STA250 Homework 1

Shuyang Ling

October 28, 2013

1 Acknowledgement

Many thanks for the help provided by Minjie Fan, Shuyang Li and Ji Hao. Especially their help in showing me how to use Gauss and illuminating discussion in algorithm design.

2 Problem 1

2.1 Problem 1a

Prove that Gibbs sampling algorithm works for $p(x_1, x_2)$

Proof. It suffices to show that

$$\iint p(x_1, y_1) K((x_1, y_1), (x_2, y_2)) dx_1 dy_1 = p(x_2, y_2)$$

which means $p(x_1, x_2)$ is the stationary distribution of the Markov kernel generated from Gibbs sampling algorithm.

Recall Gibbs sampling algorithm, in each loop:

1. $x_2 \sim p(x|y_1)$
2. $y_2 \sim p(y|x_2)$

That means

$$\begin{aligned} K((x_1, y_1), (x_2, y_2)) &= p(x_2, y_2|x_1, y_1) \\ &= p(y_2|x_1, x_2, y_1) p(x_2|x_1, y_1) \\ &= p(y_2|x_2) p(x_2|y_1) \end{aligned}$$

Substitute the above result into the double integral:

$$\begin{aligned}\iint p(x_1,y_1)K((x_1,y_1),(x_2,y_2))dx_1y_1 &= \iint p(x_1,y_1)p(y_2|x_2)p(x_2|y_1)dx_1dy_1\\ &= \iint p(x_1,y_1)\frac{p(x_2,y_2)}{p(x_2)}\frac{p(x_2,y_1)}{p(y_1)}dx_1dy_1\\ &= p(x_2,y_2)\iint \frac{p(x_1,y_1)}{p(y_1)}\frac{p(x_2,y_1)}{p(x_2)}dx_1dy_1\\ &= p(x_2,y_2)\int p(y_1|x_2)dy_1\int p(x_1|y_1)dx_1\\ &= p(x_2,y_2)\end{aligned}$$

The above derivation implies that $p(x,y)$ is the stationary distribution of Markov Process. With the assumption that it is ergodic, it implis that Gibbs sampler converges.

□

2.2 Problem 1b

The proof for p- dimension case is similar. We would like to prove the following identity,

$$p(y_1,\dots,y_p)=\int\cdots\int_p p(x_1,\dots,x_p)K\Big((x_1,\dots,x_p),(y_1,\dots,y_p)\Big)dx_1\cdots dx_p$$

To simplify the notation, we just write $\int$ instead of $\int\cdots\int$. And also write the Markov kernel into

$$K\Big((x_1,\dots,x_p),(y_1,\dots,y_p)\Big)=p(y_1|\dots)p(y_2|\dots)\dots p(y_p|\dots)dx_1\dots dx_p$$

Every term on the right side is the standard conditional distribution in Gibbs algorithm.

Proof.

$$\begin{aligned} & \int p(x_1,\dots,x_p)p(y_1|\dots)p(y_2|\dots)\dots p(y_p|\dots)dx_1\dots dx_p\\ = & \int \Big[p(y_2|\dots)\dots p(y_p|\dots)\Big]p(x_1,\dots,x_p)\frac{p(y_1,x_2,\dots,x_p)}{p(x_2,\dots,x_p)}dx_1\dots dx_p\\ = & \int \Big[p(y_2|\dots)\dots p(y_p|\dots)\Big]p(y_1,x_2,\dots,x_p)dx_2\dots dx_p\int p(x_1|x_2,\dots,x_p)dx_1\\ = & \int \Big[p(y_2|\dots)\dots p(y_p|\dots)\Big]p(y_1,x_2,\dots,x_p)dx_2\dots dx_p\\ = & \cdots \text{By induction}\\ = & \int \Big[p(y_k|\dots)\dots p(y_p|\dots)\Big]p(y_1,\cdots,y_{k-1},x_k,\dots,x_p)dx_k\dots dx_p\\ = & \; p(y_1,y_2,\dots,y_p)\end{aligned}$$

$\square$

3 Problem 2

3.1 Problem 2a

Write down the posterior distribution for β up to proportionality.

$$y_i|\beta \sim Bin(m_i, logit^{-1}(x_i^T\beta), \quad i = 1, \ldots, n.$$

So define

$$p_i(\beta) = \frac{\exp{(x_i^T\beta)}}{1 + \exp{(x_i^T\beta)}}$$

The conditional distribution of $\vec{y}$ given β is

$$p(y_1, \ldots, y_n|\beta) \propto \prod_{i=1}^{n} p_i(\beta)^{y_i}(1 - p_i(\beta))^{m_i - y_i}$$

Use prior normal distribution of $\beta \sim N(\mu_0, \Sigma_0)$ the posterior distribution could be written as

$$p(\beta|\vec{y}) \propto \exp[-\frac{1}{2}(\beta - \mu_0)^T\Sigma_0^{-1}(\beta - \mu_0)] \prod_{i=1}^{n} p_i(\beta)^{y_i}(1 - p_i(\beta))^{m_i - p_i(\beta)}$$

Assign the values of parameters as $p = 2$, $\mu = (0, 0)^T$, $\Sigma_0 = diag(1, 1)$. The proposal distribution is

$$q(\alpha|\beta) = \exp[-\frac{1}{2v^2}|\alpha - \beta|^2], \quad v \text{ is a parameter for retune}$$

Metropolis-Hastings

1. Generate $\alpha \sim q(\alpha|\beta^t)$
2. - Take $\beta^{t+1} = \alpha$ with probability $\rho(\beta^t, \alpha)$.
   - Take $\beta^{t+1} = \beta^t$ with probability $1 - \rho(\beta^t, \alpha)$
3. where

$$\rho(\beta^t, \alpha) = \min\{1, \frac{p(\alpha|y)}{p(\beta^t|y)}\}$$

$$\frac{p(\alpha|y)}{p(\beta^t|y)} = \exp(-\frac{1}{2}(|\alpha|^2 - |\beta|^2)) \prod_{i=1}^{n} (\frac{p_i(\alpha)}{p_i(\beta^t)})^{y_i} (\frac{1 - p_i(\alpha)}{1 - p_i(\beta^t)})^{n_i - y_i}$$

Question

3.2 What proposal distribution did you use?

I use multivariate Gaussian random variable

$$q(\alpha|\beta) = \exp[-\frac{1}{2v^2}|\alpha - \beta|^2], \quad v \text{ is a parameter for retune}$$

3.3 How is the tune process?

The criteria on when to retune a process is based on the acceptance rate after retune= 100 iterations. After that,

- If acceptance rate $\leq 30\%$, then set $v/2$ as the new parameter
- If acceptance rate $\geq 60\%$, then set $2v$ as the new parameter

3.4 How many iterations did you run? Burnin period?

I run 11000 iterations including 1000 Burnin iterations and 10000 niter iterations.

3.5 Starting values

I set starting values as $(0, 0)$

3.6 Numerical Coverge Properties

3.6.1 coverage summaries.txt

```
     beta_0 beta_1
p_01  0.005  0.005
p_05  0.065  0.030
p_10  0.115  0.070
p_25  0.245  0.190
p_50  0.520  0.485
p_75  0.740  0.745
p_90  0.880  0.875
p_95  0.945  0.950
p_99  0.990  1.000
```

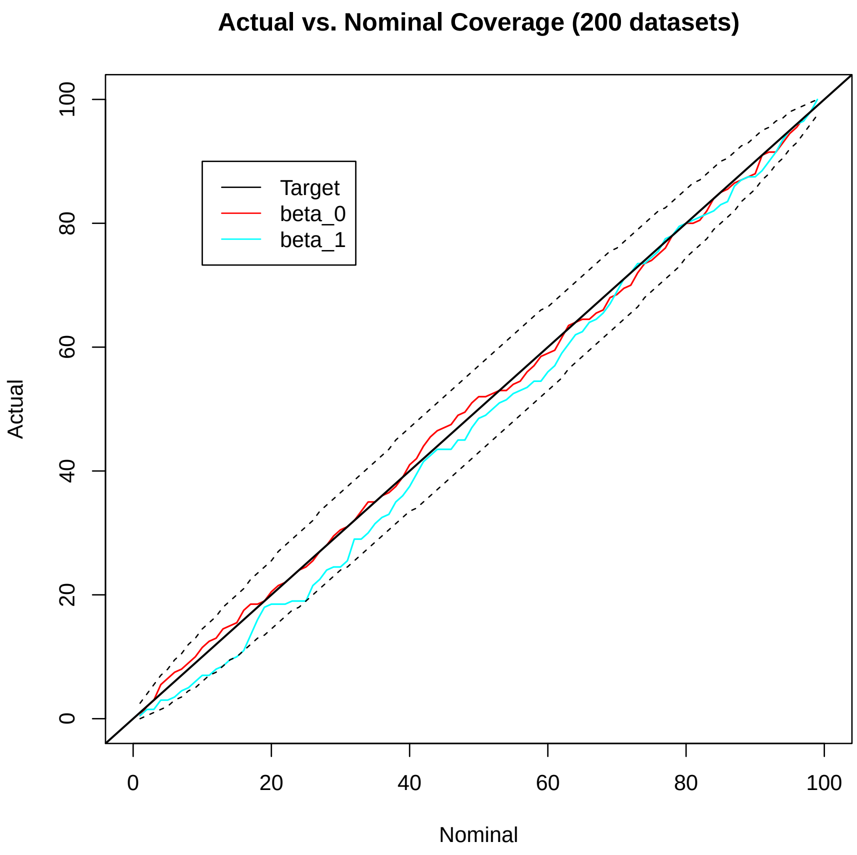

Figure 1: Coverage Line Plot

	beta_0	beta_1
p_01	0.01	0.01
p_05	0.07	0.03
p_10	0.12	0.07
p_25	0.24	0.19
p_50	0.52	0.48
p_75	0.74	0.74
p_90	0.88	0.88
p_95	0.94	0.95
p_99	0.99	1.00

3.6.2 coverage summaries.tex

3.6.3 coverage line plot.pdf

4 Problem 3

4.1 Problem 3a

Let

$$p_i(\beta) = \frac{\exp(x_i^T\beta)}{1+\exp(x_i^T\beta)}$$

The response vector is $\vec{y}$. With initial distribution of $\beta \sim \Sigma(0, \text{diag}(1000, \dots, 1000))$. So the posterior distribution is

$$p(\beta|y) \propto \exp(-\frac{1}{2000}\beta^T\beta)\prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$$

I use the most simply proposal distribution,

$$p(\alpha|\beta) \propto \exp(\frac{1}{2v^2}\|\beta-\alpha\|_2^2)$$

And update retuning parameter with the same rule as above. The result with 2000 burnin iterations and 10000 niter iterations seems unsatisfactory. The effective size for each parameter is as follows:

β_0	β_1	β_2
174.110623	9.496031	102.789437
β_3	β_4	β_5
162.506816	351.517212	285.155245
β_6	β_7	β_8
1.672261	2.473896	470.785473
β_9	β_{10}	
1585.541215	1473.690954	

I also use Metropolis Hasting to update each variable respectively and use acceptance rate for each variable to update the covariance matrix in proposal distribution. But it seems that results are not good at well. There must be some bugs inside. Other people say they could improve the performance by generalized linear model to estimate covariance matrix. But I have no background knowledge in that part... So finally, I don't know how to solve this problem...

5 Code for Problem 2

```
##
#
# Logistic regression
#
# Y_{i} | \beta \sim \textrm{Bin}\left(n_{i},
# e^{x_{i}^{T}\beta}/(1+e^{x_{i}^{T}\beta})\right)
# \beta \sim N\left(\beta_{0},\Sigma_{0}\right)
#
##

library(MASS)
library(coda)

#########################################
#########################################
## Handle batch job arguments:

# 1-indexed version is used now.
args <- commandArgs(TRUE)

cat(paste0("Command-line arguments:\n"))
print(args)

####
# sim_start ==> Lowest simulation number to be analyzed by this particular batch jo
###

#######################
sim_start <- 1000
length.datasets <- 200
#######################

if (length(args)==0){
  sinkit <- FALSE
  sim_num <- sim_start + 1
  set.seed(1330931)
} else {
  # Sink output to file?
  sinkit <- TRUE
  # Decide on the job number, usually start at 1000:
  sim_num <- sim_start + as.numeric(args[1])
  # Set a different random seed for every job number!!!
  set.seed(762*sim_num + 1330931)
}

# Simulation datasets numbered 1001-1200

########################################
########################################



log.eva = function(X , Y , N , coef ){
  # return the log value of posterior given beta = coef
  # coef1: newly updated value, coef2: current value
  p = exp( X%*%coef )
  p = p/(1 + p)

  value = sum ( Y*log( p ) + ( N - Y )*log( 1 - p ) ) - 0.5 * sum(coef^2)
  return(value)
}


bayes.logreg <- function(n,y,X,beta.0,Sigma.0.inv,niter=10000,burnin=1000,
                           print.every=1000,retune=100,verbose=TRUE)
{
  num.it = niter + burnin
  beta = matrix(0, 2, num.it)
  v = 1
  accept.count = 0
  for (i in 2:num.it){
    # The candidator

    alpha = matrix(rnorm(2)*v,2,1) + beta[ ,i-1]
    temp.value1 = log.eva(X = X, Y = y, N = n, coef = as.matrix(alpha) )

    temp.value2 = log.eva(X = X, Y = y, N = n, coef = as.matrix(beta[,i-1]) )
    # Compute the ratio
    rho = min( exp( temp.value1 - temp.value2 )  ,1)

    flag = runif(1)
    beta[,i] = beta[,i-1] + (alpha - beta[,i-1]) * ( flag < rho)

    if ( flag<rho & i<=burnin){
      accept.count = accept.count + 1
    }
    if (i %% retune == 0 & i<=burnin){
      accept.rate = accept.count/retune
      #print(c(accept.rate,v))
      if (accept.rate<=0.3){
        v = v/2
      }
      if (accept.rate>=0.6)
        v = 2*v
      }
      accept.count = 0

    }
  }
  return(beta)
}

#################################################
# Set up the specifications:
p=2
beta.0 <- matrix(c(0,0))
Sigma.0.inv <- diag(rep(1.0,p))
niter <- 10000
# etc... (more needed here)
#################################################

# Read data corresponding to appropriate sim_num:

filename = paste("data/blr_data_", as.character(sim_num), ".csv",sep='')
#filename = paste('blr_data_', sim_num, '.csv',sep='')
data.set = read.csv(filename)

# Extract X and y:
x1 = data.set$X1
x2 = data.set$X2
X = cbind(x1,x2)
y = data.set$y
n = data.set$n
# Fit the Bayesian model:
result = bayes.logreg(n = n,y = y, X = X, beta.0 = beta.0, Sigma.0.inv = Sigma.0.In
# Extract posterior quantiles...
test1 = result[1,1001:11000]
test2 = result[2,1001:11000]
```

```
percent.beta1 = sapply(seq(from = 0.01, to = 0.99, by =0.01), function(x) quantile(t
percent.beta2 = sapply(seq(from = 0.01, to = 0.99, by =0.01), function(x) quantile(t

quantile.beta = cbind(percent.beta1, percent.beta2)

# Write results to a (99 x p) csv file...

result.filename = paste("results/blr_res_", as.character(sim_num), ".csv",sep='')
#result.filename = paste('blr_result_', sim_num, '.csv',sep='')

write.table(quantile.beta, result.filename, row.names=FALSE, col.names=FALSE, na="",

# Go celebrate.

cat("done. :)\n"
```

6 Code for Problem 3

```
bc.data = read.table('breast_cancer.txt')
names(bc.data) = c('area', 'compactness', 'concavepts', 'concavity',
                    'fracdim', 'perimeter','radius', 'smoothness',
                    'symmetry', 'texture', 'diagnosis')
bc.data = bc.data[-1,]
bc.res = matrix(1, nrow(bc.data),1)
bc.res[bc.data$diagnosis == 'B'] = 0
bc.data$response = bc.res
design.m = as.matrix(bc.data[1:nrow(bc.data),1:10])
design.m = as.numeric(unlist(design.m))
design.m = matrix(design.m, nrow(bc.data),10)

design.m = cbind(matrix(1, 569, 1), design.m)

y = bc.res

# 1: Bad, 0: Good mvtnorm

niter = 102000
burnin = 2000
retune = 100
```

```
design.mean = sapply(c(1:11), function(x) mean(design.m[,x]))
design.sd = sapply(c(1:11), function(x) sd(design.m[,x]))

norm.design = sapply(c(2:11), function(x)
             (design.m[,x]-design.mean[x])/design.sd[x] )
norm.X = cbind(matrix(1,569,1), norm.design)

X = norm.X

fun.eva = function( coef ){
  # return the log value of posterior given beta = coef
  # coef1: newly updated value, coef2: current value
  prob = exp( X%*%coef )
  prob = prob/(1 + prob)
  index = which( prob>10e-6 & abs(prob-1)> 10e-6 )
  value = exp(sum ( y[index]*log( prob[index] ) +
   ( 1 - y[index] )*log( 1 - prob[index] ) ) - 5e-4 * sum(coef^2) )

  # value = log( prod(prob^y * (1 - prob)^(1-y) ))  - 5e-4*sum(coef^2)

  return(value)
}



num.it = niter + burnin
beta = matrix(0, 11, num.it)
v = 1

# The counting number of acceptance in each retune period
accept.count = 0

for (i in 2:num.it){

  alpha = beta[,i-1] + v*matrix(rnorm(11),11,1)

  rho = min(  fun.eva(alpha)/fun.eva(beta[,i-1]) , 1)

  flag = runif(1)
  # If accepted, the next current value is the now updated value.
  # Otherwise the same.

  beta[,i] = beta[,i-1] + (alpha - beta[,i-1]) * ( flag < rho)
  if ( flag<rho & i<=burnin){
    accept.count = accept.count + 1
  }
  if (i %% retune == 0 & i<=burnin){
    accept.rate = accept.count/retune

    #print(c(accept.rate,v))

    if (accept.rate<=0.3){
      v = v/2
    }
    if(accept.rate>=0.6){
      v = 2*v
    }
    accept.count = 0

  }
}


library(lattice)
library(coda)
sapply(c(1:11), function(x) effectiveSize(beta[x, burnin:num.it]))


#effectiveSize(test1)

#densityplot(test1)
```