April 7

Reese Guo

4/6/2020

Important Sampling

```r
library(ggplot2)
CEdata <- read.csv("CEdata.csv")
CEdata$LogIncome <- log(CEdata$Income)
CEdata$LogExpenditure <- log(CEdata$Expenditure)
```

```r
require(runjags)
require(coda)

modelString <-"
model {
## sampling
for (i in 1:N){
y[i] ~ dnorm(beta0 + beta1*x[i], invsigma2)
}
## priors
beta0 ~ dnorm(mu0, g0)
beta1 ~ dnorm(mu1, g1)
invsigma2 ~ dgamma(a, b)
sigma <- sqrt(pow(invsigma2, -1))
}"

y <- as.vector(CEdata$LogIncome)
x <- as.vector(CEdata$LogExpenditure)
N <- length(y)

the_data <- list("y" = y, "x" = x, "N" = N,
                "mu0" = 0, "g0" = 0.0001,
                "mu1" = 0, "g1" = 0.0001,
                "a" = 1, "b" = 1)

initsfunction <- function(chain){
.RNG.seed <- c(1,2)[chain]
.RNG.name <- c("base::Super-Duper",
"base::Wichmann-Hill")[chain]
return(list(.RNG.seed=.RNG.seed,
.RNG.name=.RNG.name))
}

posterior <- run.jags(modelString,
                      n.chains = 1,
                      data = the_data,
                      monitor = c("beta0", "beta1", "sigma"),
                      adapt = 1000,
                      burnin = 5000,
                      sample = 5000,
                      thin = 50,
                      inits = initsfunction)
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Tue Apr  7 18:28:08 2020
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 994
##    Unobserved stochastic nodes: 3
##    Total graph size: 3990
## . Reading parameter file inits1.txt
## . Initializing model
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 5000
## -------------------------------------------------| 5000
## ************************************************** 100%
## . . . . Updating 250000
## -------------------------------------------------| 250000
## ************************************************** 100%
## . . . . Updating 0
## . Deleting model
## .
## Note: the model did not require adaptation
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Calculating summary statistics...
## Finished running the simulation
```

```
library(coda)
post <- as.mcmc(posterior)

synthesize <- function(X, index, n){
mean_Y <- post[index, "beta0"] + X * post[index, "beta1"]
synthetic_Y <- rnorm(n, mean_Y, post[index, "sigma"])
data.frame(X, synthetic_Y)
}

n <- dim(CEdata)[1]
m <- 20

synthetic_m <- vector("list", m)
for (l in 1:m){
synthetic_one <- synthesize(CEdata$LogExpenditure, 4980+l, n)
names(synthetic_one) <- c("logExpenditure", "logIncome_syn")
synthetic_m[[l]] <- synthetic_one
}
```

```
LogIncome_ori <- round(CEdata$LogIncome, digits = 1)
LogIncome_syn <- round(synthetic_one$logIncome_syn, digits = 1)
LogExpenditure_ori <- round(CEdata$LogExpenditure, digits = 1)
LogExpenditure_syn <- round(synthetic_one$logExpenditure, digits = 1)

compute_logsumexp <- function(log_vector){
  log_vector_max <- max(log_vector)
  exp_vector <- exp(log_vector - log_vector_max)
  sum_exp <- sum(exp_vector)
  log_sum_exp <- log(sum_exp) + log_vector_max
  return(log_sum_exp)
}

CUProb <- function(n){
  List <- list()

  G <- 11
  H <- 50
  beta0_draws <- post[1:H, "beta0"]
  beta1_draws <- post[1:H, "beta1"]
  sigma_draws <- post[1:H, "sigma"]
  for(i in 1:n){
    y_i <- LogIncome_ori[i]
    y_i_guesses <- seq((y_i - 2.5), (y_i + 2.5), 0.5)
    X_i <- LogExpenditure_syn[i]

    CU_i_logZ_all <- rep(NA, G)
    for (g in 1:G){
      q_sum_H <- sum((dnorm(y_i_guesses[g], mean = (beta0_draws + beta1_draws * X_i), sd =
                              sigma_draws)) /(dnorm(y_i, mean = (beta0_draws + beta1_draws *
                                                                   X_i), sd = sigma_draws)))
      log_pq_h_all <- rep(NA, H)
      for (h in 1:H){
        log_p_h <- sum(log(dnorm(LogIncome_syn, mean = (beta0_draws[h] + beta1_draws[h] *
                       LogExpenditure_syn), sd = sigma_draws[h])))
        log_q_h <- log(((dnorm(y_i_guesses[g],
                         mean = (beta0_draws[h] + beta1_draws[h] * X_i), sd = sigma_draws[h]))
                        sigma_draws[h]))) / q_sum_H)
        log_pq_h_all[h] <- log_p_h + log_q_h
      }
      CU_i_logZ_all[g] <- compute_logsumexp(log_pq_h_all)
    }

    prob <- exp(CU_i_logZ_all - max(CU_i_logZ_all)) / sum(exp(CU_i_logZ_all -
                                                                max(CU_i_logZ_all)))
    outcome <- as.data.frame(cbind(y_i_guesses, prob))
    names(outcome) <- c("guess", "probability")

    List[[i]] <- outcome[order(outcome$probability, decreasing = TRUE), ]
  }
  return (List)
}
```

```
Result_list <- CUProb(n)
```

```
Result_list[[7]]
```

```
##    guess probability
## 1    4.9  0.10854078
## 2    5.4  0.10437349
## 3    5.9  0.10023952
## 4    6.4  0.09625618
## 5    6.9  0.09252392
## 6    7.4  0.08912252
## 7    7.9  0.08610919
## 8    8.4  0.08351845
## 9    8.9  0.08136332
## 10   9.4  0.07963713
## 11   9.9  0.07831549
```

Project Synthesize Method

```
bnbData <- read.csv("AB_NYC_2019.csv")
bnbLength <- dim(bnbData)[1]
avail <- bnbData$availability_365
Category <- c()

for(i in 1:bnbLength){
  if (avail[i] > 330){
    Category <- c(Category, 1)
  }else if( avail[i] <= 330 & avail[i] > 270){
    Category <- c(Category, 2)
  }else if( avail[i] <= 270 & avail[i] > 60){
    Category <- c(Category, 3)
  }else{
    Category <- c(Category, 4)
  }
}

room_type <- bnbData$room_type
room_category <- c()

for(i in 1:bnbLength){
  if (room_type[i] == "Private room"){
    room_category <- c(room_category, 1)
  }else if( room_type[i] == "Entire home/apt"){
    room_category <- c(room_category, 2)
  }else if (( room_type[i] == "Shared room")){
    room_category <- c(room_category, 3)
  }else{
    room_category <- c(room_category, NA)
  }
}

neigh <- bnbData$neighbourhood_group
neigh_category <- c()
```

```
for(i in 1:bnbLength){
  if (neigh[i] == "Brooklyn"){
    neigh_category <- c(neigh_category, 1)
  }else if(neigh[i] == "Manhattan"){
    neigh_category <- c(neigh_category, 2)
  }else if(neigh[i] == "Queens"){
    neigh_category <- c(neigh_category, 3)
  }else if(neigh[i] == "Staten Island"){
    neigh_category <- c(neigh_category, 4)
  }else if(neigh[i] == "Bronx"){
    neigh_category <- c(neigh_category, 5)
  }else{
    neigh_category <- c(neigh_category, NA)
  }
}

bnbData_cat <- data.frame(bnbData, category = Category, room_category = room_category, neigh_category =
price <- bnbData$price
```

```
N <- 1000 #length of Monte Carlo
L <- dim(bnbData_cat)[1] #number of records in the database
X <- cbind(room_category, neigh_category, price, room_category*neigh_category, room_category*price, neig
X <- as.matrix(X)
y <- bnbData_cat$category
K <- 4 #number of different categories
n <- 6
z_1 <- matrix(1, nrow = 1, ncol = L)
z <- rep(NA, L)
z_matrix <- list()
g_1 <- c(-Inf, 2, 4, 6, Inf)
g_matrix <- list(g_1)
beta <- list()
```

```
library(MASS)
beta[[1]] <- mvrnorm(1, (n/(n+1)) * ginv((t(X)%*%X)) %*% t(X) %*% t(z_1), (n/(n+1)) * ginv(t(X)%*%X), t
```

```
for (j in 1:L){
    ez <- t(beta[[1]])%*%X[j, ]
    a <- max(-Inf , g_matrix[[1]][y[j]] , na.rm=TRUE)
    b <- min( g_matrix[[1]][y[j] + 1] , Inf , na.rm=TRUE)
    u <- runif(1 , pnorm( a - ez ) , pnorm(b - ez ) )
    z[j] <- ez + qnorm(u)
    if(z[j] == Inf)
      z[j] = z[j-1]
}
z_matrix[[1]] <- matrix(z, nrow = L, ncol = 1)
```

```
for(i in 2:N){
  beta[[i]] <- mvrnorm(1, n/(n+1) * ginv(t(X)%*%X) %*% t(X) %*% z_matrix[[i-1]], n/(n+1) * ginv(t(X)%*%

  z<- rep(0, L)
  for (j in 1:L){
    ez <- t (beta[[i]])%*%X[j, ]
    a <- max(-Inf , g_matrix[[i - 1]][y[j]] , na.rm=TRUE)
    b <- min( g_matrix[[i - 1]][y[j] + 1] , Inf , na.rm=TRUE)
    u <- runif(1 , pnorm( a - ez ) , pnorm(b - ez ) )
    z[j] <- ez + qnorm(u)
    if(z[j] == Inf)
      z[j] = z[j-1]
  }
  z_matrix[[i]] <- matrix(z, nrow = L, ncol = 1)

  sig <- 1
  g <- c(-Inf)
  for (k in 1:(K-1)){
    c<- max( z_matrix[[i]][y == k])
    d<- min( z_matrix[[i]][y == k+1])
    mu <- (c + d) / 2
    u<- runif ( 1 , pnorm( ( c-mu ) / sig ) , pnorm( ( d-mu ) / sig ) )
    g <- c(g, mu[k] + sig * qnorm(u))
  }
  g <- c(g, Inf)
  g_matrix[[i]] <- g

}
```

```
beta[[1000]]
```

```
## [1]  3.247326e-01  2.370869e-01 -1.240688e-04 -1.109049e-01  4.318887e-05
## [6]  3.567661e-06
```

```
g_matrix[1000]
```

```
## [[1]]
## [1]     -Inf 1.692864       NA       NA      Inf
```