Data Confidentiality Lab 5

Isaac Kleisle-Murphy

March 1, 2020

```r
suppressMessages(require(dplyr))
options(scipen = 999)
acs_org = read.csv("~/Documents/Swat_2020/Data_Privacy/Data-Confidentiality/datasets/ACSdata_org.csv")
acs_syn = read.csv("~/Documents/Swat_2020/Data_Privacy/Data-Confidentiality/datasets/ACSdata_syn.csv")
acs_syn1 = read.csv("~/Documents/Swat_2020/Data_Privacy/Data-Confidentiality/datasets/ACSdata_syn2.csv"
acs_syn2 = read.csv("~/Documents/Swat_2020/Data_Privacy/Data-Confidentiality/datasets/ACSdata_syn3.csv"
#org_df = acs_org; syn_df = acs_syn; ref = "id"; c_vars = knwn
```

First, we define a helper function to compute our three risk measures:

```r
# @param: org_df, the true dataset, in a dataframe
# @param: syn_df, the synthesized dataset, in a dataframe
# @param: c_vars, a character vector of variables (in both syn_df and org_df) of items known
#                 to the intruder; these help identify c_i
# @param: ref, the string column name of the entry id reference/label in both org_df and syn_df,
#              used to pair a synthetic entry with a true entry.
# @return: a dataframe with the three variables/measures of interest.
# @dontrun: match_risk_helper(org_df = acs_org, syn_df = acs_syn, c_vars = knwn, ref = "id")

match_risk_helper <- function(org_df, syn_df, c_vars, ref){

  match_df = left_join(syn_df, org_df, by = c_vars, suffix =  c(".s", ".o"))
  match_df$is_match = match_df%>%pull(paste0(ref, ".s")) == match_df%>%pull(paste0(ref, ".o"))

  match_score = match_df%>%
    group_by_at(paste0(ref, ".s"))%>%
    summarise(t_i = sum(is_match), c_i = n())%>%
    ungroup()%>%
    mutate(k_i = ifelse(c_i*t_i==1, 1, 0),
           f_i = ifelse(c_i*(1-t_i)==1, 1, 0))%>%
    summarise(exp_match_risk = sum(t_i/c_i),
              true_match_rate = sum(k_i)/n(),
              false_match_rate = sum(f_i)/n())

  return(match_score)

}
```

Then, we run it on the data

```r
knwn = c('SEX', 'RACE', 'MAR') #known to intruder variables
syn = c('WAOB', 'DIS', 'HICOV', 'MIG', 'SCH') #synthesized variables

#give each entry an id to identify matches
```

```
acs_org$id = 1:nrow(acs_org);
acs_syn$id = 1:nrow(acs_syn)
acs_syn1$id = 1:nrow(acs_syn1)
acs_syn2$id = 1:nrow(acs_syn2)

#compute
em_scores = lapply(list(acs_syn, acs_syn1, acs_syn2), function(x)
  match_risk_helper(org_df = acs_org, syn_df = x, c_vars = knwn, ref = "id")%>%as.data.frame()
  )
```

Curiously, the measures are identical for each synthetic dataset.

```
em_scores
```

```
## [[1]]
##   exp_match_risk true_match_rate false_match_rate
## 1             57          0.0004                0
##
## [[2]]
##   exp_match_risk true_match_rate false_match_rate
## 1             57          0.0004                0
##
## [[3]]
##   exp_match_risk true_match_rate false_match_rate
## 1             57          0.0004                0
```

It's highly possible there's a bug in my function (though the results correspond to my classmates' in the last lab, so it's accurate to some degree). Another possibility is only a handful of the ACS data "known" rows are so unique that the model does not give them "new" synthesized variables; hence, the synthesis remains unchanged from the original, and tus we get these results.

First, we re-run the synthesis model for `CEsample`, just as we did in Labs 2-3.

```
suppressMessages(require(dplyr))
suppressMessages(require(ggplot2))
suppressMessages(require(runjags))
suppressMessages(require(coda))
suppressMessages(require(tidyr))
suppressMessages(require(fastDummies))
options(warn = -1)
setwd("~/Documents/Swat_2020/Data_Privacy/Data-Confidentiality/datasets")

CEsample = read.csv("CEdata.csv")%>%
  mutate(logInc = log(Income),
         logExp = log(Expenditure))


the_data = list("logInc" = CEsample$logInc,
                "logExp" = CEsample$logExp,
                "r" = CEsample$Race,
                "R" = max(CEsample$Race),
                "u" = CEsample$UrbanRural,
                "U" = max(CEsample$UrbanRural),
                "N" = nrow(CEsample),
                "mu_b0" = 0,
                "mu_b1" = 0,
                "prec_b0" = 1,
                "prec_b1" = 1
                )


the_formula = "

model{

#model
for (i in 1:N){
  logInc[i] ~ dnorm(B_0[r[i], u[i]] + B_1[r[i], u[i]]*logExp[i], inv_sigma_sq[r[i]])
}

#priors
for (rc in 1:R){
  for (ur in 1:U){
    B_0[rc, ur] ~ dnorm(b_0, tau_sq_0)
    B_1[rc, ur] ~ dnorm(b_1, tau_sq_1)
  }
  inv_sigma_sq[rc] ~ dgamma(1,1)
  sigma[rc] <- sqrt(pow(inv_sigma_sq[rc], -1))
}

#hyperpriors
b_0 ~ dnorm(mu_b0,prec_b0)
b_1 ~ dnorm(mu_b1,prec_b1)
tau_sq_0 ~ dgamma(1,1)
tau_sq_1 ~ dgamma(1,1)
}

"




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_jags  <- run.jags(the_formula,
                            n.chains = 2,
                            data = the_data,
                            monitor = c("B_0", "B_1", "sigma"),
                            adapt = 1000,
                            burnin = 5000,
                            sample = 2500,
                            thin = 100,
                                 inits = initsfunction)
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Tue Mar 24 11:36:12 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: 34
##    Total graph size: 6019
## . Reading parameter file inits1.txt
## . Reading parameter file inits2.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...
## Calculating the Gelman-Rubin statistic for 30 variables....
## Finished running the simulation
```

```
#extract, store in dictionary for quicker retrieval
posterior_df = data.frame(as.mcmc(posterior_jags))
posterior_list = lapply(colnames(posterior_df), function(x) posterior_df%>%pull(x))%>%
  `names<-`(colnames(posterior_df))
```

As before, we also re-initialize our synthesis helper functions. Note that `synth_helper()` generates a single synthetic dataset, while

```
synth_helper <- function(posterior_list, df, ii = NULL){

  #randomly select posterior draw to use

  if (is.null(ii)){
    ii = sample(1:nrow(df), 1)
  }


  beta_suffixes = paste0(".", df$Race, ".", df$UrbanRural, ".")
  sigma_suffixes = paste0(".", df$Race, ".")

```

```r
  b0 = sapply(beta_suffixes, function(x) posterior_list[[paste0("B_0", x)]][ii])%>%as.vector()
  b1 = sapply(beta_suffixes, function(x) posterior_list[[paste0("B_1", x)]][ii])%>%as.vector()


  mu_post = b0+b1*df%>%pull(logExp)
  sig_post = sapply(sigma_suffixes, function(x) posterior_list[[paste0("sigma", x)]][ii])%>%as.vector()

  result = rnorm(nrow(df), mu_post, sig_post)

  return(result)
}


synth_logInc <- function(posterior_list, df, m=1){

  set.seed(4*m-1)
  sample_idx = sample(1:length(posterior_list[[1]]), m, replace = F)

  lapply(sample_idx,
         function(y)
           synth_helper(posterior_list, df, ii = y)
         )%>%
    return()
}
```

Using these functions, we take $m = 1$ synthetic draws of `logInc`.

```r
set.seed(2020)
synth_draw = synth_logInc(posterior_list, CEsample, m = 1)
CEsample.s = CEsample; CEsample.s$logInc = synth_draw[[1]]
```

My utility measure is pretty straightforward. Consider the synthesized values, $y_1^{(s)}, \dots, y_n^{(s)}$ of the variable we synthesize, as well as the actual values, $y_1, \dots, y_n$ (here, think of y as `logIncome`). As before in our categorical case, let c_i describe the variables known externally by the intruder for entry i in the data.

Formally, my synthesis measure is:

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{N_{c_i}}\left(\left|\{y_j^{(s)} \in \{y_1^{(s)}, \dots, y_n^{(s)}\} : y_i < y_j^{(s)} < y_i^{(s)}\}\right|\right) I(y_i < y_i^{(s)}) + \frac{1}{N_{c_i}}\left(\left|\{y_j^{(s)} \in \{y_1^{(s)}, \dots, y_n^{(s)}\} : y_i^{(s)} < y_j^{(s)} < y_i\}\right|\right) I(y_i$$

For a single entry i, this measure describes the number of synthesized values (all of whom are identical to entry i with respect to known variables c_i) in between the true value of i and its synthesized value, before division by the number of entries that also have c_i. The overall measure is then the average of these aforementioned values.

Despite the apparent verbosity/complexity, the idea here is simple. We want to know, for each row (and on a relative scale), how many of the other synthesized values with similar eternally known variables are closer to the "true" value than the synthesis? If many are closer, then the entry should be less identifiable. Hence, measures closer to 1 reflect less disclosure risk (note that this measure is in $[0, 1]$), while measures closer to 0 reflect greater disclosure risk.

We can compute this measure for our synthetic draw above, as follows. Pretend `Race` and `UrbanRural` are known to the intruder (i.e. they constitute C – we could discretize other continuous variables, such as `Expenditure` too, if we wanted to "know" them and still use them in a join). For now, we'll focus on one variable, `logInc`, though this analysis could be extended to multiple continuous variables – doing so would amount to more "conditions" in the inner term of the above sum. We have:

```r
CEsample$logInc.syn = CEsample.s$logInc
measure = left_join(CEsample%>%mutate(id = row_number()),
                      CEsample.s%>%mutate(id = row_number()),
                      by = c("UrbanRural", "Race"), suffix = c(".t", ".s"))%>%
  group_by(id.s)%>%
  summarise(n_between = sum((logInc.t < logInc.s) & (logInc.s< logInc.syn)) +
                          sum((logInc.syn < logInc.s) & (logInc.s< logInc.t)),
            n = n())%>%
  mutate(between_rate = n_between/n)%>%
  pull(between_rate)%>%
  mean()

cat("Measure: ", measure)
```

```
## Measure:  0.2584795
```

Above, we see that our synthesis holds some disclosure risk – as touched on earlier, this may be due to the unavoidable situations where we have rare sets of c_i, wherein the synthesis being "closest" can tank the score with a zero.

Of course, this disclosure risk measure should also be compared against other utility measures – we still want to avoid the scenario where disclosure risk is low, but only because the synthetic dataset looks nothing at all like the true dataset.