4293
beta * gamma	3830.2895	1854.689	6809.591	7505.4934	4437.5379

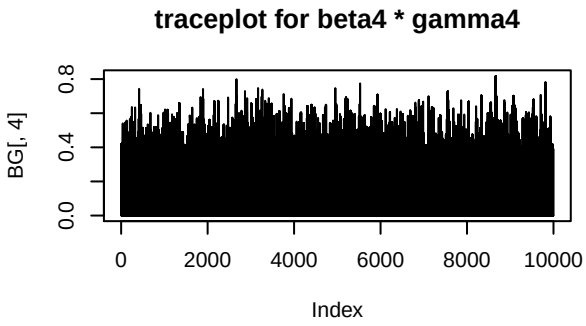
```
top5 = do.call(paste0, as.data.frame(Gamma)) %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  .[1:5]
top5/10000
```

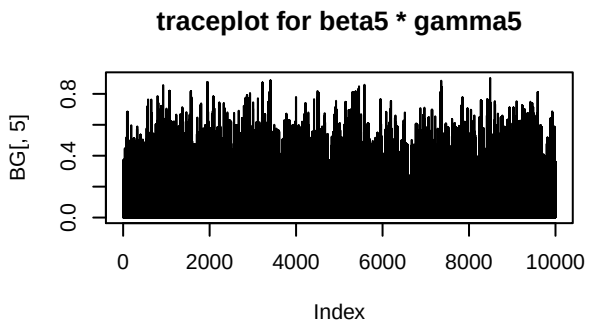
```
## .
##  00000  00100  00001  00010  01000
## 0.1753 0.1016 0.0781 0.0710 0.0618
```











Judging from the traceplot and effective sample size, we know it has similar mixing as the previous approach.

```
top10 = do.call(paste0, as.data.frame(Gamma)) %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  .[1:10]
top10/10000
```

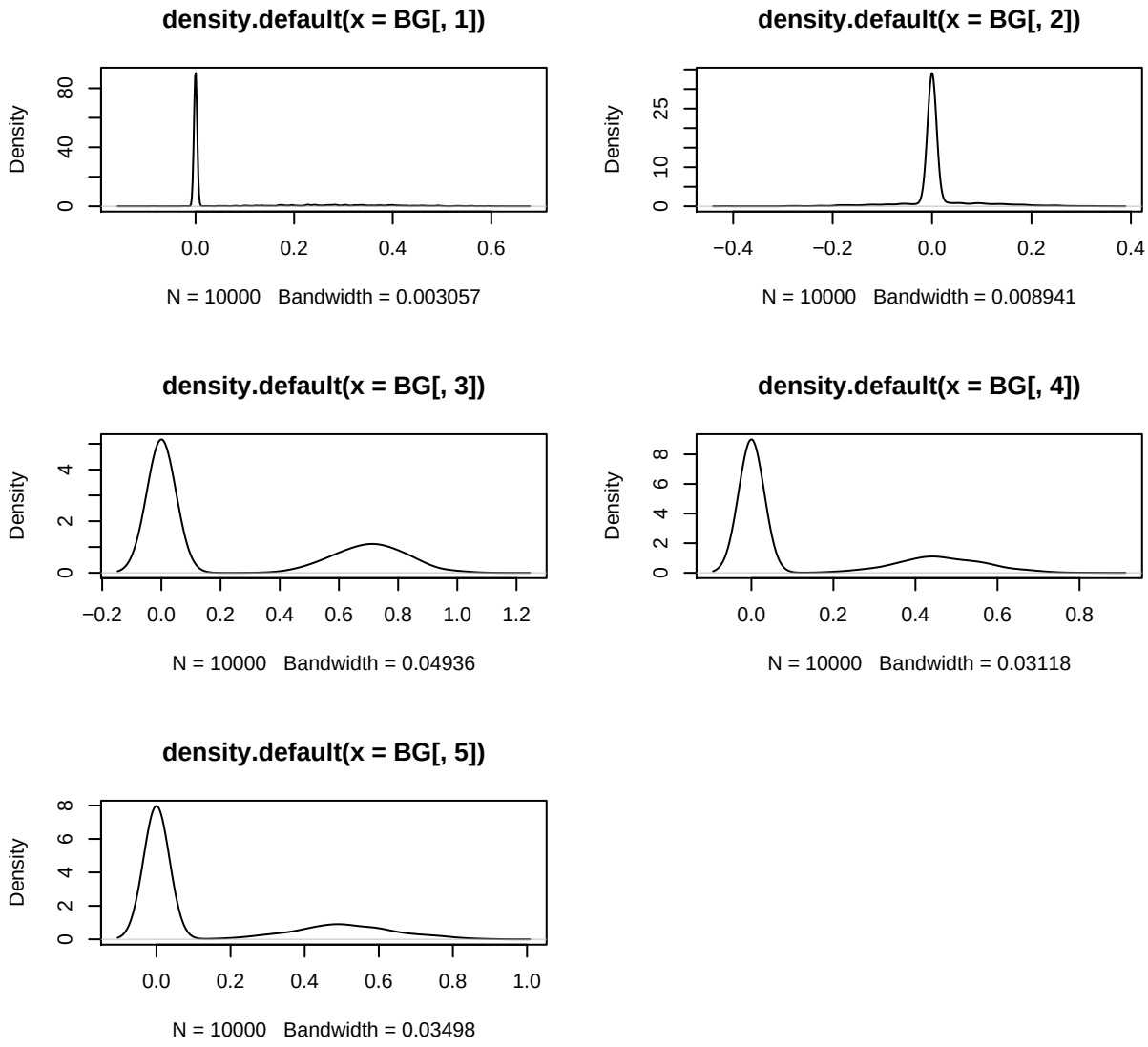
```
## .
##  00000  00100  00001  00010  01000  10000  00101  00110  10100  00011
## 0.1753 0.1016 0.0781 0.0710 0.0618 0.0597 0.0424 0.0422 0.0357 0.0316
```

We can see the Mixture Prior Approach is very likely to choose only one variable, or even none at all. But if we choose to check the top 10 most popular γ, and we may find some useful information: most popular models contain x_3. So probably it's a very important variable.

Also, we can find that no single model is dominating under this approach.

```
par(mfrow = c( 3,2))
plot(density(BG[,1]))
plot(density(BG[,2]))
plot(density(BG[,3]))
plot(density(BG[,4]))
plot(density(BG[,5]))
res = cbind(
  c(
    mean(BG[,1]),
    mean(BG[,2]),
    mean(BG[,3]),
    mean(BG[,4]),
    mean(BG[,5])
  ),
  c(
    mean(Gamma[,1]),
    mean(Gamma[,2]),
    mean(Gamma[,3]),
    mean(Gamma[,4]),
    mean(Gamma[,5])
  )
)
rownames(res) = paste('beta', 1:5, 'gamma', 1:5)
colnames(res) = c('posterior mean', 'Pr(gamma_j = 1|x,y)')

kable(res)
```

	posterior mean	Pr(gamma_j = 1|x,y)
beta 1 gamma 1	0.0745605	0.2555
beta 2 gamma 2	0.0062436	0.2457
beta 3 gamma 3	0.2539379	0.3597
beta 4 gamma 4	0.1363311	0.2954
beta 5 gamma 5	0.1528321	0.3003



The $Pr(\gamma_j = 1|x, y)$ is not so high as in previous approach, which means generally we tend to choose less variabls and keep our model simple.