Conducting a Simulation Study Using R and Mplus

An Example of Latent Class Analysis Under 24 Conditions

Chi Chang, PhD, MS
Office of Medical Education, Research and Development
Collge of Human Medicine
Michigan State University
Technical Report: CC190417

22 April, 2019

Purpose

The purpose of this report is to show how to set up a simulation study for running R and M*plus*. Latent class analysis was utilized as an example. Study factors include: 1) sample size: 200, 500; 2) number of indicators: 6, 12; 3) conditional response probabilities: 0.9, 0.8, and 0.7, and 4) latent class probability in the logit scale: 0, 0.5. In total, 24 conditions were generated in R, and then batch files were created in each folder for running M*plus* in batch. At the end of the document, I describe how the label switching problem occurred in this simualtion study, and show one of the common ways to handle the issue. The advantages and disadvantages of this way of dealing with the issue are discussed.

Methodological Studies

Methodological studies can usually be categorized into four areas:

- Methodological Developement
- Methodological Improvement
- Methodological Evaluation
- Methodological Illustration

Simulation studies can be used to:

1. Examine the performance of parameter recovery (i.e., the quality of the parameter estimation) under different scenarios:

- when models are misspecified
- when assumptions are violated
- when a specified effect (e.g., clustering effect) is ignored

2. Investigate the minimum-required sampel size

3. Examine the power of an estimate

Here we used a latent class anaylsis to demonstrate the simulation process.

```
# We load these packages first
library(Hmisc)
library(taRifx)
library(MplusAutomation)
```

Latent Class Analysis

Latent class analysis (LCA) is used to group the possible patterns of measured indicators using a probability model. It is a type of finite mixture model where the indicators are binary and class membership is a latent structure. When the indicators are continuous, it is called latent profile analysis. Both types are classification methods for cross-sectional studies.

LCA can be formulated as follows:

$$P(Y_i) = \Sigma_c^H P(C_h) \Pi_i^I P(Y_i|C_h)$$

where $i = 1...I$,I denotes the number of indicators, $c = 1...H$, H denotes the number of latent classes. If the indicators are binary, then $P(Y_i|C_h)$, the conditional response probability, can be written as $P(Y_i = 1|C_h) = \frac{exp(\beta_0)}{1+exp(\beta_0)}$, with response zero as the reference category. It is the response probability controlling for the latent class membership when there are no covariates involved. Under the local independence assumption, the probabilty of response pattern in each latent class can be obtained by multiplying across the response probability of each indicator. If there are only two latent classes specified, $P(C_h)$, latent class probability, can be formulated as $P(C_h = 1) = \frac{exp(\gamma_0)}{1+exp(\gamma_0)}$. Let $\gamma_0 = 0$. $P(C_h) = 0.5$ indicates that each subject's probability to be categorized into the first latent class is 0.5. If $\gamma_0 = .5$, then the probability of subjects to be categorized to the first class is $\frac{exp(0.5)}{1+exp(0.5)} = 0.622$. The probability of subjects to be categorized to the other latent class is $1 - 0.622 = 0.378$.

Here, latent class analysis is only used for a didactic purpose in the simulation study.

Decide the Study Factors

For the demonstration, the number of latent class is fixed at two.

```
write.cfg <- function()
{
  cfg <- expand.grid(
    n = c(200L, 500L),          # sample size
    ni = c(6L, 12L),            # number of indicators
    qi = c(0.9, 0.8, 0.7),      # crp, conditional response probability
    int = c(0, 0.5),            # latent class probability in logit scale, gamma_0 above
    stringsAsFactors = F)

 cfg <- within(cfg,
               {
                 id <- sprintf("S%02d", 1L:nrow(cfg))  # adding ID variable in the cfg data set.
               })

 write.table(cfg, 'C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/dat/cfg.txt',
             quote = F, sep = '\t', row.names = F)
 invisible(cfg)

}

write.cfg()
```

Check if cfg is what we wanted.

```
read.table('C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/dat/cfg.txt',
           header = T, as.is = T)
```

```
##      n ni  qi int  id
## 1  200  6 0.9 0.0 S01
## 2  500  6 0.9 0.0 S02
## 3  200 12 0.9 0.0 S03
## 4  500 12 0.9 0.0 S04
## 5  200  6 0.8 0.0 S05
## 6  500  6 0.8 0.0 S06
```

```
## 7  200 12 0.8 0.0 S07
## 8  500 12 0.8 0.0 S08
## 9  200  6 0.7 0.0 S09
## 10 500  6 0.7 0.0 S10
## 11 200 12 0.7 0.0 S11
## 12 500 12 0.7 0.0 S12
## 13 200  6 0.9 0.5 S13
## 14 500  6 0.9 0.5 S14
## 15 200 12 0.9 0.5 S15
## 16 500 12 0.9 0.5 S16
## 17 200  6 0.8 0.5 S17
## 18 500  6 0.8 0.5 S18
## 19 200 12 0.8 0.5 S19
## 20 500 12 0.8 0.5 S20
## 21 200  6 0.7 0.5 S21
## 22 500  6 0.7 0.5 S22
## 23 200 12 0.7 0.5 S23
## 24 500 12 0.7 0.5 S24
```

Data Generation

In M*plus*, the scheme of the data generation design is specified in the population section of Monte Carlo model. Here, we generate the data sets using R. In addition to the four study factors above, the seed, the replication number, and the location need to be specified in the function.

Thus, the elements of the *datagen* function include: sample size (nstu), replication number (rep), number of indicators (nind), γ_0 (int),conditional response probability (crp), seeds (seeds), and location(location).

```r
datagen <- function(nstu, rep = 1, nind, int, crp, seeds, location)
{
  set.seed(seeds)
  dat = NA

  # variable names: LC membership, indicators, subID.

    # Generate the latent class probability
    datemp = matrix(NA, ncol = nind + 2, nrow = nstu)
    denom1 = 1 + exp(int)
    vProb1 = cbind(exp(int)/denom1, 1/denom1)
    datemp[,1] = rbinom(n = nstu, prob = vProb1[2], size = 1) + 1

    # Generate the measurement model, conditional response probability in each latent class
    # for latent class 1, the condition probabilities are crp;
    if(sum(datemp[,1]==1) > 0)
    {datemp[,2:(1+nind)][datemp[,1] == 1] =
      matrix(rbinom(crp, n = sum(datemp[,1] == 1)*nind, size = 1), byrow = F)}

    # for latent class 2, the conditional probabilities are (1-crp);
    if(sum(datemp[,1]==2) > 0)
    {datemp[,2:(1+nind)][datemp[,1] == 2] =
      matrix(rbinom(1-crp, n = sum(datemp[,1] == 2)*nind, size = 1), byrow = F)}

    datemp[,(1+nind+1)] = 1:nstu
    dat = rbind(dat, datemp)
```

```
  file.str <- paste(location, "/sim", rep, ".txt",sep = "")
  dat <- dat[-1,] #get rid of the first NA row
  write.table(dat, file.str, row.names = F, col.names = F)
  dat
}
```

Check descriptive statistics of the data set using the *dategen* function.

```
dst <- 'C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/sim'

test = datagen(nstu = 1000, rep = 1, nind = 6, int = 0.5,
               crp = 0.8, seeds = 123, location = dst)

# size of latent class memeberships
sum(test[,1] == 2)/nrow(test)
```

```
## [1] 0.377
```

```
# conditional response probability of latent class 1
cls1 = test[test[,1] == 1,]
apply(cls1[,2:7], 2, sum)/nrow(cls1)
```

```
## [1] 0.8138042 0.7897271 0.8041734 0.7881220 0.7736758 0.8218299
```

```
# conditional response probability of latent class 2
cls2 = test[test[,1] == 2,]
apply(cls2[,2:7], 2, sum)/nrow(cls2)
```

```
## [1] 0.1830239 0.1909814 0.1803714 0.2254642 0.1830239 0.1777188
```

Write M*plus* scripts

(Note: The credit for these scripts should go to Dr. M. Lee Van Horn at the University of New Mexico and his team.)

```
MplusGen <- function(infile, rep, nind){
    mpmat <-'title: Latent Class Analysis with categorical indicators;'
    file1 <- paste(infile,'sim', rep, '.txt', sep='')
    mpmat <- rbind(mpmat, paste('data: file is ', file1, ';', sep=''))
    mpmat<- rbind(mpmat,'variable:')
    ni <- paste('u1-u',nind,sep = '')
    mpmat <- rbind(mpmat,paste('names are lc ',ni,' stuid;', sep =''))
    mpmat <- rbind(mpmat,'classes = c(2);')
    mpmat <- rbind(mpmat, paste('categorical are ', ni,' ;',sep = ''))
    mpmat <- rbind(mpmat, paste('usevariables are ',ni, ' ;', sep = ''))
    mpmat <- rbind(mpmat,'')
    mpmat <- rbind(mpmat, 'analysis: type= mixture;')
    file3 <- paste(infile,'Est',rep,'.txt;', sep='')
    mpmat <- rbind(mpmat,'Savedata: ')
    mpmat <- rbind(mpmat, paste('results are ', file3, sep = ''))
    file4 <- paste(infile, 'cp', rep, '.txt;', sep = '')
    mpmat <- rbind(mpmat, paste('file is ', file4, sep = ''))
    mpmat <- rbind(mpmat, 'save = cprobabilities;')
    file5 <- paste(infile,'inp',rep, '.inp', sep='')
    write(mpmat,file5)
```

```
}
```

Check if the M*plus* input script is correct

```
MplusGen(infile = dst, rep = 1, nind = 6)
```

Batch create folders, and put generated data sets (replication data sets) in each folder

```
dst <- 'C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/sim'
cfg <- read.table('C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/dat/cfg.txt',
                  header = T, as.is = T)

seeds.def = seq(21345, 23345, by = 2)

  for(i in 1L : nrow(cfg))
  {
    # For each condition, create a folder, using ID as the name of the folder
    cfg.row <- cfg[i,]
    sim.root <- file.path(dst, cfg.row$id)
    dir.create(sim.root, showWarnings = F, recursive = T)
    id <- cfg$id[i]

      # In each folder, generate 100 replication data sets and the Mplus script for each rep data.
      for(j in 1L:100L) {
        datagen(nstu = cfg[i, "n"], rep = j,
                nind = cfg[i, "ni"],
                int = cfg[i, "int"],
                crp = cfg[i, "qi"],
                seeds = seeds.def[j],
                location = paste(dst, '/', id, '/',sep = ''))

        MplusGen(infile = paste(dst,'/', id, '/', sep = ''), rep = j, nind = cfg[i,'ni'])
      }
  }
```

Create a `.bat` file in each folder for batch running

(Note: The credit for this script should go to Dr. M. Lee Van Horn and his team.)

In each condition folder, we begin with creating a `batchrun.bat` file, and then change the working directory to the folder and execute the file for batch running.

```
for (i in 1L: nrow(cfg))
{
  cfg.row <- cfg[i, ]
  sim.root <- file.path(dst, cfg.row$id)
  writeLines('For /L %%A in (1, 1, 100) do (CALL Mplus inp%%A.inp /b /o )',
             file.path(sim.root, 'batchrun.bat'))
  setwd(sim.root)                       # change working directory
  shell.exec('batchrun.bat')            # execute the batchrun.bat
}
```

Examine the simulated results

Did the label-switching problem occur across different replication data sets?

Label switching occurred when latent class memberships switched across replication data sets. We can see the following two outputs as an example. Extracting from codition 1 output 6 and output 7, the latent class labels do not represent the same subgroups across the replication data sets. The conditional response probabilities (i.e., the inverse logit of the threshold estimates) of latent class 1 is around 0.9 , while the crp of latent class 2 in the 6th replication result is around 0.1. However, in the results of 7th replication, the class label switched: class 1 has low crps and class 2 has high crps. The label switching phenomenon does not mean that parameter estimates are compromised. This occurred in the simulation studies since each replication data set was analyzed individually. The switch causes a problem when the researcher doesn't identify the switched cases and wrongly aggregates the parameter estimates across replication results by the estimated latent class label when evaluating the parameter recovery. Due to the fact that applied researchers deal with only one data set, this issue does not need to be addressed in applied research.

```
readModels("C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/sim/S01/inp6.out",
           what = "parameters")$parameters$unstandardized[2:7]
```

```
## Reading model:  C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/sim/S01/inp6.out
```

```
##    param    est    se est_se  pval                   LatentClass
## 1   U1$1  2.182 0.344  6.339 0.000                             1
## 2   U2$1  2.879 0.527  5.460 0.000                             1
## 3   U3$1  2.090 0.335  6.238 0.000                             1
## 4   U4$1  2.287 0.365  6.258 0.000                             1
## 5   U5$1  1.735 0.289  5.997 0.000                             1
## 6   U6$1  1.832 0.296  6.196 0.000                             1
## 7   U1$1 -2.305 0.378 -6.095 0.000                             2
## 8   U2$1 -2.709 0.426 -6.359 0.000                             2
## 9   U3$1 -2.451 0.385 -6.368 0.000                             2
## 10  U4$1 -2.564 0.413 -6.201 0.000                             2
## 11  U5$1 -2.097 0.336 -6.244 0.000                             2
## 12  U6$1 -1.677 0.294 -5.709 0.000                             2
## 13   C#1  0.002 0.145  0.011 0.991 Categorical.Latent.Variables
```

```
readModels("C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/sim/S01/inp7.out",
           what = "parameters")$parameters$unstandardized[2:7]
```

```
## Reading model:  C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/sim/S01/inp7.out
```

```
##    param    est    se est_se  pval                   LatentClass
## 1   U1$1 -2.525 0.393 -6.425 0.000                             1
## 2   U2$1 -2.333 0.371 -6.284 0.000                             1
## 3   U3$1 -2.381 0.366 -6.509 0.000                             1
## 4   U4$1 -2.192 0.334 -6.560 0.000                             1
## 5   U5$1 -2.686 0.427 -6.285 0.000                             1
## 6   U6$1 -2.042 0.317 -6.435 0.000                             1
## 7   U1$1  2.133 0.330  6.470 0.000                             2
## 8   U2$1  2.311 0.350  6.604 0.000                             2
## 9   U3$1  2.358 0.368  6.416 0.000                             2
## 10  U4$1  1.972 0.317  6.220 0.000                             2
## 11  U5$1  2.368 0.364  6.508 0.000                             2
## 12  U6$1  2.640 0.417  6.327 0.000                             2
## 13   C#1 -0.004 0.143 -0.031 0.976 Categorical.Latent.Variables
```

Relabel latent classes manually

Researchers proposed several ways to solve label switching problems in simulation studies which used Bayesian methods and MCMC algorithm, but few investigate solutions for studies which used ML algorithm. Constraining the parameter estimation may solve the problem. However, this results in biased estimates when the parameters are near the border of the constrained parameter space.

Instead of constraining, we manually switch the estimates. Let latent class one be the class having high crps, and latent class two be the one having low crps. In the following, the estimates of crp are sorted in each output. If the mean of the crp estimates in the classes has a flipped order, their estimates will be moved to the corresponding position in class 1 so that 100 replication results will have consistent labels.

Note that the estimate of latent probability, γ_0, needs to be flipped as well, if the labels are switched. Because the inverse logit of γ_0 in latent class one equals to 1 minus the inverse logit of $-\gamma_0$, to switch it to the estimate in the other class, the latent class probability in logit scale is multiplied by -1 in class-flipped case.

```
MplusReadRaw <- function(cond, nind = 6, dst){
 int = NULL
 est = matrix(rep(0, 2*(2*nind+1)*100), nrow = 100, byrow = T)
 for (i in 1L:100L){
    out = readModels(paste(dst, '/', cond,'/inp', i, '.out' ,sep = ''), what = "all", quiet = T)
    if(mean(out$parameters$unstandardized$est[1:nind]) >
       mean(out$parameters$unstandardized$est[(nind+1):(2*nind)]))
      {
      est[i, ] = c(out$parameter$unstandardized$est[c(1:nind, (nind+1):(2*nind), 2*nind+1)],
                   out$parameters$unstandardized$se[c(1:nind, (nind+1):(2*nind), 2*nind+1)])

      } else {
                  # change the order of crp estimates
      est[i, ] = c(out$parameter$unstandardized$est[c((nind+1):(2*nind), 1:nind, (2*nind+1))],
                   # change the order of standard errors of crp estimates
                   out$parameters$unstandardized$se[c((nind+1):(2*nind), 1:nind, (2*nind+1))])
      est[i, 2*nind+1] <- (-1) * est[i, 2*nind+1]  # change the latent class probability
      }
 }
  est <- as.data.frame(est)
  nm = c(paste('L1ind', 1:nind, sep = ""), paste('L2ind', 1:nind, sep = ""), 'int')
  names(est) <- c(nm, paste(nm, '_se', sep = ""))

  write.csv(est, paste(dst, '/', cond,"/RawEst.csv",sep = ""), row.names = F,quote = F)
}
```

Looping through all the conditions, so that in each condition folder, `RawEst.csv` file will be created with all 100 replication results extracted from the outputs.

```
cfg <- read.table('C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/dat/cfg.txt',
                  header = T, as.is = T)

for(i in 1L:nrow(cfg)){
  MplusReadRaw(cond = cfg[i, 'id'], nind = cfg[i, 'ni'],
               dst = 'C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/sim')
}
```

Take a look at the estimates across 100 replications in the first condition. The first six columns are the six crp estimates in latent class 1, the following six columns are the six crp estimates in latent class 2, and the 13th column is the latent class probability in the logit scale. The following 13 columns are the standard error of the corresponding parameter estimates.

```r
# the first condition out of 24
cfg <- read.table('C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/dat/cfg.txt',
                  header = T, as.is = T)

cfg[1, ]
```

```
##     n ni  qi int  id
## 1 200  6 0.9   0 S01
```

```r
# The estimates from 100 replication results.
head(read.csv('C:/Users/chisq/Dropbox/SimulationGroup/SimGroup/SimGrpDemo/sim/S01/RawEst.csv', header = T))
```

```
##   L1ind1 L1ind2 L1ind3 L1ind4 L1ind5 L1ind6 L2ind1 L2ind2 L2ind3 L2ind4
## 1  2.717  2.248  2.310  2.052  1.994  3.007 -1.830 -1.599 -1.704 -2.102
## 2  2.220  2.184  2.346  1.879  2.372  1.999 -2.650 -2.444 -1.924 -2.722
## 3  1.919  1.737  1.677  2.247  2.310  2.058 -2.441 -2.316 -1.642 -2.580
## 4  2.124  2.066  3.011  1.810  2.240  2.060 -1.994 -1.847 -2.651 -1.946
## 5  1.922  1.958  2.331  2.270  2.221  2.272 -2.484 -2.157 -1.828 -2.167
## 6  2.182  2.879  2.090  2.287  1.735  1.832 -2.305 -2.709 -2.451 -2.564
##   L2ind5 L2ind6    int L1ind1_se L1ind2_se L1ind3_se L1ind4_se L1ind5_se
## 1 -2.146 -2.035 -0.116     0.454     0.366     0.367     0.333     0.321
## 2 -1.853 -1.991  0.077     0.347     0.332     0.350     0.294     0.373
## 3 -1.973 -2.251 -0.132     0.336     0.299     0.288     0.390     0.388
## 4 -2.634 -2.376  0.070     0.321     0.316     0.504     0.296     0.351
## 5 -2.492 -2.285  0.052     0.299     0.306     0.350     0.347     0.334
## 6 -2.097 -1.677  0.002     0.344     0.527     0.335     0.365     0.289
##   L1ind6_se L2ind1_se L2ind2_se L2ind3_se L2ind4_se L2ind5_se L2ind6_se
## 1     0.509     0.287     0.263     0.277     0.318     0.334     0.316
## 2     0.313     0.420     0.396     0.319     0.456     0.300     0.320
## 3     0.335     0.366     0.361     0.277     0.392     0.305     0.366
## 4     0.315     0.331     0.306     0.427     0.310     0.419     0.383
## 5     0.349     0.389     0.334     0.298     0.337     0.395     0.354
## 6     0.296     0.378     0.426     0.385     0.413     0.336     0.294
##   int_se
## 1  0.144
## 2  0.144
## 3  0.146
## 4  0.144
## 5  0.142
## 6  0.145
```

Note that manually switching labels has its limitations as well. First, in a scenario where the parameters among classes are very close, sorting may not provide an optimal solution. It may end up with biased or unbiased estimates with unsolved label switching cases in it. Second, when there are k latent classes specified in the simluation, the results can end up with 2^k possibile ways that labels can be assigned. When $k = 3$, the following property of logarithm can be used to reparameterize the estimates to a different reference group.

$$log\frac{P(C_h = A)}{P(C_h = B)} = log\frac{P(C_h = A)}{P(C_h = C)} - log\frac{P(C_h = B)}{P(C_h = C)}$$

However, since standard errors of the estimates do not have the same properties, reparameterization for standard errors becomes very difficult. Third, when the data has a nested structure and classification for the higher-level units (e.g., school classification) are the purpose, modeling is possible (e.g., using nonparameteric multilevel latent class modeling) but label switching at different levels can occur at the same time. The complexity of hierarchical models could make the process of relabling more complicated.

The `MplusAutomation` package can be used to extract estimates from the replication data sets to examine the performance of parameter recovery. Since there is no model misspecification in this simulation study, it is not the focus in this document. Readers who are interested in the phase of parameter recovery are referred to `MplusAutomation` documentation.