CE Samples - Risk Evaluation Function

```r
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(fastDummies)
knitr::opts_chunk$set(echo = TRUE)
def.chunk.hook  <- knitr::knit_hooks$get("chunk")
knitr::knit_hooks$set(chunk = function(x, options) {
  x <- def.chunk.hook(x, options)
  ifelse(options$size != "normalsize", paste0("\\", options$size,"\n\n", x, "\n\n \\normalsize"), x)
})
require(runjags)
```

```
## Loading required package: runjags
```

```r
require(coda)
```

```
## Loading required package: coda
```

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

```r
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))
}
"
```

```r
y <- as.vector(data$LogIncome)
x <- as.vector(data$LogExpenditure)
N <- length(y)
the_data <- list("y" = y, "x" = x, "N" = N,
                 "mu0" = 0, "g0" = 0.0001,
                 "mu1" = 0, "g1" = 0.0001,
```

```r
                 "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))
}
```

```r
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 12:22:52 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...
```

```
## Warning: Convergence cannot be assessed with only 1 chain
```

```
## Finished running the simulation
```

```r
post <- as.mcmc(posterior)
```

```r
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)
}
```

```r
n <- dim(data)[1]
synthetic_one <- synthesize(data$LogExpenditure, 1, n)
names(synthetic_one) <- c("LogExpenditure", "LogIncome")
```

```r
data_org <- data[, 1:4]
data_syn <- as.data.frame(cbind(data_org[, "UrbanRural"],
                                  exp(synthetic_one
                                      [, "LogIncome"]),
                                  cbind(data_org
                                        [, c("Race",
                                             "Expenditure")])))
names(data_syn) <- c("UrbanRural", "Income",
                     "Race", "Expenditure")
```

```r
data_org$LogInc <- round(log(data_org$Income),digits=1)
data_org$LogEx <- round(log(data_org$Expenditure),digits=1)
data_syn$LogInc <- round(log(data_syn$Income),digits=1)
data_syn$LogEx <- round(log(data_syn$Expenditure),digits=1)
```

```r
N <- dim(data_org)[1]
```

```r
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)
}
```

```r
rank <- tibble()
```

```r
for (i in 1:N){
  y_i <- data_org$LogInc[i]
  y_i_stars <- seq((y_i-2),(y_i+2),0.2)
  X_i <- data_syn$LogEx[i]
  G <- length(y_i_stars)
  H <- 50
  beta0_draws <- post[1:H, "beta0"]
  beta1_draws <- post[1:H, "beta1"]
  sigma_draws <- post[1:H, "sigma"]
  CU_i_logZ_all <- rep(NA,G)
  for (g in 1:G){
    q_sum_H <- sum((dnorm(y_i_stars[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(data_syn$LogInc,
                               mean = (beta0_draws[h] + beta1_draws[h] *
                                         data_syn$LogEx),
                               sd = sigma_draws[h])))

      log_q_h <- log(((dnorm(y_i_stars[g],
                             mean = (beta0_draws[h] + beta1_draws[h] * X_i),
                             sd = sigma_draws[h])) /
            (dnorm(y_i, mean = (beta0_draws[h] + beta1_draws[h] * X_i),
                   sd = 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_stars, prob))
  names(outcome) <- c("guess", "probability")
  outcome <- outcome %>% mutate(rank = dense_rank(desc(probability)))
  rank <- rbind(rank, outcome[11,])
}
rank[1:10,]
```

```
##     guess probability rank
## 11   11.5  0.04699707   11
## 111  10.1  0.04695356   15
## 112  11.3  0.04698671   11
## 113  11.9  0.04700751   11
## 114  11.8  0.04699296   11
## 115  10.4  0.04694667   11
## 116   7.4  0.04719679   11
## 117  11.6  0.04698781   12
## 118   8.7  0.04702503   11
## 119  11.6  0.04697688   11
```

```
rank_g11 <- tibble()
for (i in 1:N){
  y_i <- data_org$LogInc[i]
  y_i_stars <- seq((y_i-2.5),(y_i+2.5),0.5)
  X_i <- data_syn$LogEx[i]
  G <- length(y_i_stars)
  H <- 50
  beta0_draws <- post[1:H, "beta0"]
  beta1_draws <- post[1:H, "beta1"]
  sigma_draws <- post[1:H, "sigma"]
  CU_i_logZ_all <- rep(NA,G)
  for (g in 1:G){
    q_sum_H <- sum((dnorm(y_i_stars[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(data_syn$LogInc,
                               mean = (beta0_draws[h] + beta1_draws[h] *
                                         data_syn$LogEx),
                               sd = sigma_draws[h])))

      log_q_h <- log(((dnorm(y_i_stars[g],
                             mean = (beta0_draws[h] + beta1_draws[h] * X_i),
                             sd = sigma_draws[h])) /
            (dnorm(y_i, mean = (beta0_draws[h] + beta1_draws[h] * X_i),
                   sd = 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_stars, prob))
  names(outcome) <- c("guess", "probability")
  outcome <- outcome %>% mutate(rank = dense_rank(desc(probability)))
  rank_g11 <- rbind(rank_g11, outcome[6,])
}
rank_g11[1:10,]
```

```
##    guess probability rank
## 6   11.5  0.08897536    6
## 61  10.1  0.08883354    9
## 62  11.3  0.08894241    6
## 63  11.9  0.08900835    6
## 64  11.8  0.08896183    6
## 65  10.4  0.08881260    7
## 66   7.4  0.08960501    7
## 67  11.6  0.08894803    8
## 68   8.7  0.08907418    6
## 69  11.6  0.08891118    6
```