Bayesian logistic models with MCMCglmm: A brief tutorial

Version as of 27.04.2015

Natalia Levshina, F.R.S. – FNRS, Université catholique de Louvain

www.natalialevshina.com

1. Bayesian inference vs. null hypothesis testing

Null hypothesis testing (aka frequentist approach) contrasts a null hypothesis with an alternative hypothesis. The *p*-value shows the probability of observing a statistic if the null hypothesis is true.

Advantages:

- dominant (many packages and functions)
- computationally tractable (many tests can be performed with paper and pencil)

Disadvantages:

- conceptually paradoxical: it is impossible to evaluate the probability of a hypothesis. Instead, we test the data given a hypothesis.

Bayesian inference is based on Bayes' theorem, a procedure for revising and updating the probability of some event in the light of new evidence:

$p(B|A) = p(A|B) \times p(B)/p(A)$

In other words, the posterior probability is proportional to prior probability multiplied by the likelihood. The frequentist approach only deals with the likelihood.

Advantages:

- conceptually coherent (we can test a hypothesis given the data)
- one can compare the fit of different models based on different data and methodologies
- very complex models possible

Disadvantages:

- not always clear how to determine the priors
- computationally intensive (e.g. Markov chain Monte Carlo algorithm)

However, both the frequentist and Bayesian methods allow for parameter estimation, prediction of data values and model comparison.

2. Logistic models with MCMCglmm: a case study of letting constructions

2.1. Introduction

MCMCglmm is a package developed by Jarrod Hadfield for fitting generalized linear mixed models using Markov chain Monte Carlo techniques:

- linear

- Poisson

- **multinomial**

- zero-inflated Poisson

Documentation:

J. Hadfield's Course Notes:

http://cran.r-project.org/web/packages/MCMCglmm/vignettes/CourseNotes.pdf

J. Hadfield's Tutorial:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.160.5098&rep=rep1&type=pdf

```
> library(MCMCglmm)
```

2.2. The data

Open an R session with letting.RData. The RData object with the data set is available at www.natalialevshina.com/statistics.html

```
> str(letting)
'data.frame': 891 obs. of  5 variables:
 $ Vinf   : Factor w/ 397 levels "access","accommodate",..: 168 67
207 166 345 207 363 20 242 24 ...
 $ LetVerb: Factor w/ 3 levels "let","allow",..: 1 2 1 2 2 2 1 2 1 2
...
 $ Channel: Factor w/ 2 levels "Spoken","Written": 1 2 2 2 2 1 2 2 2
2 ...
```

```
 $ CrSem  : Factor w/ 2 levels "Anim","Inanim": 1 1 1 1 1 1 1 1 1 2
...
 $ MS     : num  0.005558 0.000511 0.000745 0.000637 0.000748 ...
```

The data set contains corpus observations of near-synonymous constructions let, allow and permit + (to)-Infinitive. The variables are as follows:

- Vinf: the infinitive (random effect)
- LetVerb: let, allow or permit
- Channel: Spoken or Written
- CrSem (semantics of the Causer): Anim or Inanim
- MS: minimum sensitivity, a collostructional measure that reflects the attraction between the letting verb and the infinitive

2.3. The MCMCglmm function

First, we determine the priors for a model with the categorical (multinomial or binomial) response (see Hadfield's Tutorial, Table 1):

```
> k <- length(levels(letting$LetVerb))

> I <- diag(k-1)

> J <- matrix(rep(1, (k-1)^2), c(k-1, k-1))

> priors = list(R = list(fix=1, V=(1/k) * (I + J), n = k - 1),
                G = list(G1 = list(V = diag(k - 1), n = k - 1)))
```

Notes:

- R is the residual error structure
- G is a list for every random effect (so far, only one…)
- V is a (co)variance matrix
- n is the degrees of freedom (k – 1)

A model with a random intercept and Channel, CrSem and MS as fixed effects can be fit as follows:

```
> m <- MCMCglmm(LetVerb ~ -1 + trait + trait:(Channel + CrSem + MS),
random = ~us(trait):Vinf, rcov = ~ us(trait):units, data = letting,
family = "categorical", prior = priors, verbose = TRUE, burnin =
10000, nitt = 60000, thin = 50)
```

Comments:

- `LetVerb ~ -1 + trait + trait:(Channel + CrSem + MS) #` formula
- `trait #` a reserved variable that expresses the latent variables: a log-odds ratio of observing allow vs. observing let and that of observing permit vs. observing let.
- `random = ~us(trait):Vinf #` specification of the random intercept
- `rcov = ~ us(trait):units #` residual covariance structure ('units' are individual observations)
- `burnin = 10000 #` the number of initial MCMC iterations that are discarded, by default 3000. The reason for discarding the burn-in component is that the initial samples can follow a very different distribution. An illustration of a situation without burn-in is provided in Figure 1. Compare this pattern with Figure 2 with a large burn-in value.
- `nitt = 60000 #` the number of MCMC iterations, by default 13000
- `thin = 50 #` a thinning interval, by default 10.

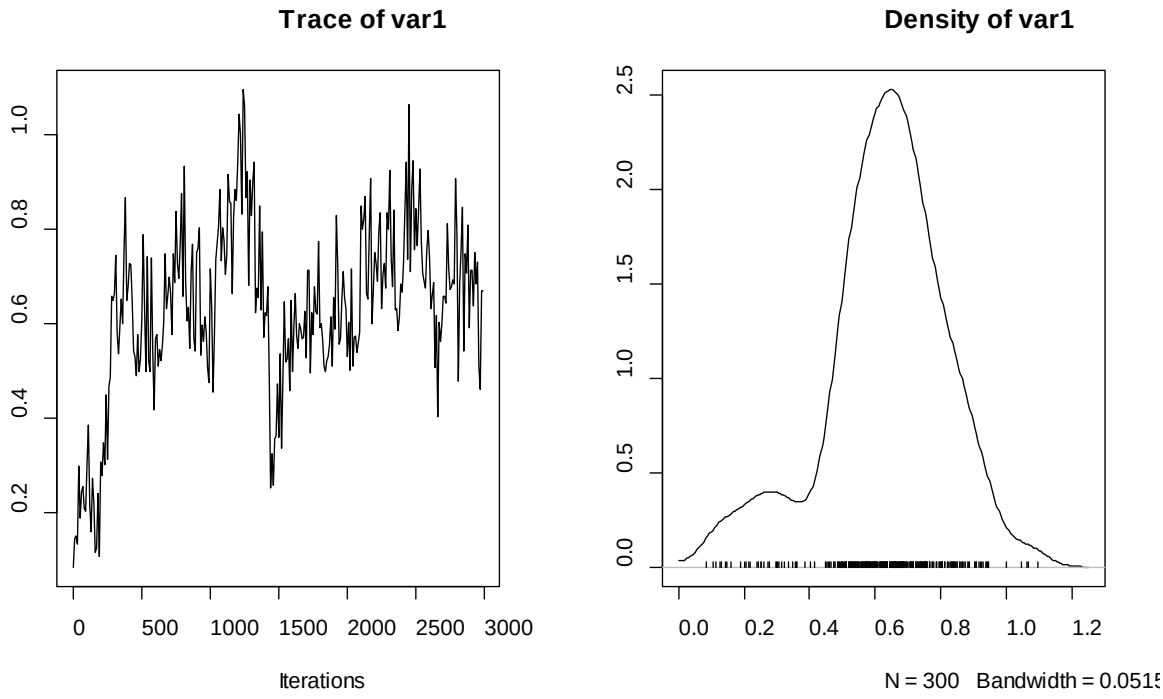

Figure 1. The trace graph of MCMC with a zero burn-in parameter.

Note: burnin, nitt and thin should be specified in such a way that you have 1000- 2000 iterations at least. To calculate the number of iterations, use this simple formula:

sample size = (nitt – burnin)/thin

if `Verbose = TRUE`:

…

```
  Acceptance ratio for latent scores = 0.316846

                      MCMC iteration = 58000

  Acceptance ratio for latent scores = 0.317089

                      MCMC iteration = 59000

  Acceptance ratio for latent scores = 0.316019

                      MCMC iteration = 60000

  Acceptance ratio for latent scores = 0.315633
```

The acceptance ratios show how fast the algorithm explores the space. According to a rule of thumb, the ratios should be optimally from 0.25 to 0.5.

```
> summary(m)

 Iterations = 10001:59951
 Thinning interval  = 50
 Sample size  = 1000

 DIC: 1296.195

 G-structure:  ~us(trait):Vinf

                                    post.mean l-95% CI u-95% CI
eff.samp
LetVerb.allow:LetVerb.allow.Vinf       2.352   0.9561    4.188
38.772
LetVerb.permit:LetVerb.allow.Vinf      4.363   1.5099    8.489
28.293
LetVerb.allow:LetVerb.permit.Vinf      4.363   1.5099    8.489
28.293
LetVerb.permit:LetVerb.permit.Vinf    14.194   3.0994   26.405
5.148

 R-structure:  ~us(trait):units

                                     post.mean l-95% CI u-95% CI
eff.samp
LetVerb.allow:LetVerb.allow.units      0.6667   0.6667   0.6667
0
LetVerb.permit:LetVerb.allow.units     0.3333   0.3333   0.3333
0
LetVerb.allow:LetVerb.permit.units     0.3333   0.3333   0.3333
0
LetVerb.permit:LetVerb.permit.units    0.6667   0.6667   0.6667
0
```

```
 Location effects: LetVerb ~ -1 + trait + trait:(Channel + CrSem +
MS)

                                    post.mean   l-95% CI   u-95% CI
eff.samp  pMCMC
traitLetVerb.allow                    -0.4873    -1.4872     0.6192
207.413  0.366
traitLetVerb.permit                   -2.6048    -5.3344     0.4593
25.822  0.062 .
traitLetVerb.allow:ChannelWritten      1.3618     0.3422     2.3420
151.997  0.004 **
traitLetVerb.permit:ChannelWritten     4.4596     1.6742     7.2702
23.412 <0.001 ***
traitLetVerb.allow:CrSemInanim         3.6148     2.8364     4.4012
68.105 <0.001 ***
traitLetVerb.permit:CrSemInanim        4.5869     3.6136     5.5863
81.102 <0.001 ***
traitLetVerb.allow:MS               -695.6722  -931.4232  -390.9225
89.497 <0.001 ***
traitLetVerb.permit:MS             -3445.0526 -4795.8705 -2238.4426
5.911 <0.001 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Notes:

- DIC is Deviance Information Criterion, also accessible as `m$DIC`. Similar to Akaike’s Information Criterion (AIC) in that they both serve as parsimony criteria.
- `post.mean`: posterior means (similar to coefficient estimates in a frequentist logistic regression model), log-odds.
- It is also possible to obtain the 95% highest posterior density intervals separately (e.g. for plotting purposes). They are functionally similar to 95% confidence intervals in frequentist models:

```
> HPDinterval(m$Sol) #Highest Posterior Density Intervals. By
default, prob = 0.95

                                         lower         upper
traitLetVerb.allow                    -1.4872301     0.6192129
traitLetVerb.permit                   -5.3343778     0.4593211
traitLetVerb.allow:ChannelWritten      0.3421836     2.3420162
traitLetVerb.permit:ChannelWritten     1.6741524     7.2701989
traitLetVerb.allow:CrSemInanim         2.8364339     4.4011622
traitLetVerb.permit:CrSemInanim        3.6135503     5.5863309
traitLetVerb.allow:MS               -931.4232410  -390.9224773
traitLetVerb.permit:MS             -4795.8704925 -2238.4426179
attr(,"Probability")
[1] 0.95
```

2.4. Examining the random intercepts

To examine the random intercepts, one should add `pr = TRUE` (`FALSE` by default). This saves the posterior distribution of random effects in the Solution mcmc object:

```
> m1 = MCMCglmm(LetVerb ~ -1 + trait + trait:(Channel + CrSem + MS),
random = ~us(trait):Vinf, rcov = ~ us(trait):units, data = letting,
family = "categorical", prior = priors, verbose = TRUE, burnin =
10000, nitt = 60000, thin = 50, pr = TRUE)
```

In this case, the Solution object contains 8 + 397 × 2 = 802 columns, which represent 8 fixed effects and two times 397 columns for the infinitives, first for allow vs. let and second for permit vs. let. The data contain the posterior probabilities of these parameters for 1000 iterations (rows):

```
> dim(m1$Sol)
[1] 1000  802
```

There are two ways to obtain measures that are similar to Best Linear Unbiased Predictors in generalized linear mixed-effect models. It is possible to compute both the posterior modes (the values with the highest probability) and the posterior means.

```
> rmode <- posterior.mode(m1$Sol[,-c(1:8)])
> rmode.allow <- rmode[1:397]
> rmode.permit <- rmode[398:794]

> rmean <- colMeans(m1$Sol[,-c(1:8)])
> rmean.allow <- rmean[1:397]
> rmean.permit <- rmean[398:794]
```

The modes and means are not identical, but very similar, as the high correlation coefficients suggest:

```
> cor(rmode.allow, rmean.allow)

[1] 0.92683

> cor(rmode.permit, rmean.permit)

[1] 0.9535617
```

Figure 2 displays the quantile-quantile (Q-Q) plots of the posterior means and modes for allow (vs. let) and permit (vs. let).

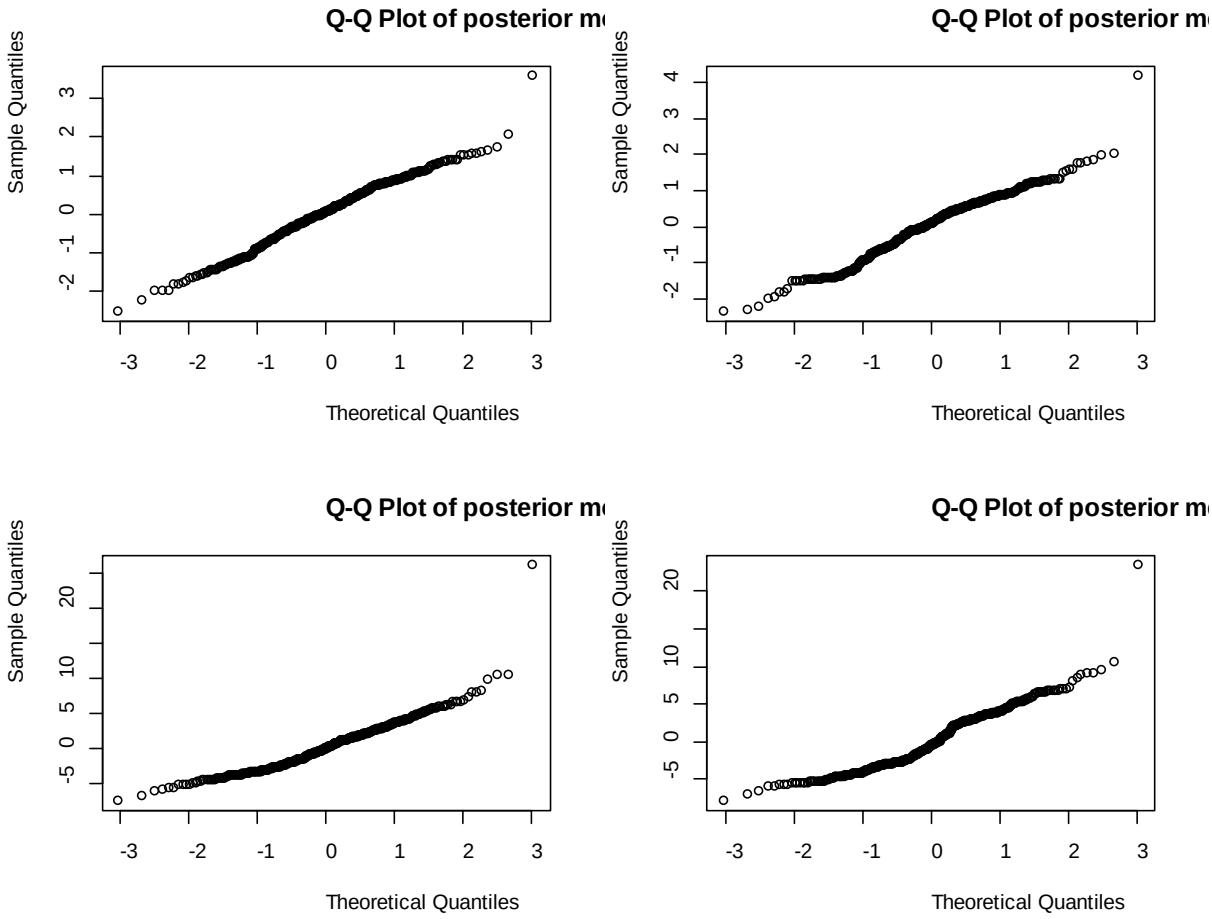

Figure 2. Q-Q plots of the posterior means and modes of random intercepts for allow vs. let (top) and permit vs. let (bottom).

2.5. Model Diagnostics

The main requirement is that the chain should not display autocorrelation. It is observed when there are trends in the data. Autocorrelation means that a next value is influenced by the previous value. In case of strong autocorrelations, the model estimates (posterior means or modes) are unreliable. In other words, the chain does not mix well. Autocorrelations can be tested as shown below. According to a rule of thumb, the values should be < 0.1.

```
> autocorr(m$Sol) # m1 may take very long!
```

It is also useful to examine the traces visually with the help of `plot(m$Sol)`. Figure 3 displays the results for one of the model parameters (`allow:ChannelWritten`).

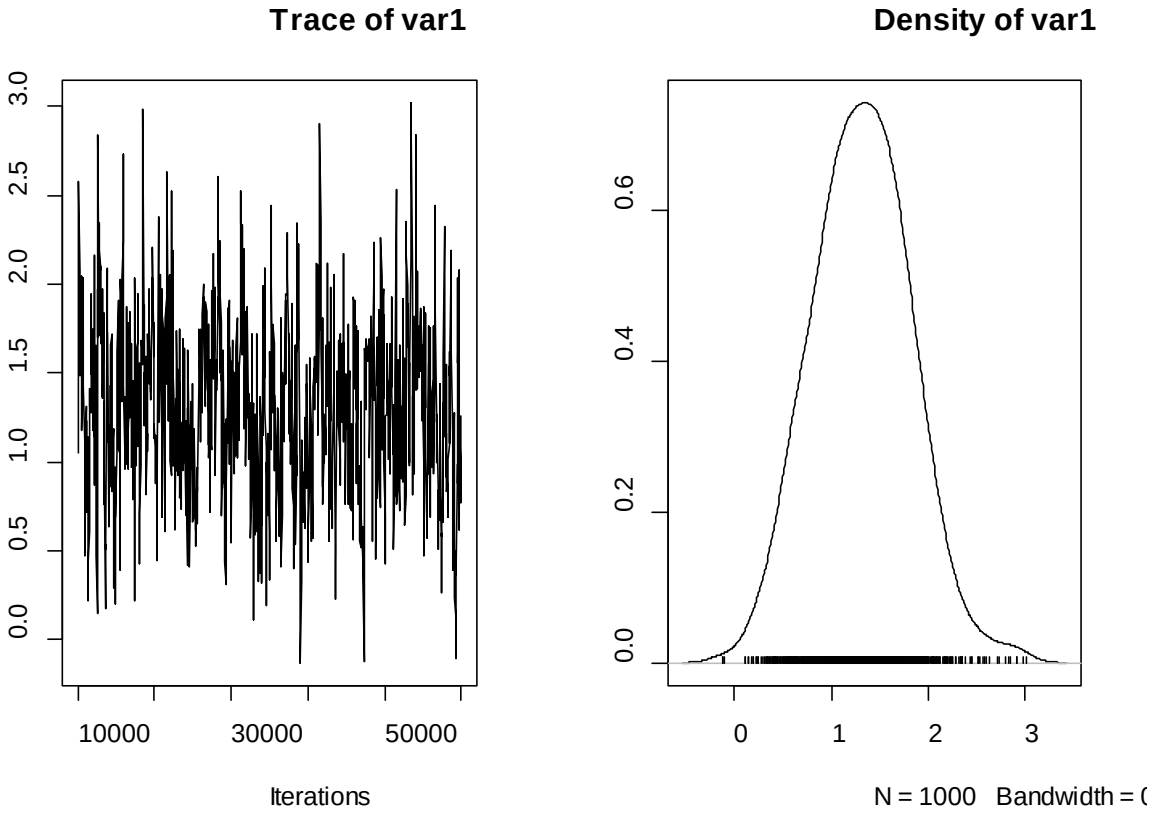

Figure 3. Posterior distribution of a model parameter. Left: Time series of a parameter in the model as MCMC iterates (note that the range of x-values is from 10000 to 60000). Right: probability density estimate of the parameter. The peak of the distribution (the posterior mode) is the most likely value.

For comparison, consider an example of a strong autocorrelation in Figure 4.

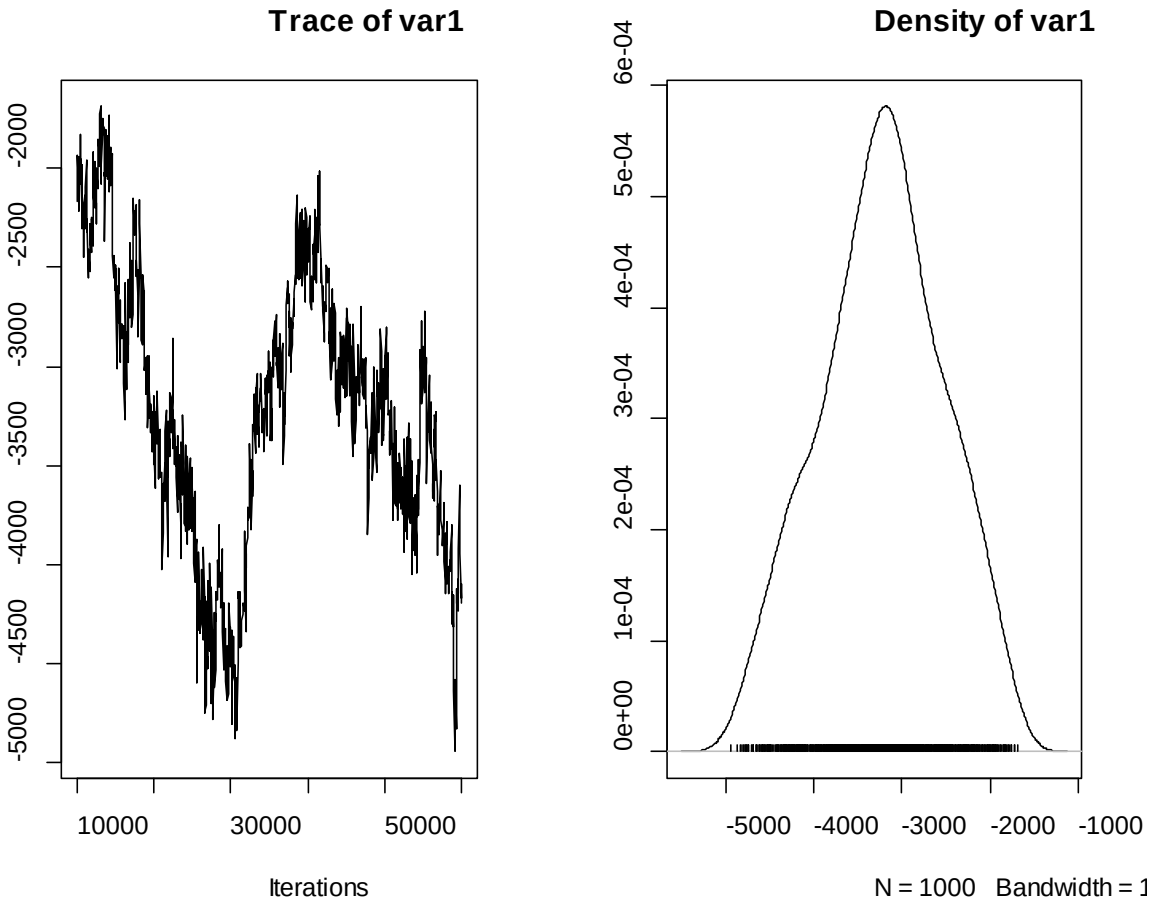

Figure 4. An example of autocorrelated data (MCMC iterations).

If the chain does not mix well, you can do the following:

- increase the burn-in period (`burnin`)
- increase the number of iterations (`nitt`)
- increase the thinning interval (`thin`)

3. Final remarks

- The larger the data set, the less influence the priors have on the posterior probabilities. With a large data set, the problem of priors is negligible.
- It is also possible to model the response represented by counts of the outcomes in two or more columns. In that case, the columns should be combined by using `cbind(Outcome1, Outcome2, Outcome3)`. This is called a multi-response model. The family name is then "multinomial*k*", where *k* is the number of outcomes, e.g. "multinomial2". The final column will be treated as the baseline category.

Reading suggestions

A useful blog and discussion: https://hlplab.wordpress.com/2009/05/07/multinomial-random-effects-models-in-r/

Krushke, J. 2011. Doing Bayesian Data Analysis: A tutorial with R and BUGS. Oxford: Elsevier.