Paper CT-09

Fitting Bayesian hierarchical multinomial logit models in PROC MCMC

Jacob C. Fisher, Duke University, Durham, NC

ABSTRACT

The paper illustrates how to use the MCMC procedure to fit a hierarchical, multinomial logit model for a nominal response variable with correlated responses in a Bayesian framework. In particular, the paper illustrates how to perform three important parts of Bayesian model fitting. First, to make sure appropriate prior distributions are selected, the paper shows how to simulate draws directly from the prior distribution. Second, since the reference category and random effects may require special attention, the paper shows how to code the sampling model into PROC MCMC using the RANDOM statement, new to SAS® 9.3. Finally, the paper demonstrates how to run two chains simultaneously on a multi-core processor, and how to use those two chains to check convergence of the MCMC chain using the Gelman-Rubin diagnostic test. By following these steps, many common pitfalls associated with fitting complicated models in PROC MCMC may be avoided. The target audience for this paper is people with some knowledge of Bayesian methods and a moderate level of SAS experience, but who may not be familiar with PROC MCMC or multinomial logit models.

INTRODUCTION

Nominal response variables with correlated observations are common in social science research. One approach to modeling nominal response variables is the multicategory (also called multinomial) logit model (Agresti, 2007). With a multinomial logit model, the log odds of a response falling into a particular category, *j*, relative to a baseline category, *J*, are modeled by separate equations for each of the *J – 1* comparisons. This generalized linear model can be extended to include random effects to account for the correlated observations which may arise in clustered or panel data. The MCMC procedure, and particularly the RANDOM statement new to SAS 9.3, greatly simplifies fitting such hierarchical, multinomial logit models within a Bayesian framework.

This paper will illustrate how to construct and assess the convergence of a Bayesian multinomial logit model with random effects for households using panel data on yogurt purchases. The yogurt data, analyzed previously by Jain, Vilcassim, and Chintagunta (1994) and Chen and Kuo (2001), describes purchases of yogurt over approximately two years by a panel of 100 households in Springfield, MO. The data are freely available from the mlogit package in R; syntax to import the data from R into SAS appears in the appendix.

DESCRIPTION OF EXAMPLE & MODEL

YOGURT DATA

The data set used for this example contains information from optical scanner records about yogurt purchases by a panel of 100 households, as well as information about the price of each type of yogurt and information about the marketing of each type of yogurt (namely whether or not the yogurt was featured in a newspaper advertisement). Table 1 displays the first few lines of the data set, to illustrate the data structure.

id	yoplait	dannon	weight	hiland	price_ yoplait	price_ dannon	price_ hiland	price_ weight	feat_ yoplait	feat_ dannon	feat_ hiland	feat_ weight
1	0	0	1	0	0.11	0.08	0.06	0.08	0	0	0	0
1	0	1	0	0	0.11	0.10	0.06	0.08	0	0	0	0
1	0	1	0	0	0.11	0.10	0.06	0.09	0	0	0	0
1	0	1	0	0	0.11	0.10	0.06	0.09	0	0	0	0
1	0	1	0	0	0.13	0.10	0.05	0.08	0	0	0	0
1	0	1	0	0	0.11	0.09	0.05	0.08	0	0	0	0
1	0	1	0	0	0.10	0.08	0.05	0.08	0	0	0	0
1	0	0	1	0	0.11	0.09	0.05	0.08	0	0	0	0
2	1	0	0	0	0.11	0.10	0.05	0.08	0	0	0	0
2	1	0	0	0	0.11	0.10	0.05	0.08	0	0	0	0

Table 1: First ten lines of the yogurt data

The variable id lists the household ID. The variables yoplait, dannon, weight, and hiland are indicator variables, denotes which brand of yogurt, Yoplait, Dannon, Weight Watchers, or Hiland, respectively, the household purchased on that occasion. The price in dollars of each brand of yogurt at the time of purchase is given by the four variables

beginning with "price_," and whether each brand of yogurt was featured in a newspaper advertisement at the time of purchase is given by the indicator variables whose names begin with "feat_".

MODEL

From these data, the model described in Chen and Kuo (2001) can be constructed. The model predicts brand choice as a function of whether the yogurt was featured in an advertisement, the price of the yogurt, and a random intercept for each household. Let j index the 4 brands of yogurt, and let $BRAND_{it}$ denote the brand chosen by the ith household at the tth purchase. Then

$$BRAND_{it} \sim Multinomial(p_{it})$$

$$\text{where } p_{itj} = \Pr(BRAND_{it} = j) = \frac{\exp(\alpha_{ij} + \beta_1 FEATURE_{itj} + \beta_2 PRICE_{itj})}{\sum_{j=1}^{4} \exp(\alpha_{ij} + \beta_1 FEATURE_{itj} + \beta_2 PRICE_{itj})}, j = 1, \ldots, 4$$

The random intercepts α_j are distributed according to a multivariate normal distribution.

$$\alpha_j \sim N_3(\boldsymbol{\mu}, \Sigma)$$

where $\alpha_i = (\alpha_{i1}, \alpha_{i2}, \alpha_{i3})'$, which represents the household-specific preference effects for Yoplait, Dannon, and Weight Watchers yogurt, respectively, and $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3)'$, which represents the population averages for each of the three brands. Hiland yogurt is the reference category and therefore $\alpha_4 = 0$. Diffuse priors reflect our lack of specific prior beliefs about the values of the parameters.

$$\begin{bmatrix}\mu_1\\ \mu_2\\ \mu_3\end{bmatrix} \sim N_3\left(\begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}625 & 200 & 200\\ 200 & 625 & 200\\ 200 & 200 & 625\end{bmatrix}\right)$$

$$\Sigma \sim \text{inverse-Wishart}_3\left(5, \begin{bmatrix}500 & 50 & 50\\ 50 & 500 & 50\\ 50 & 50 & 500\end{bmatrix}\right)$$

$$\begin{bmatrix}\beta_1\\ \beta_2\end{bmatrix} \sim N_2\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}10000 & 0\\ 0 & 10000\end{bmatrix}\right)$$

Once written, the model can be coded into PROC MCMC.

SETTING UP PROC MCMC

The easiest way to construct a model in PROC MCMC is to construct it in pieces. First, the prior can be coded and tested, then the sampling model with random effects can be incorporated.

PRIOR DISTRIBUTIONS

When constructing a Bayesian model, it is always advisable to begin by double-checking that the priors chosen accurately represent one's prior beliefs. The easiest way to accomplish this in PROC MCMC is to program the priors, and then have PROC MCMC simulate draws from the prior distribution. By setting the likelihood function to a constant, PROC MCMC will draw directly from the prior distributions. The following code illustrates generating samples from the prior distributions for the model above.

```
/* Prior predictive checks */
PROC MCMC
 DATA = yogurt /* must specify a valid input data set, even though no data will be
used from it */
 NMC = 5000 /* Size of the sample */
 NBI = 0; /* No need to generate burn-in samples, because the draws are taken
directly from the prior */
 ODS SELECT PostSummaries;

 /* Parameters for the prior distribution */
 ARRAY mu[3];
 ARRAY sigma[3,3];
 PARMS beta1 beta2;
 PARMS mu;
 PARMS sigma;
```

```
 /* Hyperprior constants */
 ARRAY mu0[3] (0 0 0);
 ARRAY sigma0[3,3] (625, 200, 200, 200, 625, 200, 200, 200, 625);
 ARRAY S[3,3] (500, 50, 50, 50, 500, 50, 50, 50, 500);

 /* Prior distributions */
 PRIOR beta: ~ NORM(0, var = 10000); /* Note that since the prior distributions for
beta1 and beta2 are independent, they can be programmed without a multivariate
distribution. */
 PRIOR sigma ~ IWISH(5, S);
 PRIOR mu ~ MVN(mu0, sigma0);

 /* Constant likelihood function */
 MODEL GENERAL(0);
RUN;
```

Running the code above generates the following output

Posterior Summaries						
Parameter	**N**	**Mean**	**Standard Deviation**	**Percentiles**		
				25%	**50%**	**75%**
beta1	5000	3.0855	102.3	-73.7626	9.2271	73.0170
beta2	5000	5.7560	93.1207	-53.3014	4.6455	66.8533
mu1	5000	0.1098	24.1076	-17.0237	-1.4396	16.4542
mu2	5000	1.0608	24.7808	-14.5453	-0.3585	19.1437
mu3	5000	-1.0792	24.9704	-17.3001	-0.3479	16.2052
sigma1	5000	521.1	3352.1	124.1	217.1	429.1
sigma2	5000	47.5459	1591.2	-52.4282	18.8079	105.4
sigma3	5000	56.6791	1284.7	-54.0974	18.6658	112.2
sigma4	5000	47.5459	1591.2	-52.4282	18.8079	105.4
sigma5	5000	499.8	1935.3	121.3	211.6	419.8
sigma6	5000	32.3706	1369.2	-56.1831	17.0032	104.2
sigma7	5000	56.6791	1284.7	-54.0974	18.6658	112.2
sigma8	5000	32.3706	1369.2	-56.1831	17.0032	104.2
sigma9	5000	482.1	1411.2	124.6	219.1	429.1

Table 2. Simulated Draws from Prior Distributions

The results show that β_1 and β_2 are centered at 0 with wide variances. μ_1 - μ_3 are also centered at 0 with standard deviations of 25. The variances for the random effects (represented by sigma1, sigma 5, and sigma9) are also wide with relatively small covariances (represented by the other values of the sigma matrix), indicating our belief that the means of the random effects are largely uncorrelated. The values of the hyperprior parameters can be tuned and this process can be repeated until the prior distributions accurately represent one's prior beliefs. Once the prior distributions have been tuned appropriately, the rest of the model may be programmed in to PROC MCMC.

DATA AND SAMPLING MODEL

Following the tuning of the prior distributions, the next step in fitting the multinomial model is to program the likelihood function. When fitting this model with PROC NLMIXED, Chen and Kuo (2001) offer an alternative specification of the model, a Poisson non-linear model. To fit the Poisson non-linear model, the authors transpose the data set so that, within every purchase occasion, *t*, every brand has its own row. Transforming the data and the model is unnecessary to fit the model in PROC MCMC. As of SAS 9.3 (Stokes, 2011), PROC MCMC supports multivariate distributions such as the multinomial distribution. Therefore the model can be programmed directly as it appears above.

The following code illustrates the PROC MCMC statement within a macro, which will allow running multiple chains more easily. Multiple chains are necessary for Gelman-Rubin convergence diagnostics, which will be discussed in a later section.

```
%MACRO mcmc_model (iter, burnin, thin, indata, outdata);
PROC MCMC
 DATA = &indata
 NMC = &iter NBI = &burnin THIN = &thin
 INIT = RANDOM /* Random initial values allow Gelman-Rubin diagnostics */
 OUTPOST = &outdata; /* Save draws from the posterior distribution */

 ARRAY mu[3];
 ARRAY sigma[3,3];
 PARMS beta1 beta2 mu sigma; /* Since this model uses conjugate sampling, the
parameters can all be included in the same PARMS statement without being proposed
together. */

 /* Hyperprior constants */
 ARRAY mu0[3] (0 0 0);
 ARRAY sigma0[3,3] (625, 200, 200, 200, 625, 200, 200, 200, 625);
 ARRAY S[3,3] (500, 50, 50, 50, 500, 50, 50, 50, 500);

 /* Prior distributions */
 PRIOR beta: ~ NORM(0, VAR = 10000);
 PRIOR sigma ~ IWISH(5, S);
 PRIOR mu ~ MVN(mu0, sigma0);

 /* Data */
 ARRAY brand[4] yoplait dannon weight hiland;
 ARRAY feature[4] feat_yoplait feat_dannon feat_weight feat_hiland;
 ARRAY price[4] price_yoplait price_dannon price_weight price_hiland;

 ARRAY p[4];       /* Vector of probabilities */
 ARRAY exp_eta[4]; /* Vector of linear predictor */
 ARRAY alpha[3];   /* Random effects */

 /* Sampling model */
 RANDOM alpha ~ MVN(mu, sigma) SUBJECT = id MONITOR = (alpha);

 /* Calculate exponentiated linear predictor */
 DO j = 1 TO 3;
  exp_eta[j] = exp(alpha[j] + beta1 * feature[j] + beta2 * price[j]);
 END;

 /* Calculate baseline category, where alpha4 = 0 */
 exp_eta[4] = exp(beta1 * feature[4] + beta2 * price[4]);

 /* Calculate the denominator by summing over exp_eta. */
 denom = exp_eta[1] + exp_eta[2] + exp_eta[3] + exp_eta[4];

 /* Convert to probabilities */
 DO j = 1 TO 4;
  p[j] = exp_eta[j]/denom;
 END;

 MODEL brand ~ MULTINOM(p);
RUN;
%MEND mcmc_model;
```

The updated code includes statements for the likelihood function. A loop is used to calculate the values for the numerator, exp_eta, for categories $j = 1,2,3$. For the baseline category, the intercept α_4 is constrained to equal 0, and the formula to calculate the numerator is changed accordingly. If, for example, the effect of price and advertising had been allowed to vary by brand as well, then the coefficients for price and advertising would also have to be constrained to equal 0 in the baseline category. A second loop is used to create the vector of probabilities, p, by dividing each of the numerators by their sum, denom. Finally, brand is modeled as a multinomial distribution using the vector of probabilities.

To incorporate random effects for households, a RANDOM statement is used. The RANDOM statement in PROC MCMC is new to SAS 9.3, and simplifies including random effects considerably. The RANDOM statement in the code above includes a separate intercept term for each household/brand combination. Including the MONITOR option displays diagnostics for each of the random effect parameters included. Monitoring all of the random effect parameters in a multinomial model produces a large amount of output – 300 extra parameters in this case. If any of the random effects have not converged, however, then the chain as a whole may not have converged. Therefore, for completeness, the code above monitors each of the random effects. At minimum, some parameters from each of the three categories should be examined for convergence.

With the model coded into a macro, we are ready to "turn the Bayesian crank!" The following section discusses running and assessing convergence of the Markov chain.

ASSESSING CONVERGENCE

A wide variety of options exist for checking the convergence of an MCMC chain. SAS includes many of those options by default. For the sake of brevity, this paper will focus on producing one diagnostic which requires a bit of extra effort in SAS, the Gelman-Rubin diagnostic test. The Gelman-Rubin diagnostic compares the results from two or more chains. Once the chains converge, in principle draws from both chains should be indistinguishable. As such, the Gelman-Rubin diagnostic compares the variance within one chain to the variance between the two chains. If the ratio of the variances – called the potential scale reduction factor – is close to 1, then the chains are indistinguishable, meaning they may have converged on the target distribution.

Figure 1 illustrates this graphically. In the traceplot to the left, both chains are taking draws from the same target distribution, and appear to be indistinguishable. These chains may have converged, and the potential scale reduction factor is approximately 1. By contrast, in the traceplot to the right, the chains do not appear to be covering the same range of the parameter space. As such, these chains have not yet converged, and the potential scale reduction factor is approximately 12, which is much larger than 1.

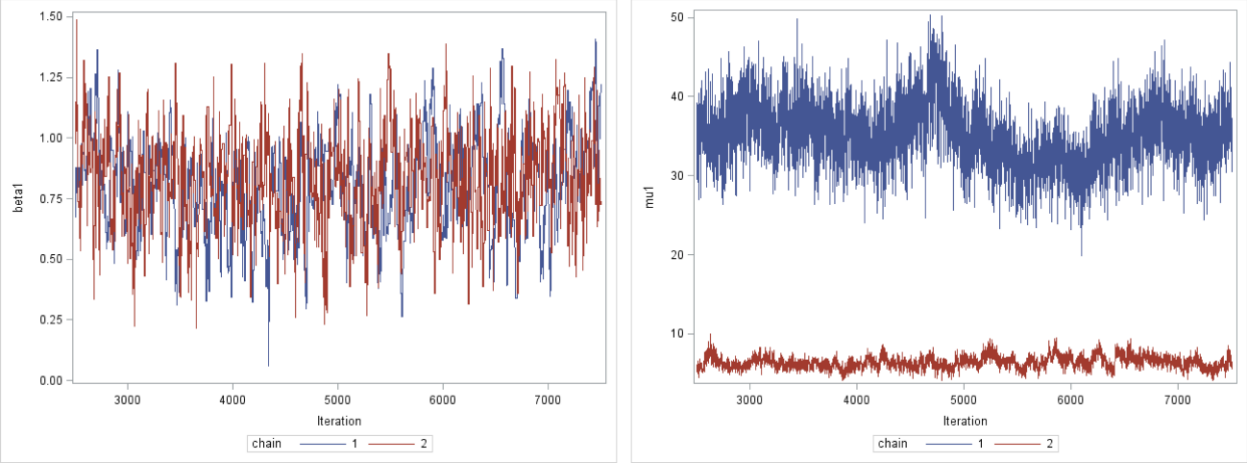

Figure 1. Gelman-Rubin Convergence Diagnostic Tests.

Gelman-Rubin diagnostics are included in the SAS autocall macro %gelman. Given more than one chain to compare, and a list of variables, the %gelman macro will calculate the potential scale reduction factor and its upper bound for each of the variables in the list. Running several MCMC chains can be very time consuming, however, and creating a list of all of the random effects to be tested can require quite a bit of macro coding. The following sections will illustrate how to speed processing by taking advantage of the multiple core processors that most modern computers have, and how to efficiently calculate diagnostics for all of the variables.

RUNNING MULTIPLE MCMC CHAINS

Running a single MCMC chain to convergence can take over thirty minutes on even a high-end PC. Running multiple chains in succession multiplies the amount of running time by the number of chains which are run. Fortunately, since each chain does not depend on the others, the chains can be run simultaneously on computers which have multi-core CPUs, which greatly reduces the amount of time needed. To use multiple cores simultaneously, a new session of SAS must be opened for each chain to be run.

The RSUBMIT command can be used to open additional SAS sessions from within a single SAS program. When invoking separate SAS sessions, however, library locations and macros must be passed to each of the sessions.

Saving the %mcmc_model macro to a separate program file and bringing it in using the %INCLUDE statement simplifies the process of submitting two chains simultaneously.

```
/* First MCMC chain */
SIGNON chain1 SASCMD = "!SASCMD" INHERITLIB=(work=work1);
RSUBMIT chain1 WAIT = NO;
 %INCLUDE "C:\SESUG\mcmc.sas";
 %mcmc_model(5000, 2500, 1, work1.yogurt, work1.posterior1);
ENDRSUBMIT;

/* Second MCMC chain */
SIGNON chain2 SASCMD = "!SASCMD" INHERITLIB=(work=work1);
RSUBMIT chain2 WAIT = NO;
 %INCLUDE "C:\SESUG\mcmc.sas";
 %mcmc_model(5000, 2500, 1, work1.yogurt, work1.posterior2);
ENDRSUBMIT;

/* After both chains have finished running, concatenate the results */
WAITFOR _ALL_ chain1 chain2;
DATA chains;
 SET posterior1 posterior2 (IN = a) NOBS = nmc;
 IF a THEN chain = 2;
 ELSE chain = 1;
 /* Create a variable for the number of draws, required for %gelman macro */
 CALL SYMPUT('nsim', nmc/2);
RUN;
```

Note that the INHERITLIB= option allows the statements in each session to read from and write to the original session's WORK library. For additional information on and methods for using multiple CPU cores simultaneously, see Dilts (2011). Once the chains have completed running, all that remains is to construct the list of variables to be tested.

PREPARING VARIABLE LISTS FOR THE %GELMAN MACRO

Ideally, all the parameters should be tested for convergence. The model constructed above, however, estimates 315 parameters. Fortunately, a small amount of additional programming, suggested by Carpenter (2004), allows us to create macro variables which will form the inputs for the %gelman macro.

```
/* Note that we exclude the automatically created variables Iteration, LogPrior,
LogLike, and LogPost and the variable chain */
PROC CONTENTS DATA = chains (DROP = Iteration Log: chain) OUT = vars NOPRINT;
RUN;

/* Create macro variables for the number of parameters to be tested, and a list of
the names of the parameters to be tested. */
DATA _NULL_;
 LENGTH allvars $32767;
 RETAIN allvars ' ';
 SET vars END = eof NOBS = nvar;
 allvars = TRIM(LEFT(allvars))||' '||LEFT(name);
 IF eof THEN DO;
  CALL SYMPUT('varlist', allvars);
  CALL SYMPUT('nparm', nvar);
 END;
RUN;

%GELMAN(chains, &nparm, &varlist, &nsim, nc = 2);
```

The %gelman macro creates two data sets, _gelman_ests and _gelman_parms, which can be inspected for signs of non-convergence among the parameters. In general, values that are much larger than 1 indicate parameters whose simulated values have not yet converged on the target distribution.

CONCLUSION

This paper demonstrates how to construct and test a hierarchical, multinomial logit model in PROC MCMC. It illustrates three aspects of fitting a Bayesian model, setting the prior distributions, coding the sampling model, and checking convergence of the MCMC chains. In particular, it provides advice for areas where pitfalls commonly occur: testing the prior distributions, writing the full PROC MCMC syntax including the reference category, running several chains simultaneously, and testing many parameters for non-convergence.

In general, the solution to most problems of non-convergence is to run the chain longer. In principle, since the next value for any parameter only depends on the immediately previous value, a chain could be continued by using the previous chain's final values as the new chain's initial values. Extending existing chains, as well as a full treatment of non-convergence, are unfortunately beyond the scope of this paper.

By walking through the steps involved in setting up a hierarchical, multinomial model, this paper illustrates tips for fitting complicated models in a Bayesian framework using SAS 9.3. With a little bit of extra attention, many of the challenges which commonly arise while constructing complicated models in PROC MCMC can be overcome.

APPENDIX: SYNTAX TO IMPORT YOGURT DATA INTO SAS

```
/* Loads yogurt data from the mlogit package in R */
PROC IML;
 SUBMIT / R;
  # Load required libraries for mlogit to work
  library(zoo)
  library(MASS)
  library(miscTools) # maxLik dependency
  library(maxLik)
  library(lmtest)
  library(statmod)
  library(Formula)

  # Load mlogit & yogurt data
  library(mlogit)
  data(Yogurt)
 ENDSUBMIT;

 /* Send to data set in SAS */
 CALL ImportDataSetFromR("yogurt", "Yogurt");
QUIT;

/* Prepare the yogurt data for PROC MCMC */
DATA yogurt;
 SET yogurt;

 /* Construct dummy variables for dependent variable */
 yoplait = (choice = "yoplait");
 dannon = (choice = "dannon");
 hiland = (choice = "hiland");
 weight = (choice = "weight");

 /* Change scale of prices to be in dollars, rather than cents */
 ARRAY prices price:;
 DO OVER prices;
  prices = prices/100;
 END;
RUN;
```

REFERENCES

- Agresti, A. (2007). *An introduction to categorical data analysis* (2nd ed.). Hoboken, NJ: Wiley-Interscience,.
- Carpenter, A. L. (2004). Storing and Using a List of Values in a Macro Variable. *Proceedings of the 2004 Pacific Northwest SAS Users' Group*. Portland, OR. Retrieved from http://www.lexjansen.com/pnwsug/2004/c_cc_storing_and_using_a_lis.pdf

- Chen, Z., & Kuo, L. (2001). A Note on the Estimation of the Multinomial Logit Model with Random Effects. *The American Statistician*, *55*(2), 89–95.
- Dilts, E. (2011). Use Your Cores! An Introduction to Multi-Core Processing with SAS. *SESUG 2011: The Proceedings of the SouthEast SAS Users Group*. Alexandria, VA.
- Jain, D. C., Vilcassim, N. J., & Chintagunta, P. K. (1994). A Random-Coefficients Logit Brand-Choice Model Applied to Panel Data. *Journal of Business & Economic Statistics*, *12*(3), 317–328. doi:10.2307/1392088
- Stokes, M. (2011). On Deck: SAS/STAT 9.3. *SESUG 2011: The Proceedings of the SouthEast SAS Users Group*. Alexandria, VA.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jacob Fisher
Department of Sociology
Box 90088
Duke University
Durham, NC 27708
(919) 660-5603
jcf26@duke.edu