SynthesisModel

Sarah Boese

2/6/2020

```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 = 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)  # Compute the number of observations
```

- Pass the data and hyperparameter values to JAGS:

```
the_data <- list("y" = y, "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 Sun Feb 23 12:29:02 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

```r
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
```

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

```r
synthesize <- function(X, index, n){
  synth_Y=vector(mode="numeric", length = n)
  for(i in 1:n){
  mean_Y <- post_MLR[index, "beta0"] + X[i,1] * post_MLR[index, "beta1"] + X[i,2] *post_MLR[index, "beta
    X[i,3]*post_MLR[index, "beta3"] + X[i,4] *post_MLR[index, "beta4"] + X[i,5] *post_MLR[index, "beta5
    X[i,6]*post_MLR[index, "beta6"] + X[i,7] *post_MLR[index, "beta7"]
  synth_Y[i]<- rnorm(1, mean_Y, post_MLR[index, "sigma"])
  }
  synthetic_frame<-as.data.frame(X, row.names = NULL, optional = FALSE)
  synthetic_frame<-add_column(synthetic_frame, synth_Y)
  return(synthetic_frame)
}
```

```r
n <- dim(CESample)[1]
matrix_of_X<-matrix(nrow=n, ncol=7)
matrix_of_X[,1]<-as.vector(CESample$LogTotalExp)
matrix_of_X[,2]<-as.vector(CESample$Rural)
matrix_of_X[,3]<-as.vector(CESample$Race_Black)
matrix_of_X[,4]<-as.vector(CESample$Race_NA)
matrix_of_X[,5]<-as.vector(CESample$Race_Asian)
matrix_of_X[,6]<-as.vector(CESample$Race_PI)
matrix_of_X[,7]<-as.vector(CESample$Race_M)
```

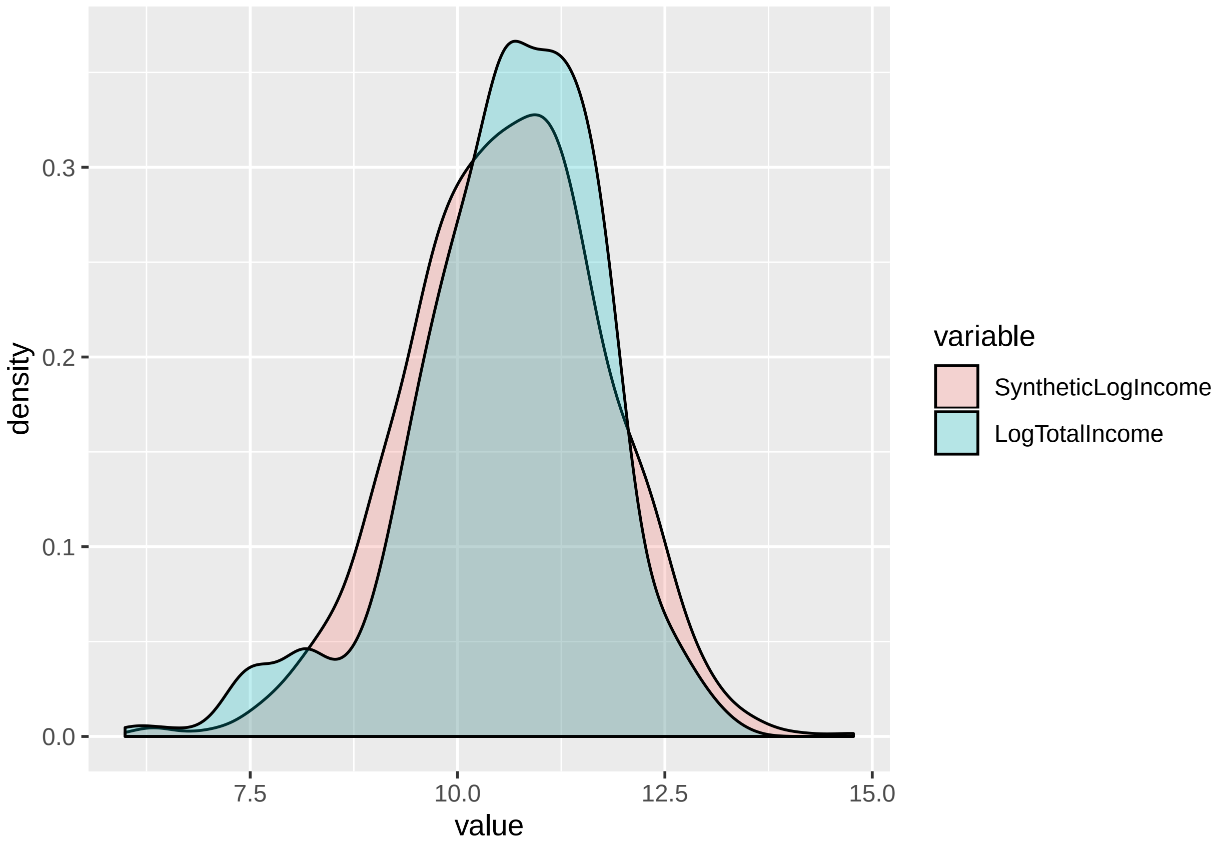
```r
synthetic_new <- synthesize(matrix_of_X, 1, n)
names(synthetic_new) <- c("logExpenditure", "Rural", "Black", "Native American", "Asian", "Pacific Islar
```

```r
SyntheticData <- data.frame(synthetic_new$logIncome_syn, CESample$LogTotalIncome)
names(SyntheticData) = c("SyntheticLogIncome", "LogTotalIncome")
```

```r
data<- melt(SyntheticData)
```

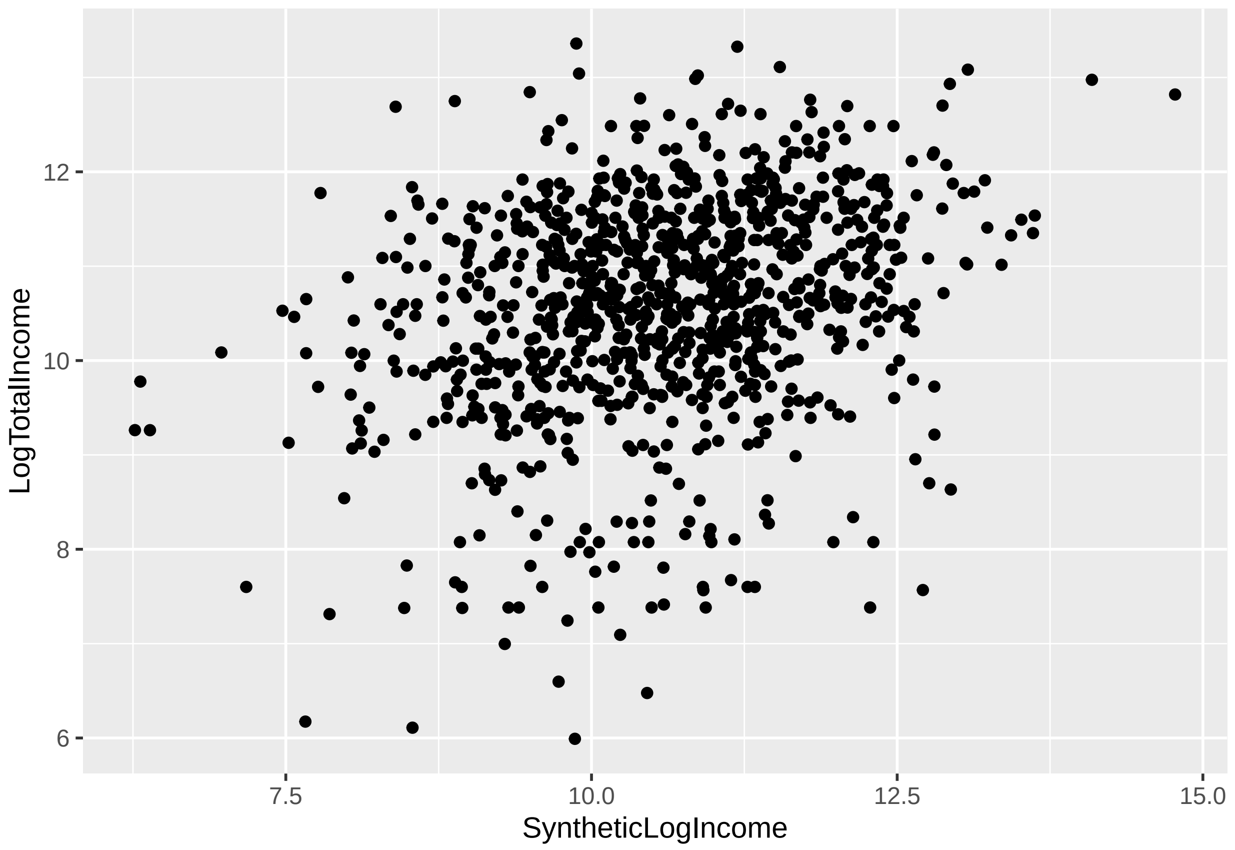
```
## Using  as id variables
```

```r
ggplot(data,aes(x=value, fill=variable)) + geom_density(alpha=0.25)
```



Here we can see the effect our model has on the data. The distribution curve, while similar to the data, looks to follow a normal curve more faithfully.

```r
ggplot(SyntheticData, aes(x=SyntheticLogIncome, y=LogTotalIncome)) +
  geom_point()
```



```
summary = summarize_all(SyntheticData, .funs=c(mean, median))
names(summary) = c("SyntheticLogIncomeMean", "SyntheticLogIncomeMedian","LogTotalIncomeMean", "LogTotal
summary
```

```
##   SyntheticLogIncomeMean SyntheticLogIncomeMedian LogTotalIncomeMean
## 1               10.58639                 10.59507           10.61863
##   LogTotalIncomeMedian
## 1             10.70574
```

These look farily close to each other to me.

```
sd(SyntheticData$LogTotalIncome)
```

```
## [1] 1.153488
```

```
sd(SyntheticData$SyntheticLogIncome)
```

```
## [1] 1.175373
```

For the point estimates, with an expenditure of 0, unsynthesize income will be 4.3219 and synthetic income will be 4.7681. And with every unit increase of expenditure, the unsynthesized income increases by 0.4211 and the synthetic income increases by 0.3782. Of course, all of these values are in log form.

```r
lm(CESample$LogTotalExp ~ CESample$LogTotalIncome)
```

```
##
## Call:
## lm(formula = CESample$LogTotalExp ~ CESample$LogTotalIncome)
##
## Coefficients:
##             (Intercept)  CESample$LogTotalIncome
##                  4.3219                   0.4211
```

```r
lm(CESample$LogTotalExp ~ SyntheticData$SyntheticLogIncome)
```

```
##
## Call:
## lm(formula = CESample$LogTotalExp ~ SyntheticData$SyntheticLogIncome)
##
## Coefficients:
##                      (Intercept)  SyntheticData$SyntheticLogIncome
##                           4.5472                            0.4002
```

To run the propensity score measure, I must first stack my synthetic data and the original data with variable T as indicator for synthetic data. Here, T == 1 if LogIncome is synthetic.

```r
synthetic_new_T<-data.frame(synthetic_new$logIncome_syn, integer(length=994) +1, integer(length=994))
names(synthetic_new_T) = c("LogIncome", "T", "T_inv")

original_T<-data.frame(CESample$LogTotalIncome, integer(length=994), integer(length=994)+1)
names(original_T)= c("LogIncome", "T", "T_inv")

merged_T<- bind_rows(synthetic_new_T, original_T)
```

Now I can run a logistic regression on the stacked data. I could not figure out how to run the models described in Woo et. all, so here I am using the glm function to run a simpler logistic regression:

$$log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 z_i$$

. Here the variable U_p is the propensity score given the simple legistic regression.

```r
mylogit <- glm(T ~ LogIncome, data = merged_T, family = "binomial")
coefs<-coef(mylogit)
merged_T<-mutate(merged_T, p_hat=1/(1+exp(-1*(coefs[1]+coefs[2]*LogIncome))))
merged_T<-mutate(merged_T, p_hat_c2=(p_hat-.5)^2)
U_p<-(1/(2*N))*sum(merged_T$p_hat_c2)
U_p
```

```
## [1] 3.476846e-06
```

To preform cluster analysis, I utilize the kmeans function. Here I wasn't sure what to use as the weight, but I think withinss makes sense. I wrote a funciton called cluster_func with parameter G, the number of desired clusters. I ran the cluster analysis with $5, 10$ and 20 clusters, I was not sure how to meaningfully pick the value for G.

```r
cluster_func<- function(G){
  fit <- kmeans(as.data.frame(merged_T$LogIncome, merged_T$T_inv), G) # 5 cluster solution
  size<-fit$size
  weight<-fit$withinss
  cluster_data<- data.frame(merged_T, fit$cluster)
  ujo_data<- cluster_data %>%
    group_by(fit.cluster) %>%
    summarize(sum(T_inv))
  ujo<-as.vector(ujo_data$`sum(T_inv)`)
  c<-1/2
  U_c<-0
  for(i in 1: G){
     U_c=U_c+weight[i]*(ujo[i]/size[i]-c)
  }
  U_c=U_c*(1/G)
  return(U_c)
}

U_c5<-cluster_func(5)
```

```
## Warning in as.data.frame.numeric(merged_T$LogIncome, merged_T$T_inv):
## 'row.names' is not a character vector of length 1988 -- omitting it. Will
## be an error!
```

```r
U_c5
```

```
## [1] -0.2097366
```

```r
U_c10<-cluster_func(10)
```

```
## Warning in as.data.frame.numeric(merged_T$LogIncome, merged_T$T_inv):
## 'row.names' is not a character vector of length 1988 -- omitting it. Will
## be an error!
```

```r
U_c10
```

```
## [1] 0.05904401
```

```r
U_c20<-cluster_func(20)
```

```
## Warning in as.data.frame.numeric(merged_T$LogIncome, merged_T$T_inv):
## 'row.names' is not a character vector of length 1988 -- omitting it. Will
## be an error!
```

```r
U_c20
```

```
## [1] 0.05986229
```

```
ecdf_orig <- ecdf(SyntheticData[,"SyntheticLogIncome"])
ecdf_syn <- ecdf(SyntheticData[,"LogTotalIncome"])
percentile_orig <- ecdf_orig(merged_T[,"LogIncome"])
percentile_syn <- ecdf_syn(merged_T[,"LogIncome"])
ecdf_diff <- percentile_orig - percentile_syn
Um <- max(abs(ecdf_diff))
Um
```

```
## [1] 0.05935614
```

```
Ua <- mean(ecdf_diff^2)
Ua
```

```
## [1] 0.001014295
```

We have that both of these values for U_m and U_a are fairly small indicating that my model has fairly high utility under both measures. However, interestingly they are larger than that which is posted on the slides for today, even though I did a multiple linear regression, which hopefully would have encoded more information than the simple linear regression. I guess this could be attributed to the pull of coeficcients that I made from the posterior or it could be a limitation of the model itself.