Using add-on packages to fit latent class models with the **xgrid** package

Sarah C. Anoke, Nicholas J. Horton*, Yuting Zhao
Department of Mathematics and Statistics
Smith College

April 6, 2012

Contents

1 Introduction

Many scientific computations can be sped up by dividing them into smaller tasks and distributing the computations to multiple systems for simultaneous processing. Such a process is referred to as *parallel computing*. When performed on existing grids of computers, this method can dramatically increase computation speed. Several solutions exist to facilitate this type of computation within R, and we describe one such solution here, that involves using the Apple Xgrid (Apple, 2009), a parallel computing environment.

We created the **xgrid** package to provide a simple interface to this distributed computing system (Horton et al., 2011). The package facilitates use of an Apple Xgrid for distributed processing of a job with many independent repetitions, by simplifying task submission (or *gridstuffing*) and collation of results.

*Corresponding author: nhorton@smith.edu

We demonstrate use of our package in the context of a real statistical problem. This example is representative of what might be encountered in the field, as it involves simulations that would ordinarily take days to complete. It involves study of the properties of latent class models, which are used to determine better schemes for classification of eating disorders (Keel et al., 2004). The development of an empirically-created eating disorder classification system is of public health interest as it may help identify individuals who would benefit from diagnosis and treatment.

2 Latent Class Analysis

As described by Collins and Lanza (2009), LCA is used to identify subgroups in a population. There are several criteria used to evaluate the fit of a given model, including the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), the Consistent Akaike Information Criterion (cAIC), and the Sample Size Adjusted Bayesian Information Criterion (aBIC). These criteria are useful, but further guidance is needed for researchers to choose between them, as well as better understand how their accuracy is affected by methodological factors encountered in eating disorder classification research, such as unbalanced class size, sample size, missing data, and under- or overspecification of the model. Swanson et al. (2011) undertook a comprehensive review of these model criteria, including a full simulation study to generate hypothetical datasets and investigate how each criterion behaved in a variety of statistical environments. In this example, we replicate some these results using an Apple Xgrid to grid to speed up computation time.

3 Using the Grid for a Simulation Study

Following the approach of Swanson et al. (2011), we generated "true models" where there was an arbitrary 4-class structure, with balanced number of observations in each class. This structure was composed of 10 binary indicators in a simulated dataset of size 300. The model was fit using the `poLCA()` function in the **poLCA** (polytomous latent class analysis) package (Linzer and Lewis, 2011). Separate latent class models were fit specifying the number of classes, ranging from 2 to 6. For each simulation, we determined the lowest values of BIC, AIC, cAIC and aBIC and recorded the class structure associated with each value.

Swanson and colleagues found that for this set of parameter values, the AIC and aBIC picked the correct number of classes more than half the time (see Table 1).

This example illustrates the computational burden of undertaking simulation studies to assess the performance of modern statistical methods, as several minutes are needed to undertake each of the single iterations of the simulation (which may explain why Swanson and colleagues only fit 100 simulations for each of their scenarios).

3.1 Directory Structure

Our first step is to set up a directory structure for our simulation (see Figure 1).

The first item is the directory '`input`', which contains two files that will be run on the remote agents. The first of these files, '`job.R`' (Figures 2 and 3), defines the code to run a particular job. In this case, '`job.R`' defines two functions. The function `datagen()` generates a 4-class data set. The

		Class				
Criteria	Acronym	2	3	**4**	5	6+
Bayesian Information Criterion	BIC	49%	44%	**7%**	0%	0%
Akaike Information Criterion	AIC	0%	0%	**53%**	31%	16%
Consistent Akaike Information Criterion	cAIC	73%	25%	**2%**	0%	0%
Sample Size Adjusted Bayesian Information Criterion	aBIC	0%	5%	**87%**	6%	2%

Table 1: Percentage of times (out of 100 simulations) that a particular number of classes was selected as the best model (where the true data derive from 4 distinct classes), when `n=300`, reprinted from Swanson et al. (2011). Note that a perfect criterion would have 100% under class 4.

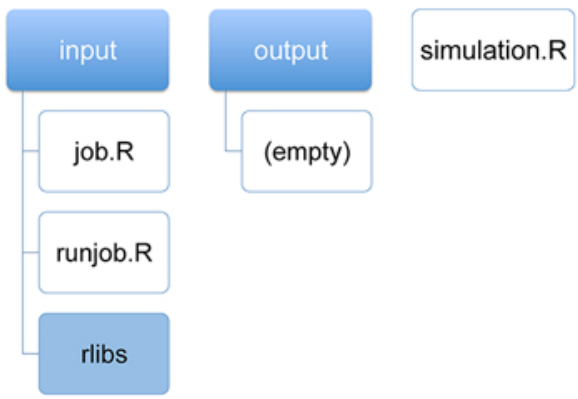

Figure 1: File structure to access the grid

sample size of this data set is specified by the argument `samplesize` and the number of indicators is specified by `dim`. The function `job()` takes six arguments: `param` specifies the sample size of the data set; `smallclass` and `largeclass` define the range of class structures; `dim` specifies the number of indicators; `nrep` defines the number of times the poLCA model is estimated for each data set; and `ntask` specifies how many tasks to run within each job. The function `job()` calls `datagen()` to generate a data set, and fits a model to these data using `poLCA()`. It returns a data frame containing the class structure (within the range of `smallclass` and `largeclass`) predicted by each information criterion (BIC, AIC, cAIC and aBIC).

The folder '`input`' also contains '`runjob.R`' (Figure 4), which retrieves and stores command line arguments from the controller, and passes them to `job()`. The results from the completed job are saved as `res0`, which is subsequently saved to the '`output`' folder.

The folder '`output`' will contain results from the simulations. After the completion of each job, the results are saved to a file in this directory. If this directory is created manually, it should be empty. If not created manually, `xgrid()` will create it.

The next item in the directory structure is '`simulation.R`' (Figure 5), which contains the R script run on the client machine that calls `xgrid()`. This function submits the simulation to the grid for calculation. Because this script is just an example of how to call `xgrid()` and manipulate the resulting object, it can be run the way one typically submits commands during an R session. Results

```
> # code adapted from Swanson et al. (2011)
> library(scatterplot3d, lib.loc="./rlibs")
> library(poLCA, lib.loc="./rlibs")
> # generates data set of size 'samplesize'
> # and 'dim' indicators
> # with a predefined class structure
> datagen <- function(samplesize = 300, dim = 10) {
   if(samplesize %% 4 == 0) {
     subset <- samplesize / 4
   } else {
     cat("Error: samplesize must be divisible by the number of classes 4.")
   }

   # predetermining the 4-class structure
   # values from Swanson et al. (2011)
   class1 <- c(1.45, -3.48,  0.04,  1.05, -0.49,
                    -1.10, -2.44, -1.99, -2.75, -3.48)
   class2 <- c(-0.1,  1.38,  1.00,  1.06,  1.28,
               0.48,  0.66,  0.43,  0.87,  0.16)
   class3 <- c(1.28, -0.07, -0.45, -1.06,  0.19,
               1.19, -0.83,  0.91, -1.12,  2.00)
   class4 <- c(   0, -2.00, -1.69,  0.31,     0,
               -0.58, 1.06, -1.51, -2.00,  0.63)
   type1 <- exp(class1) / (1 + exp(class1))
   type2 <- exp(class2) / (1 + exp(class2))
   type3 <- exp(class3) / (1 + exp(class3))
   type4 <- exp(class4) / (1 + exp(class4))

   x <- matrix(runif(samplesize * dim), samplesize)
   x[(0 * subset + 1) : (1 * subset), ] <-
     t(t(x[(0 * subset + 1) : (1 * subset), ]) < type1) * 1
   x[(1 * subset + 1) : (2 * subset), ] <-
     t(t(x[(1 * subset + 1) : (2 * subset), ]) < type2) * 1
   x[(2 * subset + 1) : (3 * subset), ] <-
     t(t(x[(2 * subset + 1) : (3 * subset), ]) < type3) * 1
   x[(3 * subset + 1) : (4 * subset), ] <-
     t(t(x[(3 * subset + 1) : (4 * subset), ]) < type4) * 1
   ds <- as.data.frame(x + 1)
   return(ds)
 }
```

Figure 2: Contents of 'job.R', part 1 – the function `datagen()`. This code, as well as that of Figure 3, should be in one file entitled 'job.R'.

from all jobs are returned as one object, `res`.

3.2 Retrieval and Analysis of Results

Figures 6 and 7 together demonstrate an example of what to expect before and after completing all simulations.

At the beginning of Figure 6, we see the expected directory structure. We then call `xgrid()` to send `numsim=20` simulations to the grid for calculation. After the completion of the entire simulation study, results from all `numsim` simulations are collated and returned by the `xgrid()` function as the object `res`. This object is also saved as the file 'RESULTS.rda' at the top of the directory structure (as seen in Figure 7).

Figure 7 also displays an example of what would be seen in the 'output' folder. As expected, there are twenty files of the form 'RESULT-1000#', which contain the results from each individual job. The second set of twenty files (e.g. 'runjob.RRESULT-1000#.Rout') contain the code that was run to generate the corresponding result file.

In this example, `res` is a data frame with `numsim` rows (one row for each simulation) and five columns (as defined in the `return` statement of 'job.R'). In Figure 6, we list the dimensions of `res` and summarize the results using `apply()`.

The output generated by `apply()` is also as expected. For `res$samplesize`, we see that the value `200` appears `20` times, once for each simulation. The information criterion `bic` chose a 2-class structure 13 times and a 3 class structure 7 times – reasonably close to what we'd expect based on the results of Swanson et al. (2011). (Table 1).

4 Installation of Add-On Packages

A complication of this example is that fitting latent class models in R requires the use of add-on packages. If the user of the grid has administrative privileges on the individual machines, any needed packages can be installed in the usual manner. However, if no administrative access is available, it is possible to utilize such packages within this setup by manually installing them in the 'input/rlibs' directory and loading them within a given job. In this example, we use the **poLCA** (polytomous latent class analysis) package and its supporting packages (e.g. **scatterplot3d**, **MASS**) with the Apple Xgrid to conduct our simulation. Below we enumerate how to make an individual package available to a job running on a given agent.

Install the appropriate distribution file from CRAN into the directory 'input/rlibs'. This can be done by using the `lib` argument of `install.packages()`. For instance, in Example 2 we can install **poLCA** by using the call `install.packages("poLCA", lib = "~/.../input/rlibs")`, where '~/.../' emphasizes use of the absolute path to the 'rlibs' directory.

Access the add-on package within a job running on an agent. This is done by using the `lib.loc` option within the standard invocation of `library()`. For instance, in Example 2 this can be done by using the call `library(poLCA, lib.loc="./rlibs")`.

It should be noted that the package will need to shipped over to the agent for each job, which may be less efficient than installing the package once per agent in the usual manner (but the latter option may not be available unless the grid user has administrative access to the individual agents).

5 Acknowledgements

This material is based in part upon work supported by the National Institute of Mental Health (5R01MH087786-02) and the US National Science Foundation (DUE-0920350, DMS-0721661, and DMS-0602110).

6 Bibliography

Apple. *Mac OS X Server: Xgrid Administration and High Performance Computing (Version 10.6 Snow Leopard)*. Apple Inc, 2009.

L. M. Collins and S. T. Lanza. *Latent Class and Latent Transition Analysis: With Applications in the Social, Behavioral, and Health Sciences*. Wiley, 2009.

N. J. Horton, S. C. Anoke, Y. Zhao, and H. Jaeger. Xgrid and R: Parallel distributed processing using heterogeneous groups of Apple computers. *In revision*, 2011.

D. A. Linzer and J. B. Lewis. poLCA: An R package for polytomous variable latent class analysis. *Journal of Statistical Software*, 42(10):1–29, 2011. URL http://www.jstatsoft.org/v42/i10/.

P. K. Keel, M. Fichter, N. Quadflieg, C. M. Bulik, M. G. Baxter, L. Thornton, K. A. Halmi, A. S. Kaplan, M. Strober, D. B. Woodside, S. J. Crow, J. E. Mitchell, A. Rotondo, M. Mauri, G. Cassano, J. Treasure, D. Goldman, W. H. Berrettini, and W. H. Kaye. Application of a latent class analysis to empirically define eating disorder phenotypes. *Archives of General Psychiatry*, 61(2):192–200, February 2004.

S. A. Swanson, K. Lindenberg, S. Bauer, and R. D. Crosby. A Monte Carlo investigation of factors influencing latent class analysis: An application to eating disorder research. *International Journal of Eating Disorders*, 2011.

```
> job <- function(ntask, param, dim = 10,
                  smallclass = 2, largeclass = 6, nrep = 100) {
   diff <- largeclass - smallclass + 1
   # LCA formula
   f <- cbind(V1, V2, V3, V4, V5, V6, V7, V8, V9, V10) ~ 1
   allbic  <- rep(0, ntask)
   allaic  <- rep(0, ntask)
   allcaic <- rep(0, ntask)
   allabic <- rep(0, ntask)
   bicmat  <- matrix(nrow = ntask, ncol = diff)
   aicmat  <- matrix(nrow = ntask, ncol = diff)
   caicmat <- matrix(nrow = ntask, ncol = diff)
   abicmat <- matrix(nrow = ntask, ncol = diff)

   for (j in 1:ntask) {
     ds <- datagen(samplesize = as.numeric(param), dim = dim)
     bic  <- rep(0, diff)
     aic  <- rep(0, diff)
     caic <- rep(0, diff)
     abic <- rep(0, diff)

     for (i in smallclass:largeclass) {
       lc <- poLCA(f, data = ds, nclass = i, graphs = FALSE,
                   verbose = FALSE, nrep = nrep)
       bic[i-1]  <- lc$bic
       aic[i-1]  <- -2 * lc$llik + 2 * lc$npar
       caic[i-1] <- -2 * lc$llik + lc$npar * log(lc$N) + 1
       abic[i-1] <- -2 * lc$llik + lc$npar * log((lc$N + 2) / 24)
     }
     bicmat[j, ]  <- bic
     minbic       <- which.min(bic) + 1
     allbic[j]    <- minbic
     aicmat[j, ]  <- aic
     minaic       <- which.min(aic) + 1
     allaic[j]    <- minaic
     caicmat[j, ] <- caic
     mincaic      <- which.min(caic) + 1
     allcaic[j]   <- mincaic
     abicmat[j, ] <- abic
     minabic      <- which.min(abic) + 1
     allabic[j]   <- minabic
   }
   rowname <- rep(param, ntask)
   res <- data.frame(samplesize = rowname, bic = allbic,
                     aic = allaic, caic = allcaic, abic = allabic)
   return(res)
 }
```

Figure 3: Contents of 'job.R', part 2 – the function `job()`. This code, as well as that of Figure 2, should be in one file entitled 'job.R'.

```
> source("job.R")
> # commandArgs() is expecting three arguments:
> # 1) number of tasks to run within this job
> # 2) parameter to pass to the function
> # 3) place to stash the results when finished
> args     <- commandArgs(trailingOnly = TRUE)
> ntask1   <- as.numeric(args[1])
> param1   <- args[2]
> resfile  <- args[3]
> res0     <- job(ntask = ntask1, param = param1)
> # stash the results
> saveRDS(res0, file = resfile)
```

Figure 4: Contents of 'runjob.R'

```
> library(xgrid)
> # run the simulation
> res <- xgrid(grid = "Burton-303-iMac", Rcmd = "runjob.R",
              param = 300, numsim = 20, ntask = 1)
```

Figure 5: Contents of 'simulation.R'

```
> list.files()
[1] "directorystructure.pdf" "example2-source.Rnw"    "example2-source.pdf"
[4] "example2-source.tex"    "example2.Rnw"           "input"
[7] "output"                 "simulation.R"
> list.files("input")
[1] "job.R"    "rlibs"    "runjob.R"
> library(xgrid)
> res <- xgrid(grid = "Burton-303-iMac", Rcmd = "runjob.R",
              param = 300, numsim = 20, ntask = 1)
> dim(res)
[1] 20  5
> head(res, 3)
  samplesize bic aic caic abic
1        300   3   4    3    4
2        300   2   4    2    4
3        300   2   5    2    4
> apply(res, 2, table)
$samplesize

300
 20

$bic

 2  3  4
10  9  1

$aic

4 5 6
8 3 9

$caic

 2  3  4
10  9  1

$abic

 3  4  5
 1 16  3
```

Figure 6: Expected console output, part 1 – before and after calling `xgrid()` (see Figure 7 for part 2)

```
> list.files()
 [1] "RESULTS.rds"            "directorystructure.pdf" "example2-source.Rnw"
 [4] "example2-source.pdf"    "example2-source.tex"    "example2.Rnw"
 [7] "input"                  "job.err"                "job.out"
[10] "output"                 "simulation.R"
> list.files("output")
 [1] "RESULT-10000"              "RESULT-10001"              "RESULT-10002"
 [4] "RESULT-10003"              "RESULT-10004"              "RESULT-10005"
 [7] "RESULT-10006"              "RESULT-10007"              "RESULT-10008"
[10] "RESULT-10009"              "RESULT-10010"              "RESULT-10011"
[13] "RESULT-10012"              "RESULT-10013"              "RESULT-10014"
[16] "RESULT-10015"              "RESULT-10016"              "RESULT-10017"
[19] "RESULT-10018"              "RESULT-10019"              "runjob.RRESULT-10000.Rout"
[22] "runjob.RRESULT-10001.Rout" "runjob.RRESULT-10002.Rout" "runjob.RRESULT-10003.Rout"
[25] "runjob.RRESULT-10004.Rout" "runjob.RRESULT-10005.Rout" "runjob.RRESULT-10006.Rout"
[28] "runjob.RRESULT-10007.Rout" "runjob.RRESULT-10008.Rout" "runjob.RRESULT-10009.Rout"
[31] "runjob.RRESULT-10010.Rout" "runjob.RRESULT-10011.Rout" "runjob.RRESULT-10012.Rout"
[34] "runjob.RRESULT-10013.Rout" "runjob.RRESULT-10014.Rout" "runjob.RRESULT-10015.Rout"
[37] "runjob.RRESULT-10016.Rout" "runjob.RRESULT-10017.Rout" "runjob.RRESULT-10018.Rout"
[40] "runjob.RRESULT-10019.Rout"
```

Figure 7: Expected console output, part 2 – results after calling `xgrid()` (see Figure 6 for part 1). Note that files 'job.err' and 'job.out' are output files unnecessary for the interpretation of results, but listed here for completeness.