Homework4

Sarah Boese

2/23/2020

```r
library(ProbBayes)
library(dplyr)
library(ggplot2)
require(gridExtra)
library(reshape)
library(runjags)
library(coda)
library(tidyverse)
library(fastDummies)
crcblue <- "#2905a1"
```

```r
CESample <- read.csv("CEsample2.csv")
```

I decided that I wanted to use all variables within CESampe logExpenditure, UrbanRural, Race) as estimators for logIncome. Thus, I used a Multilinear regression model in which I scaled Log Income and Log Expenditure by centering at 0 and dividing by standard deviation. I use the following MLR model (where * denotes a standardized continuous variable):

$$
\begin{aligned}
Y_i^* \mid \beta_0, \beta_1, \cdots, \beta_7, \sigma, \mathbf{x}_i^* \overset{ind}{\sim} \text{Normal}(\beta_0 &+ \beta_1 x^*_{i,expenditure} + \beta_2 x_{i,rural} \\
&+ \beta_3 x_{i,race_B} + \beta_4 x_{i,race_N} \\
&+ \beta_5 x_{i,race_A} + \beta_6 x_{i,race_P} \\
&+ \beta_7 x_{i,race_M}, \sigma).
\end{aligned} \tag{1}
$$

```r
CESample <- CESample %>%
  mutate(LogTotalIncome = log(TotalIncomeLastYear))
CESample <- CESample %>%
  mutate(LogTotalExp = log(TotalExpLastQ))
```

```r
CESample$Log_TotalExpSTD <- scale(CESample$LogTotalExp)
CESample$Log_TotalIncomeSTD <- scale(CESample$LogTotalIncome)
## create indictor variable for Rural
CESample$Rural = fastDummies::dummy_cols(CESample$UrbanRural)[,names(fastDummies::dummy_cols(CESample$U
 == ".data_2"]
```

```r
## create indicator variables for Black (2), Native American (3),
## Asian (4), Pacific Islander (5), and Multi-race (6)
CESample$Race_Black = fastDummies::dummy_cols(CESample$Race)[,names(fastDummies::dummy_cols(CESample$Ra
CESample$Race_NA = fastDummies::dummy_cols(CESample$Race)[,names(fastDummies::dummy_cols(CESample$Race)
CESample$Race_Asian = fastDummies::dummy_cols(CESample$Race)[,names(fastDummies::dummy_cols(CESample$Ra
CESample$Race_PI = fastDummies::dummy_cols(CESample$Race)[,names(fastDummies::dummy_cols(CESample$Race)
CESample$Race_M = fastDummies::dummy_cols(CESample$Race)[,names(fastDummies::dummy_cols(CESample$Race))
```

```
modelString <-"
model {
## sampling
for (i in 1:N){
y[i] ~ dnorm(beta0 + beta1*x_exp[i] + beta2*x_rural[i] +
beta3*x_race_B[i] + beta4*x_race_N[i] +
beta5*x_race_A[i] + beta6*x_race_P[i] +
beta7*x_race_M[i], invsigma2)
}
## priors
beta0 ~ dnorm(mu0, g0)
beta1 ~ dnorm(mu1, g1)
beta2 ~ dnorm(mu2, g2)
beta3 ~ dnorm(mu3, g3)
beta4 ~ dnorm(mu4, g4)
beta5 ~ dnorm(mu5, g5)
beta6 ~ dnorm(mu6, g6)
beta7 ~ dnorm(mu7, g7)
invsigma2 ~ dgamma(a, b)
sigma <- sqrt(pow(invsigma2, -1))
}
"
```

- Pass the data and hyperparameter values to JAGS:

```
y_income = as.vector(CESample$LogTotalIncome)
x_exp = as.vector(CESample$LogTotalExp)
x_rural = as.vector(CESample$Rural)
x_race_B = as.vector(CESample$Race_Black)
x_race_N = as.vector(CESample$Race_NA)
x_race_A = as.vector(CESample$Race_Asian)
x_race_P = as.vector(CESample$Race_PI)
x_race_M = as.vector(CESample$Race_M)
N = length(y_income)  # Compute the number of observations
```

- Pass the data and hyperparameter values to JAGS:

```
the_data <- list("y" = y_income, "x_exp" = x_exp,
                 "x_rural" = x_rural, "x_race_B" = x_race_B,
                 "x_race_N" = x_race_N, "x_race_A" = x_race_A,
                 "x_race_P" = x_race_P, "x_race_M" = x_race_M,
                 "N" = N,
                 "mu0" = 0, "g0" = 0.0001, "mu1" = 0, "g1" = 0.0001,
                 "mu2" = 0, "g2" = 1, "mu3" = 0, "g3" = 1,
                 "mu4" = 0, "g4" = 1, "mu5" = 0, "g5" = 1,
                 "mu6" = 0, "g6" = 1, "mu7" = 0, "g7" = 1,
                 "a" = 1, "b" = 1)
```

- Pass the data and hyperparameter values to JAGS:

```
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))
}
```

- Run the JAGS code for this model:

```
posterior_MLR <- run.jags(modelString,
                      n.chains = 1,
                      data = the_data,
                      monitor = c("beta0", "beta1", "beta2",
                                  "beta3", "beta4", "beta5",
                                  "beta6", "beta7", "sigma"),
                      adapt = 1000,
                      burnin = 5000,
                      sample = 5000,
                      thin = 50,
                      inits = initsfunction)
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Tue Feb 25 14:14:51 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: 9
##    Total graph size: 9984
## . 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
```

JAGS output for the MLR model

```
summary(posterior_MLR)
```

```
##          Lower95      Median    Upper95       Mean         SD Mode
## beta0  3.669740  4.28610000  4.9266900  4.2946630 0.32414868   NA
## beta1  0.647560  0.72123550  0.7888190  0.7206641 0.03640298   NA
## beta2 -0.336150 -0.05379105  0.2074630 -0.0543172 0.13912170   NA
## beta3 -0.348372 -0.16199450  0.0359885 -0.1631919 0.09822631   NA
## beta4 -1.264630 -0.58281600  0.0728679 -0.5837983 0.34284402   NA
## beta5 -0.149501  0.13428050  0.4534340  0.1331292 0.15589175   NA
## beta6 -1.039080 -0.33076200  0.4318230 -0.3323508 0.37281544   NA
## beta7 -0.941018 -0.49236050 -0.0524113 -0.4891195 0.22801659   NA
## sigma  0.913835  0.95479350  0.9986350  0.9551042 0.02162080   NA
##              MCerr MC%ofSD SSeff        AC.500 psrf
## beta0 0.0097435585     3.0  1107  1.657507e-02   NA
## beta1 0.0010948661     3.0  1105  1.915412e-02   NA
## beta2 0.0019674779     1.4  5000  3.204546e-03   NA
## beta3 0.0013819565     1.4  5052  1.890181e-03   NA
## beta4 0.0048485466     1.4  5000  5.349766e-03   NA
## beta5 0.0022896118     1.5  4636  3.068741e-02   NA
## beta6 0.0052724065     1.4  5000  9.152724e-03   NA
## beta7 0.0032246415     1.4  5000  1.546961e-02   NA
## sigma 0.0003057642     1.4  5000 -2.900956e-05   NA
```

```
post_MLR <- as.mcmc(posterior_MLR)
```

```
synthesize <- function(X, index, n){
  mean_Y <- post_MLR[index, "beta0"] + X$y_income * post_MLR[index, "beta1"] + X$x_rural * post_MLR[inde
  synthetic_Y <- rnorm(n,mean_Y, post_MLR[index,"sigma"])
  data.frame(X$y_income, synthetic_Y)
}
```

```
n <- dim(CESample)[1]
params <- data.frame(y_income, x_rural, x_race_B, x_race_N, x_race_A, x_race_P, x_race_M)
synthetic_one <- synthesize(params,1,n)
names(synthetic_one) <- c("LogIncome_org", "LogIncome_syn")
```

```
m <- 20
synthetic_m <- vector("list", m)
for (l in 1:m){
  params <- data.frame(y_income, x_rural, x_race_B, x_race_N, x_race_A, x_race_P, x_race_M)
  synthetic_i <- synthesize(params,4980+l,n)
  names(synthetic_i) <- c("LogIncome_org", "LogIncome_syn")
  synthetic_m[[l]] <- synthetic_i
}
```

Here I write a function to calculate analysis specific utility measures, which I will run on each synthetic data set.

```r
utilitymeasure<- function(list_i){
  exp<-mean(list_i$LogIncome_syn)
  med<-median(list_i$LogIncome_syn)
  stand<-sd(list_i$LogIncome_syn)
  pointEstAnal<- lm(CESample$LogTotalExp ~ list_i$LogIncome_syn)
  pointEst<-pointEstAnal$coefficients[1]
  unitInc<-pointEstAnal$coefficients[2]
  data.frame(exp,med,stand,pointEst,unitInc)
}
```

ASUM stands for Analysis Specific Utility Measures.

```r
asum_m<- data.frame(utilitymeasure(synthetic_m[[1]]))
names(asum_m)<-c("mean", "median", "standard_dev", "point_estimate", "unit_increase")
if(m>1){
  for (j in 2:m){
    asum_i<-utilitymeasure(synthetic_m[[j]])
    names(asum_i)<-c("mean", "median", "standard_dev", "point_estimate", "unit_increase")
    asum_m<-bind_rows(asum_m,asum_i)
  }
}
asum_m
```

```
##        mean   median standard_dev point_estimate unit_increase
## 1  11.84910 11.83972     1.305444       5.932063     0.2406901
## 2  11.88153 11.93562     1.280932       5.976175     0.2363202
## 3  11.94784 11.99617     1.324023       5.753981     0.2536058
## 4  11.97089 12.01812     1.320689       5.583272     0.2673778
## 5  12.00438 12.11602     1.297229       5.666384     0.2597083
## 6  11.89179 11.91895     1.269411       5.825632     0.2487758
## 7  11.86799 11.85585     1.268448       5.498784     0.2768150
## 8  11.94576 12.01081     1.271905       5.722889     0.2562526
## 9  11.90847 11.92977     1.318916       5.764561     0.2535558
## 10 11.84804 11.91408     1.232910       5.726666     0.2580473
## 11 11.85281 11.90632     1.221803       5.454562     0.2809006
## 12 11.93360 12.02291     1.257350       5.574590     0.2689408
## 13 11.84667 11.88360     1.301429       6.108460     0.2258493
## 14 11.97612 12.02908     1.288296       5.392621     0.2831803
## 15 11.89510 11.92999     1.331687       5.611213     0.2667325
## 16 11.85142 11.87861     1.281479       5.749847     0.2560180
## 17 11.96110 11.97159     1.255948       5.818081     0.2479657
## 18 11.92738 11.91579     1.283672       5.616151     0.2655965
## 19 11.93186 11.97143     1.260838       5.540059     0.2718740
## 20 12.00961 12.10996     1.316495       5.795065     0.2488804
```

Here I create the calcQ function which calculates the approxamtion for mean and variance of all 20 synthesized data sets.

```r
calcQ<-function(list_q, list_u, m){
  qm_bar<-sum(list_q)/m
  bm<-0
  for(i in 1:m){
```

```r
    bm = bm + (list_q[i]-qm_bar)^2/(m-1)
  }
  um_bar<-sum(list_u)/m
  Tp<-bm/m+um_bar
  return(c(qm_bar, Tp))
}
inferences<- calcQ(asum_m$mean, (asum_m$standard_dev)^2,m)
inferences<-data.frame(inferences[1], inferences[2])
names(inferences)<-c("mean", "variance")
inferences
```

```
##       mean variance
## 1 11.91507 1.650822
```