Log-Linear Modeling with Missing and Coarsened Values Using the `cvam` Package

Joseph L. Schafer*

February 22, 2023

Abstract

Log-linear models, when applied to frequencies in a multiway table, describe associations among categorical variables (factors) that define the dimensions of the table. The R package `cvam` fits log-linear models to factors that may have missing and coarsened values, including latent-class models. This document provides a quick review of log-linear models and explains `cvam`'s modeling functions with examples. We assume the reader understands basic ideas of factors and coarsened factors, as described in the separate document *Understanding Coarsened Factors in `cvam`*.

*Office of the Associate Director for Research and Methodology, United States Census Bureau, Washington, DC 20233, `joseph.l.schafer@census.gov`. This article is released to inform interested parties of ongoing research and to encourage discussion. The views expressed are those of the author and not necessarily those of the U.S. Census Bureau.

1 Introduction

In the companion document *Understanding Coarsened Factors in cvam*, we introduced a new class of R objects for storing categorical variables with coarsened values, but said little about how or why they ought to be used. In this present document, we show how to model relationships among categorical variables using functions in the `cvam` package. These functions were designed with coarsened factors in mind, but they also accept ordinary factors with or without missing values.

Before diving in to the specifics of `cvam`, we give some background material to show how log-linear models work with complete data. In Section 2, we review four different types of objects for storing categorical variables and show how to convert data from one format to another. Section 3 gives a quick overview of log-linear modeling in the complete-data case, establishing the notation and assumptions. The remaining sections cover the special `cvam` functions and object classes, with examples using datasets distributed with the package. Technical details of the computational methods are provided in the appendices.

2 Storing and manipulating categorical data

2.1 Multidimensional contingency tables

Contingency tables are arrays of frequencies that result from classifying individuals or sample units by one or more categorical factors. Each dimension of the array corresponds to a factor. In R, a contingency table object has a `class` attribute of `"table"` or `c("xtabs", "table")`, depending on which function (`table` or `xtabs`) created it.

A well known example of a contingency table pertaining to gender bias in graduate admissions was published by Bickel et al. (1975). The $2 \times 2 \times 6$ table `UCBAdmissions`, distributed with R as part of the `datasets` package, classifies 4,526 applicants to U.C. Berkeley by admission status, sex, and department.

```
> # show the structure of the object
> str(UCBAdmissions)

 'table' num [1:2, 1:2, 1:6] 512 313 89 19 353 207 17 8 120 205 ...
 - attr(*, "dimnames")=List of 3
  ..$ Admit : chr [1:2] "Admitted" "Rejected"
  ..$ Gender: chr [1:2] "Male" "Female"
  ..$ Dept  : chr [1:6] "A" "B" "C" "D" ...

> # display slices of the table corresponding to Dept "A" and "B"
> UCBAdmissions[,,1:2]
```

```
, , Dept = A

          Gender
Admit      Male Female
  Admitted  512     89
  Rejected  313     19

, , Dept = B

          Gender
Admit      Male Female
  Admitted  353     17
  Rejected  207      8
```

To collapse a contingency table over one or more of its dimensions and obtain marginal frequencies, use the `apply` function with argument `FUN=sum`.

```
> # one-way table for Dept (dimension 3)
> apply( UCBAdmissions, 3, sum )

  A   B   C   D   E   F
933 585 918 792 584 714

> # two-way table for Gender x Admit (dimensions 2 and 1),
> # with chisquare test for independence
> GenderByAdmit <- apply( UCBAdmissions, c(2,1), sum )
> chisq.test( GenderByAdmit )

        Pearson's Chi-squared test with Yates' continuity correction

data:  GenderByAdmit
X-squared = 91.61, df = 1, p-value < 2.2e-16
```

Contingency tables offer compact storage when the number of factors is small, but the number of factors grows, the size of these tables increases rapidly. High-dimensional tables tend to be sparse, with many of the cells containing very few or no observations.

2.2 Data frame with microdata

Categorical variables may be kept as microdata, in a data frame with one row per individual or sample unit and one column for each factor. To illustrate, we simulated a microdata version of the U.C. Berkeley Admissions dataset with 4,526 rows and three factors. The data frame, called `microUCBAdmissions`, is distributed with the `cvam` package.

```
> library(cvam)
> # display the first few rows
> head(microUCBAdmissions)
```

```
     Admit Gender Dept
1 Rejected Female    C
2 Rejected Female    F
3 Rejected   Male    D
4 Rejected   Male    D
5 Rejected Female    E
6 Admitted   Male    B
```

Microdata can be transformed into frequency tables by the functions `table` or `xtabs`.

```
> dF <- microUCBAdmissions  # to save typing
> # this reproduces the 3-way table UCBAdmissions
> result <- table( Admit = dF$Admit,
+    Gender = dF$Gender, Dept = dF$Dept )
> str(result)

 'table' int [1:2, 1:2, 1:6] 512 313 89 19 353 207 17 8 120 205 ...
 - attr(*, "dimnames")=List of 3
  ..$ Admit : chr [1:2] "Admitted" "Rejected"
  ..$ Gender: chr [1:2] "Male" "Female"
  ..$ Dept  : chr [1:6] "A" "B" "C" "D" ...

> all.equal( result, UCBAdmissions )

[1] TRUE

> # do the same thing with xtabs, which accepts formula notation
> result <- xtabs( ~ Admit + Gender + Dept, data=microUCBAdmissions )
```

For this example, the microdata object is much larger than the contingency table, but that is not always the case. Microdata may be more compact when the number of factors is large, because combinations of factor levels that do not occur in the sample are absent. Microdata files may also include variables that are continuously distributed, which may be impractical for contingency-table storage unless they are first binned into factors with a small number of levels.

2.3 Data frame with grouped data

In the grouped data frame format, units having identical values for all variables are collapsed into a single row of the data frame, and another variable is added to record frequencies. A grouped data frame can be produced from a `table` or `xtabs` object using `as.data.frame`

```
> result <- as.data.frame(UCBAdmissions)
> head(result)

     Admit Gender Dept Freq
1 Admitted   Male    A  512
2 Rejected   Male    A  313
```

```
3 Admitted Female    A   89
4 Rejected Female    A   19
5 Admitted   Male    B  353
6 Rejected   Male    B  207
```

The result from `as.data.frame` has one row for each cell in the table, including the empty cells.

To convert microdata to grouped data, use the `aggregate` function.

```
> # create a Freq variable and fill it with ones
> microUCBAdmissions$Freq <- 1
> # use aggregate to sum the Freq variable within categories of
> # Admit, Gender, and Dept
> result <- aggregate( Freq ~ Admit + Gender + Dept,
+    data=microUCBAdmissions, FUN=sum )
> head(result)
```

```
     Admit Gender Dept Freq
1 Admitted   Male    A  512
2 Rejected   Male    A  313
3 Admitted Female    A   89
4 Rejected Female    A   19
5 Admitted   Male    B  353
6 Rejected   Male    B  207
```

The grouped data frame has one row for each unique combination of the grouping factors (i.e., the variables on the right-hand side of the formula) that appear in the microdata.

When using `aggregate`, it is important to keep two behaviors in mind.

- *The resulting data frame does not include empty cells.* Conventional log-linear modeling with `glm`, which we will demonstrate in Section 3, requires a data frame that includes rows for the empty cells with frequencies of zero. To include the empty cells, process the microdata with `table` or `xtabs` followed by `as.data.frame`.
- *By default, rows of the microdata frame that contain missing values are dropped from the result.* A primary reason for using `cvam` is that it will allow us to fit log-linear models to all records, including those with missing or coarsened values.

2.4 Flat tables

Flat tables, produced by `ftable`, are two-dimensional representations of contingency tables that are convenient for display and publication. They can be created from contingency tables and from microdata. Some examples are shown below.

```
> # from a table, specifying the row and column variables
> ftable( UCBAdmissions, row.vars=c("Dept","Gender"), col.vars="Admit")

            Admit Admitted Rejected
Dept Gender
A    Male             512      313
     Female            89       19
B    Male             353      207
     Female            17        8
C    Male             120      205
     Female           202      391
D    Male             138      279
     Female           131      244
E    Male              53      138
     Female            94      299
F    Male              22      351
     Female            24      317

> # from microdata, using a formula interface
> ftable( Admit ~ Dept + Gender, data=microUCBAdmissions )

            Admit Admitted Rejected
Dept Gender
A    Male             512      313
     Female            89       19
B    Male             353      207
     Female            17        8
C    Male             120      205
     Female           202      391
D    Male             138      279
     Female           131      244
E    Male              53      138
     Female            94      299
F    Male              22      351
     Female            24      317

> # with one row variable and two column variables
> ftable( UCBAdmissions, row.vars=c("Dept"),
+     col.vars=c("Gender","Admit"))

     Gender     Male            Female
     Admit  Admitted Rejected Admitted Rejected
Dept
A               512      313       89       19
B               353      207       17        8
C               120      205      202      391
D               138      279      131      244
E                53      138       94      299
F                22      351       24      317

> # omitted variables are summed over
> ftable( Admit ~ Gender, data=microUCBAdmissions )

       Admit Admitted Rejected
Gender
Male            1198     1493
Female           557     1278
```

The `ftable` function was designed for visual displays, but not for rearranging categorical data for subsequent analysis. Contributed packages for managing and reshaping are be helpful in that regard, especially `reshape2` (Wickham, 2007).

3 Fitting log-linear models with complete data

3.1 What is a log-linear model?

Log-linear models describe relationships among the factors in a cross-classification. Consider a pair of categorical variables A and B recorded for a sample of units. Let a and b denote possible values for A and B, and let $\pi_{ab} = P(A = a, B = b)$. If the two variables are independent, then the relationship

$$\pi_{ab} \;=\; P(A = a) \;\times\; P(B = b)$$

must hold for every combination of a and b. Assuming that none of the probabilities are zero, independence is equivalent to saying that the log-probabilities have an additive structure,

$$\log \pi_{ab} \;=\; \log P(A = a) \;+\; \log P(B = b). \tag{1}$$

In a manner similar to analysis of variance (ANOVA) for a two-way factorial designs, we may decompose the log-cell probabilities as

$$\log \pi_{ab} \;=\; \lambda \;+\; \lambda^A_a \;+\; \lambda^B_b \;+\; \lambda^{AB}_{ab}, \tag{2}$$

where the λ terms sum to zero over any subscript,

$$\sum_a \lambda^A_a \;=\; \sum_b \lambda^B_b \;=\; \sum_a \lambda^{AB}_{ab} = \sum_b \lambda^{AB}_{ab} \;=\; 0.$$

The decomposition (2) implies independence (1) if all the λ^{AB}_{ab} terms are zero.

Despite the apparent similarity to ANOVA, there are some important differences.

- Two-way ANOVA involves a third variable, a response that is being predicted by the two factors. In the log-linear model, the "response" is not a random variable, but a set of log-probabilities defining the joint distribution of the factors. The log-linear model is a joint model for A and B, not a model for some other variable given A and B.
- In ANOVA, the parameter analogous to λ in Equation (2) is a grand mean, the expected value of the response variable averaged over all cells of the design. In the log-linear model, λ is not a parameter, but a normalizing constant chosen to ensure that the cell probabilities π_{ab} sum to one.
- In ANOVA, the simplest model worth considering is the intercept-only model in which all the main effects λ^A_a and λ^B_a and all the interactions λ^{AB}_{ab} are zero. In log-linear modeling, the simplest model worth considering includes λ^A_a and λ^B_a but omits λ^{AB}_{ab}. Dropping λ^A_a and λ^{AB}_{ab} would imply not only that A is independent of B, but also that A is uniformly distributed with equal probabilities over its levels, a model that is rarely of interest.

To avoid a proliferation of letters and subscripts, we now switch to notation that can accommodate any number of variables. The full details of this notation are given in Appendix A. Let $\boldsymbol{V} = (V_1, \ldots, V_J)$ denote a vector of J categorical variables to be modeled, and let $\boldsymbol{v} = (v_1, \ldots, v_J)$ denote a possible value for $\boldsymbol{V}$. (Throughout this document, vectors, matrices and higher-dimensional arrays are written in boldface type, and scalars are in lightface.) The probability that a randomly selected unit in the population has $\boldsymbol{V} = \boldsymbol{v}$ is

$$\pi_{\boldsymbol{v}} \;=\; P(\boldsymbol{V} = \boldsymbol{v}),$$

and the array of these probabilities for all possible $\boldsymbol{v}$ is denoted by $\boldsymbol{\pi}$. The shape of $\boldsymbol{\pi}$ depends on how the data are arranged. If the data are stored as a contingency table, then $\boldsymbol{\pi}$ is a J-dimensional array whose dimensions correspond to the factors $V_1, \ldots, V_J$. If the data are stored as a data frame with grouped observations, then $\boldsymbol{\pi}$ is a vector with one element for each row of the data frame. Either way, $\boldsymbol{\pi}$ has the same number of elements, assuming that the data frame includes rows for any empty cells. Obviously, the sum of these elements over all cells, which we denote by π_+, must be equal to one.

If $V_1, \ldots, V_J$ are fully observed in a random sample of N units, we can form the contingency table of frequencies which we call $\boldsymbol{f}$, with the same size and shape as $\boldsymbol{\pi}$. An element of $\boldsymbol{f}$ is $\boldsymbol{f}_{\boldsymbol{v}}$. Regarding N and $\boldsymbol{\pi}$ as fixed, $\boldsymbol{f}$ has a multinomial distribution

$$\boldsymbol{f} \,|\, N, \boldsymbol{\pi} \;\sim\; \text{Mult}(N, \boldsymbol{\pi}). \tag{3}$$

A log-linear model is an assumption that the vectorized $\boldsymbol{\pi}$ has the form

$$\log \boldsymbol{\pi} \;=\; \boldsymbol{X}\boldsymbol{\lambda}, \tag{4}$$

where $\boldsymbol{X}$ is a known model matrix with p columns, $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)$ is a vector of unknown coefficients, and 'log $\boldsymbol{\pi}$' means taking the natural logarithm of each element of $\boldsymbol{\pi}$. In most cases, the first column of $\boldsymbol{X}$ is filled with ones, and the remaining columns are terms for the main effects of the factors $V_1, \ldots, V_J$ and the interactions among them, the same kind of terms that would appear in a regression/ANOVA model with $V_1, \ldots, V_J$ as predictors. A more nuanced definition of a log-linear model is given in Appendix C.

Under the multinomial distribution, the expected value of $\boldsymbol{f}$ is $\boldsymbol{\mu} = N\boldsymbol{\pi}$. The log-linear model can be written in terms of $\boldsymbol{\mu}$ as

$$\log \boldsymbol{\mu} \;=\; \boldsymbol{X}\boldsymbol{\beta}. \tag{5}$$

If the first column of $\boldsymbol{X}$ is filled with ones, then $\boldsymbol{\beta}$ is identical to $\boldsymbol{\lambda}$, except that the first element has been shifted upward by $\log N$. Just as the first element of $\boldsymbol{\lambda}$ is a normalizing constant that ensures $\pi_+ = 1.0$, the first element of $\boldsymbol{\beta}$ is a normalizing constant that ensures $\mu_+ = N$.

3.2 Fitting techniques

The most common way to fit a log-linear model is to treat it as a generalized linear model with a Poisson response, regressing $\boldsymbol{f}$ on $\boldsymbol{X}$ with a log link (McCullagh and Nelder, 1989; Agresti, 2013). The Poisson and multinomial models differ in certain ways. Notably, the Poisson model regards N as a random variable, turning the first element of $\boldsymbol{\beta}$ into a free parameter. Despite those differences, Poisson regression serves as an appropriate surrogate for the multinomial model and does give correct answers, for reasons explained in Appendix B.

In the special case where the number of columns of $\boldsymbol{X}$ and its number of rows are equal, the model is said to be saturated. Under a saturated model, the ML estimate is simply $\hat{\boldsymbol{\mu}} = \boldsymbol{f}$, and (assuming there are no zeros) the coefficients can be obtained by $\hat{\boldsymbol{\beta}} = \boldsymbol{X}^{-1} \log \hat{\boldsymbol{\mu}}$, because $\boldsymbol{X}$ is invertible. For non-saturated models, $\hat{\boldsymbol{\beta}}$ may be computed by a Newton-Raphson (NR) procedure, also known as Fisher scoring or iteratively reweighted least squares (McCullagh and Nelder, 1989). Details of NR are given in Appendix D. The fitted values from the Poisson regression are $\hat{\boldsymbol{\mu}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$. The fitted values for the multinomial model are $\hat{\boldsymbol{\pi}} = N^{-1}\hat{\boldsymbol{\mu}}$, and the estimated coefficients for the multinomial model are

$$\begin{aligned} \hat{\lambda}_1 &= \hat{\beta}_1 - \log N, \\ \hat{\lambda}_k &= \hat{\beta}_k \quad \text{for } k = 2, \ldots, p. \end{aligned}$$

The estimated covariance matrix for $\hat{\boldsymbol{\beta}}$ supplied by the Poisson-regression software is a appropriate for both $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\lambda}}$ under the Poisson model, except that the first row and column should be ignored (i.e., set to zero), because the first element of $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$ are normalizing constants.

An older method for fitting log-linear models, called iterative proportional fitting (IPF), operates directly on margins of the contingency table (Bishop et al., 1975). IPF does not use a model matrix $\boldsymbol{X}$, so it does not provide estimated coefficients $\hat{\boldsymbol{\beta}}$ or $\hat{\boldsymbol{\lambda}}$, but the fitted values $\hat{\boldsymbol{\mu}}$ and $\hat{\pi}$ from IPF are identical to those that we would get from NR, if the latter converges. IPF is more stable than NR and does not encounter difficulty if the ML estimate is non-unique or lies on a boundary. (That stability is not always desirable, however, because if a problem exists, many users would want to be warned.) For some log-linear models, ML estimates are available in closed form; these are said to be decomposable (Lauritzen, 1996). If a model is decomposable, IPF converges after a single cycle, but NR does not. If a model is non-decomposable, IPF tends to converge more slowly.

3.3 Model interpretation

In a log-linear model, the meaning of the coefficients in $\boldsymbol{\lambda}$ or $\boldsymbol{\beta}$ depends on the coding scheme used to define the model matrix. These coefficients can be difficult to interpret, and instead of focusing on them, it is more important to understand the model holistically in terms of the kinds of relationships among $V_{i1}, \ldots, V_{iJ}$ that it allows.

Most readers should be familiar with the formula notation used by the R functions `lm` and `glm`. For example, `Y ~ V1 + V2` denotes a regression for predicting `Y` with main effects for `V1` and `V2`. A model with main effects and interaction terms is `Y ~ V1 + V2 + V1:V2` or, equivalently, `Y ~ V1*V2`. Redundancy in a formula is not a problem; for example, if you specify `Y ~ V1 + V2 + V1*V2`, the main effects for `V1` and `V2` will not be included twice. Even `Y ~ V1 + V1 + V1` will not cause an error, because R's formula interpreter removes duplicate symbols automatically.

In this discussion, we will represent log-linear models by one-sided formulas, with nothing on the left-hand side. By convention, we will only consider models that are *hierarchical*, which means that, if an interaction among a group of variables is present, then all main effects and lower-order interactions within the group must also be present. For example, if we include `V1:V2:V3`, then we must also include `V1`, `V2`, `V3`, `V1:V2`, `V1:V3`, and `V2:V3`. An easy way to ensure that a model is hierarchical is to avoid ':' and only use '*'. Using '*' allows us to represent the model by just its highest-order terms, because the lower-order terms will then be included automatically.

Many hierarchical models can be described in terms of independence and conditional independence. For example, with three variables, we may have:

- *complete independence*, as in `~ V1 + V2 + V3`.
- *two variables independent of a third*, as in `~ V1*V2 + V3`, which allows `V1` and `V2` to be related, but requires them to be jointly independent of `V3`. Similarly for `~ V1*V3 + V2` and `~ V2*V3 + V1`.
- *two variables conditionally independent given a third*, such as `~ V1*V2 + V1*V3`, which means that `V2` and `V3` are unrelated at any fixed value of `V1`. Similarly for `~ V1*V2 + V2*V3` and `~ V1*V3 + V2*V3`.
- *the saturated model*, `~ V1*V2*V3`, which allows arbitrary relationships among the three variables.

The only remaining hierarchical model with three variables is `~ V1*V2 + V1*V3 + V2*V3`, the model of *homogeneous association*. This model requires the odds ratios between any two variables, when conditioned on the third, to be constant across

levels of the third. For example, the odds ratios between `V1` and `V2` within any slice of the table that holds `V3` constant are identical across all such slices.

Multivariate models are sometimes represented graphically, with nodes corresponding to variables and edges indicating relationships among them. The absence of edges conveys assumptions of independence and conditional independence (Lauritzen, 1996; Whittaker, 2009). Some log-linear models, e.g., the model of homogeneous association, do not have a graphical representation. To be graphical, a model that has all two-way associations among a group of variables must include all higher-way associations as well. For example, if a graphical model has `V1*V2`, `V1*V3`, and `V2*V3`, then it must also have `V1*V2*V3`.

Log-linear models are also closely related to logistic regression. A logistic model for predicting a categorical outcome from categorical covariates can be fit as a joint log-linear model for the outcome and covariates, provided that the log-linear model includes all possible associations among the covariates. For example, the log-linear model represented by $\sim$ `V1*V2*V3 + V1*V4 + V2*V3*V4` implies the logistic model `V4` $\sim$ `V1 + V2*V3`. Fitting the log-linear model can be computationally more demanding, because it requires estimating nuisance parameters (the associations among the covariates) which do not appear in the logistic version. For more discussion on the relationships between log-linear and logistic models, see Christensen (2006).

3.4 Fitting log-linear models with conventional R functions

3.4.1 Conditional odds ratios from the U.C. Berkeley admissions data

Returning to the dataset `UCBAdmissions` from Section 2, we now show how to fit log-linear models using various R functions. Because these data have been analyzed *ad nauseum*, we use them for demonstration purposes but will not belabor their interpretation. To begin, we display the 2×2 marginal table with `Gender` as the row and `Admit` as the column and compute the marginal odds ratio.

```
> # display observed marginal table and odds ratio
> marg <- apply( UCBAdmissions, c(2,1), sum )
> marg
```

```
        Admit
Gender   Admitted Rejected
  Male       1198     1493
  Female      557     1278
```

```
> marg[1,1] * marg[2,2] / ( marg[2,1] * marg[1,2] )
```

```
[1] 1.84108
```

The value of 1.84 implies that, on the odds scale, male applicants were 84% more likely than female applicants to gain admission to graduate school. Within levels of Dept, however, the conditional odds ratios tell a different story.

```
> # display odds ratios for each department
> UCBAdmissions[1,1,] * UCBAdmissions[2,2,] /
+    ( UCBAdmissions[1,2,] * UCBAdmissions[2,1,] )

       A        B        C        D        E        F
0.3492120 0.8025007 1.1330596 0.9212838 1.2216312 0.8278727
```

Male applicants had lower odds of admission than females in departments A, B, D, and F, and higher odds of admission than females in departments C and E. We will examine the statistical significance of these conditional effects by comparing the fits of three log-linear models,

$$\begin{aligned} \texttt{M0:} &\sim \texttt{Dept*Gender + Dept*Admit}, \\ \texttt{M1:} &\sim \texttt{Dept*Gender + Dept*Admit + Gender*Admit}, \\ \texttt{M2:} &\sim \texttt{Dept*Gender*Admit}. \end{aligned}$$

Model M0 fixes the conditional odds ratio in every department at one; M1 forces the conditional odds ratios to be equal, but does not constrain their common value; and M2 allows the conditional odds ratios to vary. The models are nested in the sense that $\texttt{M0} \subset \texttt{M1} \subset \texttt{M2}$, so we can compare them by likelihood-ratio (LR) chi-squared tests based on their deviance statistics. The test of M0 against M1 has one degree of freedom, because the models differ by one parameter, and the test of M1 against M2 has five degrees of freedom, because they differ by five parameters, The difference in fit between M0 and M2 can be partitioned into six independent statistics corresponding to tests for independence in the 2×2 tables for each of the six departments.

3.4.2 Examples with `glm`

Using the Poisson trick, we regress the observed frequencies on the factors using glm with the argument family=poisson(), which by default applies a log link. This approach requires reshaping the contingency table into a grouped data frame.

```
> dF <- as.data.frame(UCBAdmissions)
> M0 <- glm( Freq ~ Dept*Gender + Dept*Admit, family=poisson(), data=dF )
> M1 <- glm( Freq ~ Dept*Gender + Dept*Admit + Gender*Admit,
+    family=poisson(), data=dF )
> M2 <- glm( Freq ~ Dept*Gender*Admit, family=poisson(), data=dF )
```

Instead of trying to interpret the coefficients of each model, we compute parameters of interest from the fitted values $\hat{\boldsymbol{\mu}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$. For each model, we put $\hat{\boldsymbol{\mu}}$ into the data frame, then reshape the $\hat{\boldsymbol{\mu}}$ vector into a three-dimensional array with the factors as its dimensions.

```
> dF$muHat0 <- predict(M0, type="response")
> dF$muHat1 <- predict(M1, type="response")
> dF$muHat2 <- predict(M2, type="response")
> fit0 <- xtabs( muHat0 ~ Admit + Gender + Dept, data=dF )
> fit1 <- xtabs( muHat1 ~ Admit + Gender + Dept, data=dF )
> fit2 <- xtabs( muHat2 ~ Admit + Gender + Dept, data=dF )
```

Examining the fitted conditional odds ratios within departments, we see that each of the models does what we expect.

```
> # under M0, the fitted conditional OR's should be 1.0:
> fit0[1,1,] * fit0[2,2,] / ( fit0[1,2,] * fit0[2,1,] )

A B C D E F
1 1 1 1 1 1

> # under M1, the fitted conditional OR's should be equal:
> fit1[1,1,] * fit1[2,2,] / ( fit1[1,2,] * fit1[2,1,] )

      A       B       C       D       E       F
0.904955 0.904955 0.904955 0.904955 0.904955 0.904955

> # under M2, the fitted conditional OR's should vary, and they
> # should agree with corresponding OR's based on the observed
> # frequencies, because M2 is saturated:
> fit2[1,1,] * fit2[2,2,] / ( fit2[1,2,] * fit2[2,1,] )

       A        B        C        D        E        F
0.3492120 0.8025007 1.1330596 0.9212838 1.2216312 0.8278727
```

Using `anova`, we display the deviance statistics for comparing `M0`, `M1`, and `M2`.

```
> anova(M0,M1,M2)

Analysis of Deviance Table

Model 1: Freq ~ Dept * Gender + Dept * Admit
Model 2: Freq ~ Dept * Gender + Dept * Admit + Gender * Admit
Model 3: Freq ~ Dept * Gender * Admit
  Resid. Df Resid. Dev Df Deviance
1         6     21.735
2         5     20.204  1   1.5312
3         0      0.000  5  20.2043
```

The deviance statistic for testing `M0` against `M1` is 1.5312, which gives a p-value of $p = P(\chi^2_1 \geq 1.5312) = 0.216$; the common conditional odds ratio estimated across departments is not significantly different from 1.0. The deviance for testing `M1`

against M2 is 20.2043, which gives a p-value of $p = P(\chi^2_5 \geq 20.2043) = 0.001$, so at least one of the conditional odds ratios is significantly different from the others. To see what is happening within the departments, we perform a separate test for independence for each department.

```
> # make a list of 6 data frames, one per department
> list2x2 <- as.list(1:6)
> for( j in 1:6 ) list2x2[[j]] <- subset(dF, Dept==levels(dF$Dept)[j]  )
> # function for computing deviance for LR test of independence
> # within a department
> myFunc <- function( dF ) {
+    M <- glm( Freq ~ Gender + Admit, family=poisson(), data=dF )
+    deviance(M)
+ }
> # apply LR test to each department, returning a vector of deviances
> dev <- sapply( list2x2, myFunc )
> dev

[1] 19.0540099  0.2586429  0.7509844  0.2978665  0.9903864  0.3836167

> sum(dev)

[1] 21.73551
```

Notice that the sum of the deviance test statistics across departments exactly matches the overall statistic for testing M0 against M2. The only department with a conditional odds ratio that is significantly different from 1.0 is "A".

3.4.3 Examples with `loglin`

The function `loglin` fits log-linear models using IPF. The data are supplied as a contingency table, and the model is specified not by a formula but by a list of integer vectors denoting highest-order effects that the model is fitting. In the three-dimensional table `UCBAdmissions`, the dimensions are

$$\begin{aligned} 1 &= \texttt{Admit}, \\ 2 &= \texttt{Gender}, \\ 3 &= \texttt{Dept}. \end{aligned}$$

In the integer notation of `loglin`, the model

$$\sim \texttt{Dept*Gender + Dept*Admit}$$

is expressed as `list( c(3,2), c(3,1) )`; the model

$$\sim \texttt{Dept*Gender + Dept*Admit + Gender*Admit}$$

becomes `list( c(3,2), c(3,1), c(2,1) )`; and the model

$$\sim \texttt{Dept*Gender*Admit}$$

becomes `list( c(3,2,1) )`. The `loglin` function returns a list of results, and if the argument `fit=TRUE` is supplied, that list will include a table of fitted values $\hat{\boldsymbol{\mu}}$.

```
> # fit M0, M1, and M2 using loglin
> M0 <- loglin( UCBAdmissions, margin=list( c(3,2), c(3,1) ), fit=TRUE )

2 iterations: deviation 5.684342e-14

> M1 <- loglin( UCBAdmissions, margin=list( c(3,2), c(3,1), c(2,1) ), fit=TRUE )

7 iterations: deviation 0.04308377

> M2 <- loglin( UCBAdmissions, margin=list( c(3,2,1)), fit=TRUE )

2 iterations: deviation 5.684342e-14
```

For models `M0` and `M2`, the IPF procedure stopped at iteration 2; apart from rounding error, the final solution was achieved at the end of the first cycle, because these models are decomposable. Before proceeding. let's make sure that the fitted values from `loglin` match the results we obtained from `glm`.

```
> max( abs( fit0 - M0$fit ) )

[1] 1.128669e-09

> max( abs( fit1 - M1$fit ) )

[1] 0.01320492

> max( abs( fit2 - M2$fit ) )

[1] 1.818989e-12
```

For `M0` and `M2`, the fitted values from `loglin` and `glm` are exceedingly close. For `M1`, the largest discrepancy is about 0.0132, but this can be made smaller by tightening the IPF convergence criterion through the argument `eps`.

```
> M1 <- loglin( UCBAdmissions, margin=list( c(3,2), c(3,1), c(2,1) ),
+    fit=TRUE, eps=1e-06 )

18 iterations: deviation 9.673679e-07

> max( abs( fit1 - M1$fit ) )

[1] 2.964787e-07
```

The likelihood-ratio fit statistics from `loglin` match the deviance values from `glm`.

```
> M0$lrt
[1] 21.73551
> M1$lrt
[1] 20.20428
> M2$lrt
[1] 0
```

The IPF procedure used by `loglin` can also be called indirectly, through the `loglm` function in the package `MASS` (Venables and Ripley, 2013). With `loglm`, the data may be provided either as a contingency table or as a grouped data frame, and the model may be specified with a formula. For details, see `help(loglm, package=MASS)`.

3.5 Log-linear modeling with `cvam`

3.5.1 Fitting a model with complete data

In the `cvam` package, log-linear models are fit by the function `cvam`. For conventional log-linear modeling with no missing or coarsened values, `cvam` works like `glm`, but with a few notable differences. Like `glm`, the model is specified by a formula, and the data are supplied through a data frame. Unlike `glm`, the formula is one-sided, and the variable holding the frequencies is specified not in the formula but as an argument named `freq`. For example, here is how we would fit and compare our three models for the U.C. Berkeley admissions data.

```
> library(cvam)
> dF <- as.data.frame(UCBAdmissions)
> M0 <- cvam( ~ Dept*Gender + Dept*Admit, data=dF, freq=Freq )
> M1 <- cvam( ~ Dept*Gender + Dept*Admit + Gender*Admit, data=dF, freq=Freq )
> M2 <- cvam( ~ Dept*Gender*Admit, data=dF, freq=Freq )
> anova(M0,M1,M2)
Model 1: ~ Dept * Gender + Dept * Admit
Model 2: ~ Dept * Gender + Dept * Admit + Gender * Admit
Model 3: ~ Dept * Gender * Admit
  resid.df -2*loglik df  change
1        6    -41005
2        5    -41006  1  1.5312
3        0    -41026  5 20.2043
```

The results are equivalent to those from `glm`. One minor difference is that with `cvam`, the `anova` table reports -2 times the loglikelihood value achieved under each model, whereas with `glm`, it reports a deviance statistic. The two measures differ by a constant that drops out whenever two models are compared.

3.5.2 Extracting information from a `cvam` object

The result from a call to `cvam` is a cvam object, a special kind of list that holds parameter estimates and other information from the model fit. To access this information, we recommend that you use `summary` or the various `get` functions supplied with the package. For example, the function `get.coef` returns the the vector of estimated coefficients $\hat{\beta}$. Calling `get.coef` with the argument `withSE=TRUE` produces a data frame with estimated coefficients, standard errors, t-statistics and p-values.

```
> get.coef(M0, withSE=TRUE)

                     coef         SE  zstat   pval
(Intercept)    4.805669565 0.02598938 184.91 0.0000
Dept1          0.156547492 0.04985439   3.14 0.0017
Dept2         -0.761804591 0.08789640  -8.67 0.0000
Dept3          0.539038941 0.03924728  13.73 0.0000
Dept4          0.426921589 0.04020043  10.62 0.0000
Dept5         -0.027346137 0.04843304  -0.56 0.5723
Gender1        0.334605321 0.02289450  14.62 0.0000
Admit1        -0.325781804 0.01950248 -16.70 0.0000
Dept1:Gender1  0.682020758 0.04763810  14.32 0.0000
Dept2:Gender1  1.219925158 0.08653579  14.10 0.0000
Dept3:Gender1 -0.635289930 0.03630476 -17.50 0.0000
Dept4:Gender1 -0.281525223 0.03699031  -7.61 0.0000
Dept5:Gender1 -0.695373413 0.04267138 -16.30 0.0000
Dept1:Admit1   0.622511787 0.03405394  18.28 0.0000
Dept2:Admit1   0.597214293 0.04007482  14.90 0.0000
Dept3:Admit1   0.017937243 0.03431599   0.52 0.6012
Dept4:Admit1  -0.006653238 0.03631253  -0.18 0.8546
Dept5:Admit1  -0.218968500 0.04353759  -5.03 0.0000
```

The function `get.fitted` extracts the fitted values in the form of cell probabilities, cell means, or log-cell means. By default, it returns a data frame with one row per cell. The variables in this data frame include all of the factors in the model, a variable named `freq` containing the frequencies, and a variable named `fit` containing the fitted values.

```
> # display the fitted means for the first few cells
> head( get.fitted(M0, type="mean" ) )

  Dept Gender    Admit freq       fit
1    A   Male Admitted  512 531.43087
2    B   Male Admitted  353 354.18803
3    C   Male Admitted  120 113.99782
4    D   Male Admitted  138 141.63258
5    E   Male Admitted   53  48.07705
6    F   Male Admitted   22  24.03081
```

The reason why `get.fitted` returns a data frame by default rather than a vector of fitted values is that, unlike `glm`, *the ordering of the cells in the model is determined by the internal workings of `cvam`, not by the rows of the data frame supplied to `cvam` through its `data` argument*. This is an extremely important point, so let us say it

again in a slightly different way: *With cvam, the number of fitted values and the order in which they appear may not, and in most cases do not, correspond to the rows of the user-supplied data.* With `cvam`, the data frame supplied for model-fitting does not need to have one row per cell. The `cvam` function automatically aggregates the data, identifying the unique response patterns (i.e., the unique combinations of observed values for all factors in the model) and adds up the frequencies within these response patterns. It identifies cells that have no observations in them and assigns those cells frequencies of zero even if they are absent from the data frame. The `cvam` function also accepts microdata. If the argument `freq` is not given, then `cvam` assumes that the data frame contains microdata, and it assigns each row a frequency of one. For the Berkeley graduate admissions data, we get the same results whether we use the aggregated frequencies in `UCBAdmissions` or the microdata in `microUCBAdmissions`.

```
> # refit M0 with microdata to see that results are the same
> M0 <- cvam( ~ Dept*Gender + Dept*Admit, data=microUCBAdmissions )
> get.coef(M0, withSE=TRUE)
```

```
                     coef         SE  zstat   pval
(Intercept)    4.805669565 0.02598938 184.91 0.0000
Dept1          0.156547492 0.04985439   3.14 0.0017
Dept2         -0.761804591 0.08789640  -8.67 0.0000
Dept3          0.539038941 0.03924728  13.73 0.0000
Dept4          0.426921589 0.04020043  10.62 0.0000
Dept5         -0.027346137 0.04843304  -0.56 0.5723
Gender1        0.334605321 0.02289450  14.62 0.0000
Admit1        -0.325781804 0.01950248 -16.70 0.0000
Dept1:Gender1  0.682020758 0.04763810  14.32 0.0000
Dept2:Gender1  1.219925158 0.08653579  14.10 0.0000
Dept3:Gender1 -0.635289930 0.03630476 -17.50 0.0000
Dept4:Gender1 -0.281525223 0.03699031  -7.61 0.0000
Dept5:Gender1 -0.695373413 0.04267138 -16.30 0.0000
Dept1:Admit1   0.622511787 0.03405394  18.28 0.0000
Dept2:Admit1   0.597214293 0.04007482  14.90 0.0000
Dept3:Admit1   0.017937243 0.03431599   0.52 0.6012
Dept4:Admit1  -0.006653238 0.03631253  -0.18 0.8546
Dept5:Admit1  -0.218968500 0.04353759  -5.03 0.0000
```

3.5.3 Getting Pearson residuals

The data frame returned by `get.fitted` is useful for model diagnostics. In the example below, we compute the Pearson residuals

$$\hat{r}_{\boldsymbol{v}} = \frac{f_{\boldsymbol{v}} - \hat{\mu}_{\boldsymbol{v}}}{\sqrt{\hat{\mu}_{\boldsymbol{v}}}} \tag{6}$$

for all cells, plot them against the cell index, and use the `identify` function to interactively label the outliers. Labels for the identified points are created with the ':' operator, which combines multiple factors into a single factor.

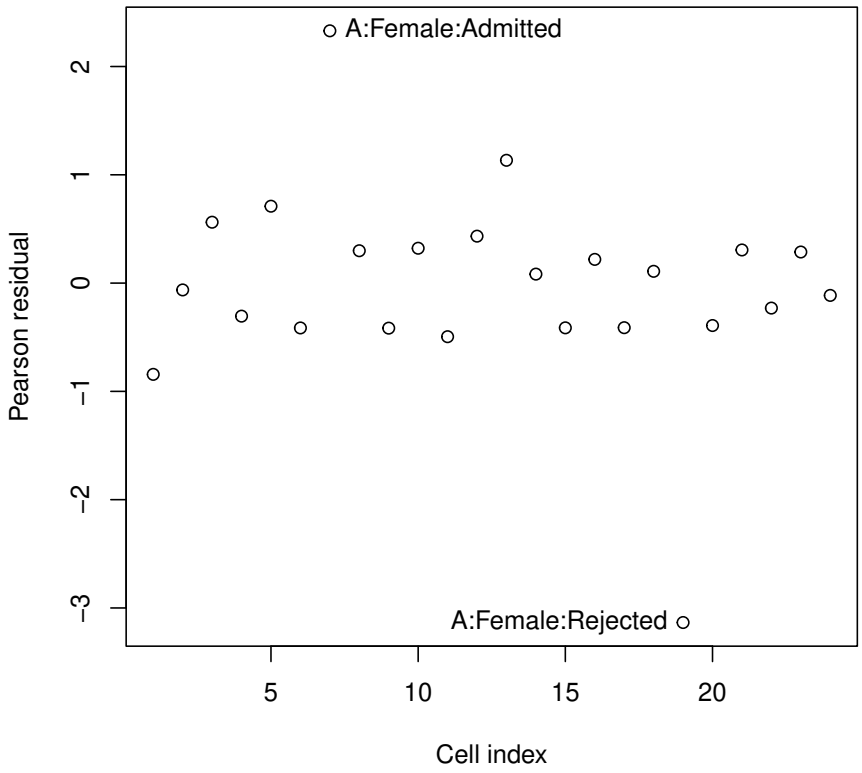

Figure 1: Pearson residuals from model M_0 plotted against cell number, with two outliers identified.

```
> fit0 <- get.fitted(M0, type="mean")
> pearson <- ( fit0$freq - fit0$fit ) / sqrt( fit0$fit )
> labs <- as.character( fit0$Dept : fit0$Gender : fit0$Admit )
> plot( 1:NROW(fit0), pearson,
+    xlab="Cell index", ylab="Pearson residual" )
> identify( 1:NROW(fit0), pearson, labels=labs )
```

The resulting plot, which is shown in Figure 1, clearly reveals where this model does not fit; `Gender` and `Admit` are conditionally related in Department `"A"`.

3.5.4 Saturated and conditional models

By default, `cvam` fits a complete-data model by the same procedure as `glm`, using NR to maximize the surrogate Poisson loglikelihood. If the model is saturated, an alternative procedure can be invoked by calling `cvam` with the argument

`saturated=TRUE`. Under this option, the model matrix $\boldsymbol{X}$ is never constructed, so the coefficients $\boldsymbol{\beta}$ are undefined, and fitted values are obtained from the frequencies as $\hat{\boldsymbol{\mu}} = \hat{\boldsymbol{f}}$ and $\hat{\boldsymbol{\pi}} = N^{-1}\hat{\boldsymbol{f}}$. An advantage of using `saturated=TRUE` is that the computations are faster and require less memory, especially when the number of cells in the table is large. A disadvantage is that standard errors will not be reported. If you need standard errors, you can fit the same model with `saturated=FALSE`.

Fitting a saturated model allows you to compute a deviance statistic. Deviance is defined as -2 times the difference between the loglikelihood achieved under the model of interest and the loglikelihood achieved under the saturated model. The loglikelihood for any model can be obtained with `get.loglik`.

```
> # compute the deviance for model M0
> M0 <- cvam( ~ Dept*Gender + Dept*Admit, data=dF, freq=Freq )
> M2 <- cvam( ~ Dept*Gender*Admit, data=dF, freq=Freq, saturated=TRUE )
> dev.M0 <- -2 * ( get.loglik(M0) - get.loglik(M2) )
> dev.M0
```

```
[1] 21.73551
```

With `cvam`, you can also fit log-linear models that regard some variables as fixed. With the Berkeley admissions data, it is natural to examine the conditional distribution of `Admit` given `Dept` and `Gender`. The model of homogeneous association

$$\sim \texttt{Dept*Gender + Dept*Admit + Gender*Admit}$$

implies a logistic model for `Admit` with main effects for `Dept` and `Gender`. To fit a conditional model, use the same model formula as for the joint model, but include the symbol '|' followed by the variables to be conditioned on, separating them with '+'.

```
> # fit M1 as a conditional model
> M1 <- cvam( ~ Dept*Gender + Dept*Admit + Gender*Admit | Dept + Gender,
+    data=dF, freq=Freq )
```

Coefficients and standard errors from a conditional model are identical to those from the joint model, because the parameters are estimated under the same surrogate Poisson regression. The only major difference is that, for a conditional model, fitted probabilities from `get.fitted` will be scaled to sum to one within every combination of levels for the variables treated as fixed.

```
> # show the first few fitted probabilities
> head( get.fitted(M1, type="prob") )
```

```
     Admit Dept Gender freq       fit
1 Admitted    A   Male  512 0.6415393
2 Rejected    A   Male  313 0.3584607
```

```
3 Admitted    B   Male  353 0.6314991
4 Rejected    B   Male  207 0.3685009
5 Admitted    C   Male  120 0.3361393
6 Rejected    C   Male  205 0.6638607
```

Conditional models are consistent with product-multinomial sampling, which is described in Appendix B. Under a product-multinomial scheme, the surrogate Poisson regression gives correct results only if the requested model (the part of the formula on the left-hand side of '|') includes all possible associations among the variables that are held fixed (Venables and Ripley, 2013). If the requested model omits any of these associations, the call to `cvam` produces an error.

4 Examples with missing and coarsened data

4.1 A 2 × 2 table with missing data

Thus far, we have shown that `cvam` gives results equivalent to those from `glm` and `loglin` when the data are are complete. The main advantage in using `cvam` is that it allows us to make full use of all the available data when some information is missing.

The `crime` dataset, previously analyzed by Kadane (1985) and Schafer (1997), came from the National Crime Survey conducted by the U.S. Census Bureau. Occupants of housing units were interviewed to determine whether they had been victimized by crime in the preceding six-month period. Six months later, the same units were visited again to determine whether the occupants had been victimized during the intervening months. Missing values for various reasons occurred at both occasions. Variables `V1` and `V2` are factors indicating whether victimization was reported at occasion 1 and occasion 2, and `n` is the frequency.

```
> data(crime)   # load the crime dataset distributed with cvam
> crime

    V1   V2   n
1   no   no 392
2  yes   no  76
3 <NA>   no  31
4   no  yes  55
5  yes  yes  38
6 <NA>  yes   7
7   no <NA>  33
8  yes <NA>   9
9 <NA> <NA> 115

> sum(crime$n)

[1] 756
```

If we try to explore the joint distribution of V1 and V2, we encounter a problem. Conventional methods expect that each observation has been unambiguously assigned to a single cell of the 2×2 table indexed by

$$\boldsymbol{v} \in \{ (1,1), (2,1), (1,2), (2,2) \}.$$

Conceptually, each household represented in this dataset does have a true victimization status for each time period and thus belongs to a single cell. The difficulty is that cell membership is fully known only for the 561 households with non-missing values for both time periods. For the 31 households with V1=NA and V2="no", we know that a portion fall into cell $(1,1)$ and a portion fall into cell $(2,1)$, but we do not know what those portions are. Similarly, the 33 households with V1="no" and V2=NA are divided between cell $(1,1)$ and $(2,1)$, but we do not know how to divide them. It may be reasonable to discard the 115 households with V1=NA and V2=NA, as they appear to be noninformative. But the other households do provide partial information that may be useful for assessing the joint distribution of V1 and V2.

If we had to analyze this dataset with conventional software, we would be forced to choose between

- analyzing the 2×2 table with only the 561 complete cases, producing answers that are inefficient and possibly biased, or
- using all 756 cases, but treating NA as an additional level, leading to a 3×3 table that is difficult to interpret.

With cvam, however, we just specify a model that we would use if no data were missing, supply the incomplete data, and fit the model.

```
> # fit the model of independence
> M0 <- cvam( ~ V1 + V2, freq=n, data=crime )
> # fit the model of non-independence
> M1 <- cvam( ~ V1 * V2, freq=n, data=crime )
> # compare them
> anova(M0,M1, pval=TRUE)

Model 1: ~ V1 + V2
Model 2: ~ V1 * V2
  resid.df -2*loglik df change pval
1        1   -5853.2
2        0   -5878.6  1 25.382    0
```

The large improvement of fit when moving from the first model to the second provides strong evidence that V1 and V2 are related.

To understand what cvam did, let's examine the results from the independence model using summary.

```
> summary(M0)

~ V1 + V2

Prior:
      Flattening frequency = 0
Total nuggets + flattening = 0
              Ridge factor = 0
          Intensity factor = 1

Sample size:
              total N in supplied data = 756
N from supplied data used in model fit = 641
           prior effective sample size =   0

Degrees of freedom:
    patterns of coarsened data = 9
  cells in complete-data table = 4
cells without latent variables = 4
         structural zero cells = 0
   parameters in Poisson model = 3
                            df = 1

Starting values:
default, center
jitter SD = 0.000000

EM algorithm:
Converged at iteration 9
Gradient length = 0.000000

  Final logP = 2926.608
Final loglik = 2926.608

Estimates from EM, with Hessian-based SEs
              coef      SE zstat pval
(Intercept) 4.5677 0.06154 74.23    0
V11         0.6808 0.05053 13.47    0
V21         0.8037 0.05478 14.67    0
```

By default, `cvam` computes ML estimates by an EM algorithm. This procedure has NR embedded within it and reduces to NR when there are no missing or coarsened values. The standard errors displayed in the table of coefficients account for the uncertainty due to missing or coarsened values. The EM algorithm and standard-error computations are described in Appendix G and Appendix I.

4.2 Fitted values, predicted true frequencies, and residuals

Another interesting result is seen in the data frame that contains the fitted values.

```
> dF <- get.fitted(M0, type="mean")
> dF

   V1  V2      freq      fit
1  no  no 520.42720 501.3251
2 yes  no 109.36244 128.4646
```

```
3  no yes  81.36384 100.4660
4 yes yes  44.84651  25.7444
```

The variable `fit` holds the estimated cell means $\hat{\boldsymbol{\mu}}$. Because this model assumes independence, the odds ratio among these fitted values is exactly one.

```
> ( dF$fit[1] * dF$fit[4] ) / ( dF$fit[2] * dF$fit[3] )
[1] 1
```

The variable `freq` holds the predicted true frequencies, a vector that we will call $\hat{\boldsymbol{f}}$, and the odds ratio among these values is not one.

```
> ( dF$freq[1] * dF$freq[4] ) / ( dF$freq[2] * dF$freq[3] )
[1] 2.622944
```

The vector $\hat{\boldsymbol{f}}$ is called the *predicted true frequencies*. It represents our best guess as to how the 756 cases in the `crime` dataset should be apportioned to the cells of the 2×2 table, assuming that the model is correct. If there were no missing values, then the cell membership of each case would be known; in that case, no prediction would be needed, and $\hat{\boldsymbol{f}}$ would be identical to $\boldsymbol{f}$ regardless of the model. When some of the data are missing, $\hat{\boldsymbol{f}}$ combines the observed data with parameter estimates from EM to predict the unknown elements of $\boldsymbol{f}$. From the results of the LR test, we know that this model does not fit. The apportionment of incomplete cases in $\hat{\boldsymbol{f}}$ is based on the implausible assumption of independence, so residuals that compare $\hat{\boldsymbol{f}}$ to $\hat{\boldsymbol{u}}$ are likely to understate the lack of fit.

A better strategy for assessing fit is to use quasi-Pearson residuals, which we define as

$$\tilde{r}_{\boldsymbol{v}} = \frac{\hat{f}_{\boldsymbol{v}}^{(sat)} - \hat{\mu}_{\boldsymbol{v}}}{\sqrt{\hat{\mu}_{\boldsymbol{v}}}}, \tag{7}$$

where $\hat{f}_{\boldsymbol{v}}^{(sat)}$ is the predicted true frequency under a saturated model, and $\hat{\mu}_{\boldsymbol{v}}$ is the estimated cell mean under the current model. With no missing data, the quasi-Pearson residuals are equal to the Pearson residuals from Equation (6). With missing data, the quasi-Pearson residuals behave differently from Pearson residuals, with distributional properties that depend on the rates of missingness and the unmodeled missingness mechanism. Nevertheless, they can help to reveal areas of the table where a model does not fit, and they reduce to zero for all cells when the model is saturated.

To compute quasi-Pearson residuals, you need to fit two models: the model of interest, which provides the fitted cell means, and the saturated model, which provides the predicted true frequencies. Keep in mind that these two vectors may order

their observations differently. As we emphasized in Section 3.5.2, the ordering of cells in a `cvam` model is defined by `cvam`, and is not directly controlled by the user. More precisely, the cell ordering is determined by the order in which variables appear in the model formula. To check whether $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{f}}^{(sat)}$ have the same ordering, you should examine the frames returned by `get.fitted` for both models and make sure that the factors appear in the same columns. If not, you will need to revise one of the model formulas so that the variables appear in the same order as in the other formula, and then re-fit the model whose formula was revised.

```
> # examine the frames from get.fitted for M0 and M1
> # to make sure that they use the same cell ordering
> get.fitted(M0)

   V1  V2      freq       fit
1  no  no 520.42720 0.66312843
2 yes  no 109.36244 0.16992666
3  no yes  81.36384 0.13289147
4 yes yes  44.84651 0.03405344

> get.fitted(M1)

   V1  V2      freq       fit
1  no  no 527.02525 0.69712335
2 yes  no 102.65197 0.13578303
3  no yes  74.56461 0.09863044
4 yes yes  51.75817 0.06846318

> # compute the quasi-Pearson residuals
> muHat <- get.fitted(M0, type="mean")$fit
> fHatSat <- get.fitted(M1, type="mean")$freq
> quasiPearson <- ( fHatSat - muHat ) / sqrt( muHat )
> quasiPearson

[1]  1.147826 -2.277402 -2.584121  5.126983
```

Examining these residuals, we see that the independence model underpredicts membership in cells $(1,1)$ and $(2,2)$ and overpredicts membership in $(1,2)$ and $(2,1)$.

4.3 One variable with coarsened values

In the companion vignette *Understanding Coarsened Factors in `cvam`*, we described our extension to R's factor mechanism to store categorical variables with coarsened values. Using the `abortion2000` dataset distributed with `cvam`, we created a coarsened factor based on race and Hispanic origin. The code for creating this factor is reproduced below.

```
> data(abortion2000)
> CenRace <- addNA(abortion2000$CenRace)
> Hisp <- addNA(abortion2000$Hisp)
```

```
> RH <- Hisp:CenRace
> RH <- droplevels(RH)
> levels(RH) <- list(
+    nonHispWhite = "nonHisp:White",
+    nonHispBlack = "nonHisp:Black",
+    nonHispOther = "nonHisp:Other",
+    Hisp = c("Hisp:White", "Hisp:Black", "Hisp:Hisp", "Hisp:Other", "Hisp:NA"),
+    nonHispNA = "nonHisp:NA",
+    NAWhite = "NA:White" )
> RH  <- coarsened( RH, levelsList = list(
+    nonHispNA = c("nonHispWhite", "nonHispBlack", "nonHispOther"),
+    NAWhite = c("nonHispWhite", "Hisp" ) ) )
> summary(RH)

nonHispWhite nonHispBlack nonHispOther         Hisp    nonHispNA      NAWhite
        1042          198           44          212         1320            1
        <NA>
           0
```

This coarsened factor has four base levels, which represent categories with no missing information.

```
> baseLevels(RH)

[1] "nonHispWhite" "nonHispBlack" "nonHispOther" "Hisp"
```

The other three levels, which we call coarse levels, represent groupings of the base levels. Relationships between the coarse and base levels are shown by the mapping matrix.

```
> coarseLevels(RH)

[1] "nonHispNA" "NAWhite"   NA

> mapping(RH)

          nonHispWhite nonHispBlack nonHispOther Hisp
nonHispNA            1            1           1    0
NAWhite              1            0           0    1
<NA>                 1            1           1    1
```

To estimate the proportions of the population that fall into the four base levels, we use the `cvam` function to fit a one-variable model.

```
> result <- cvam( ~ RH )
> summary(result, showCoef=FALSE)

~ RH

Prior:
      Flattening frequency = 0
Total nuggets + flattening = 0
```

```
             Ridge factor = 0
          Intensity factor = 1

Sample size:
              total N in supplied data = 2817
N from supplied data used in model fit = 2817
           prior effective sample size =    0

Degrees of freedom:
    patterns of coarsened data = 6
  cells in complete-data table = 4
cells without latent variables = 4
         structural zero cells = 0
   parameters in Poisson model = 4
                            df = 0

Starting values:
default, center
jitter SD = 0.000000

EM algorithm:
Converged at iteration 31
Gradient length = 0.000001

  Final logP = 18071.1
Final loglik = 18071.1
```

```
> # display the fitted proportions
> get.fitted(result)
```

```
           RH       freq        fit
1 nonHispWhite 2114.29975 0.75055014
2 nonHispBlack  401.40743 0.14249465
3 nonHispOther   89.20165 0.03166548
4         Hisp  212.09117 0.07528973
```

4.4 Mixing complete, incomplete, and coarsened variables

A variable in a `cvam` modeling formula may be an ordinary factor with complete data, an ordinary factor with missing values, or a coarsened factor created by the function `coarsened`. It may even be a latent class, a categorical variable whose values are entirely missing. Latent-class models will be discussed in Section 8.

The `abortion2000` dataset is a frame of microdata with 2,817 records extracted from the General Social Survey (GSS) (Smith et al., 2019). For illustration, we will fit a model with four variables from this dataset.

- `Sex`, a factor with levels `"Female"` and `"Male"` and no missing values;
- `RH`, the coarsened factor that we created from `CenRace` and `Hisp`;
- `PolViews`, a factor that represents the respondent's self-described political orientation, with levels `"Con"` (Conservative), `"Mod"` (Moderate), and `"Lib"` (Liberal), and 173 missing values; and

- `AbAny`, whether the respondent thinks it should be possible for a woman to obtain a legal abortion if the woman wants it for any reason, with levels `"Yes"`, `"No"`, `"DK"` (don't know) and 962 missing values.

To begin, we place these four variables in a data frame of their own. This step is not required; we do this only to keep our visual displays tidy.

```
> # copy the four variables into a data frame
> dF <- data.frame( Sex = abortion2000$Sex, RH = RH,
+    PolViews = abortion2000$PolViews, AbAny = abortion2000$AbAny )
> # display the first few rows
> head(dF)
```

```
     Sex           RH PolViews AbAny
1   Male    nonHispNA      Con    No
2 Female         Hisp      Con    No
3 Female    nonHispNA      Con  <NA>
4 Female nonHispWhite      Con  <NA>
5 Female nonHispWhite      Lib  <NA>
6 Female nonHispWhite      Lib   Yes
```

Next, we create a formula to specify the model that we want to fit. Thinking of `AbAny` as a response and the other three variables as potential predictors, one model worth considering is

```
> myFormula <- ~ Sex*RH*PolViews + AbAny*Sex + AbAny*RH + AbAny*PolViews
```

which implies a multinomial logistic regression for `AbAny` with main effects for `Sex`, `RH`, and `PolViews`. If we tried to fit this logistic model directly, any record that has a missing or coarsened value for any of the three predictors would be dropped. By embedding this logistic regression into a log-linear model and calling `cvam`, we can fit the joint model to all the records.

Let's fit this model and compare it to the saturated model in two different ways: with an LR test, and by Akaike's information criterion (AIC). The latter is obtained by calling `anova` with the argument `method="AIC"`.

```
> myMod <- cvam( myFormula, data=dF )
> satMod <- cvam( ~ Sex*RH*PolViews*AbAny, data=dF, saturated=TRUE )
```

```
Note: Estimate at or near boundary
```

```
> anova( myMod, satMod, pval=TRUE )
```

```
Model 1: ~ Sex * RH * PolViews + AbAny * Sex + AbAny * RH + AbAny * PolViews
Model 2: ~ Sex * RH * PolViews * AbAny
  resid.df -2*loglik df change   pval
1       34    -23618
2        0    -23661 34 43.091 0.1364
```

```
> anova( myMod, satMod, method="AIC" )
```

```
Model 1: ~ Sex * RH * PolViews + AbAny * Sex + AbAny * RH + AbAny * PolViews
Model 2: ~ Sex * RH * PolViews * AbAny
  resid.df -2*loglik    AIC rank
1       34    -23618 -23542    1
2        0    -23661 -23517    2
```

```
> # compute and summarize the fitted values
> muHat <- get.fitted(myMod, type="mean")$fit
> summary( muHat )
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
  0.8271   3.8323  11.8807  39.1250  27.9982 267.2953
```

The LR comparison reveals some lack of fit. The p-value is not especially accurate, because many of the fitted values are small. AIC favors the smaller model. We can also test the significance of each of the three predictors by omitting their associations with AbAny, one a time, and performing the LR tests against the current model.

```
> noSex <- cvam( ~ Sex*RH*PolViews + AbAny*RH + AbAny*PolViews, data=dF)
> anova( noSex, myMod, pval=TRUE )
```

```
Model 1: ~ Sex * RH * PolViews + AbAny * RH + AbAny * PolViews
Model 2: ~ Sex * RH * PolViews + AbAny * Sex + AbAny * RH + AbAny * PolViews
  resid.df -2*loglik df  change  pval
1       36    -23617
2       34    -23618  2 0.76848 0.681
```

```
> noRH  <- cvam( ~ Sex*RH*PolViews + AbAny*Sex + AbAny*PolViews, data=dF)
> anova( noRH, myMod, pval=TRUE )
```

```
Model 1: ~ Sex * RH * PolViews + AbAny * Sex + AbAny * PolViews
Model 2: ~ Sex * RH * PolViews + AbAny * Sex + AbAny * RH + AbAny * PolViews
  resid.df -2*loglik df change  pval
1       40    -23601
2       34    -23618  6  16.56 0.011
```

```
> noPol <- cvam( ~ Sex*RH*PolViews + AbAny*Sex + AbAny*RH, data=dF)
> anova( noPol, myMod, pval=TRUE )
```

```
Model 1: ~ Sex * RH * PolViews + AbAny * Sex + AbAny * RH
Model 2: ~ Sex * RH * PolViews + AbAny * Sex + AbAny * RH + AbAny * PolViews
  resid.df -2*loglik df change pval
1       38    -23533
2       34    -23618  4 84.486    0
```

Both RH and PolViews are strong predictors of AbAny, but Sex is clearly not.

4.5 Control parameters

Many aspects of the behavior of cvam are determined by control paramaters, the internal settings for determining whether EM has converged, for deciding how many

iterations to take before stopping, and so on. Control parameters, which should not be confused with model parameters, may be changed through the optional argument `control`. This argument should be a named list. That list is processed by another function, `cvamControl`, which sets the control parameters to the user-supplied values and applies default values to any that were not supplied. Typing

```
> cvamControl()
```

will display the names and default values for all control parameters. For example, the control parameter that judges proximity to a boundary is called `critBoundary`, and its default value `1e-08` means that the estimates are considered to be at or near a boundary if the fitted probability for any cell is zero when rounded to eight decimal places. To change `critBoundary` to something else, put the desired value into a named list and supply it to `cvam`.

```
> # use a boundary criterion that is less strict
> satMod <- cvam( ~ Sex*RH*PolViews*AbAny, data=dF, saturated=TRUE,
+    control=list(critBoundary=1e+06 ) )

Note: Estimate at or near boundary
```

By extracting and rounding the fitted probabilities, we can see exactly which cells have estimates close to zero.

```
> round( get.fitted(satMod, type="prob")$fit, 6)

 [1] 0.029066 0.039228 0.007533 0.002446 0.001595 0.003044 0.002963 0.000505
 [9] 0.064676 0.054623 0.016674 0.006016 0.004030 0.001907 0.003721 0.003368
[17] 0.066472 0.045043 0.009213 0.003694 0.001559 0.001708 0.007848 0.003118
[25] 0.095406 0.095293 0.018363 0.005178 0.001465 0.000000 0.007574 0.010448
[33] 0.095604 0.052458 0.022985 0.022680 0.007306 0.003615 0.010528 0.009189
[41] 0.042217 0.036816 0.010613 0.009881 0.002851 0.000000 0.009341 0.003602
[49] 0.005072 0.006238 0.001320 0.001171 0.000000 0.000981 0.000572 0.001010
[57] 0.008873 0.007538 0.001618 0.000905 0.000000 0.000000 0.001497 0.000000
[65] 0.005163 0.002090 0.000000 0.000832 0.000000 0.001656 0.000000 0.000000
```

4.6 Flattening constants and prior information

Estimates near a boundary may cause a variety of problems. If the model is fit using `saturated=FALSE`, then, depending on the complexity of the model, some of the estimated coefficients might run away toward $\pm\infty$. Standard errors computed by `cvam` assume that the loglikelihood function is concave and approximately quadratic in the vicinity of the ML solution; this may not be the case if the solution lies near a boundary, making the reported standard errors unreliable or causing the

procedure to fail. Boundary estimates can also make it difficult to compare models by hypothesis tests. Rules for counting degrees of freedom in boundary solutions are difficult to implement, and many software packages give incorrect results (Bishop et al., 1975; Clogg et al., 1991). The situation becomes even harder when missing data are involved (Fuchs, 1982; Little and Rubin, 2002).

To address this problem, `cvam` allows you to introduce a flattening constant, a small, positive number that is added to each cell of the complete-data table during the fitting procedure. Flattening constants are helpful in sparse, high-dimensional tables where many cells have no observations in them. From a Bayesian standpoint, they can be viewed as a certain kind of prior distribution called a data-augmentation prior (DAP) (Bedrick et al., 1996). Using a DAP is functionally equivalent to adding fictitious observations to a dataset that look like frequencies but are not necessarily integers. DAPs are convenient when comparing the fit of alternative models, because the same prior can be applied to any model regardless of how the parameters are defined. From a non-Bayesian perspective, a flattening constant can be regarded as adding a penalty function that penalizes the fit of estimates near the boundary. The effect of a flattening constant is to smooth the estimate of π and μ toward a uniform table, one in which all cells have equal probability, and to rein in the elements of β to keep them from running away. If the flattening constant is small, the effect on parameter estimates is barely noticeable, but the computations for saturated and non-saturated models are effectively stabilized.

To apply a flattening constant, use the `prior` argument to `cvam`. This argument must be a prior distribution created by the function `cvamPrior`. For example,

```
> myPrior <- cvamPrior( flatten=7.2 )
```

introduces information equivalent to 7.2 prior observations, distributed equally across the cells of the complete-data table. There are 72 cells in the current example, so each cell receives a prior count of 0.1.

```
> # re-fit and compare models using the flattening constant
> myMod <- cvam( myFormula, data=dF, prior=myPrior )
> satMod <- cvam( ~ Sex*RH*PolViews*AbAny, data=dF,
+    saturated=TRUE, prior=myPrior )
```

With prior information, the function being maximized is not the loglikelihood, but an objective function called $\log P$ described in Appendix H. The solution is no longer an ML estimate, but a penalized ML estimate or posterior mode. When comparing the fit of alternative models, we recommend that you use the same prior for every model. We also suggest that you test hypotheses not by an LR test but by comparing values of $-2 \log P$, which can be done by calling `anova` with the argument `method="logP"`.

```
> anova( myMod, satMod, pval=TRUE, method="logP")

Model 1: ~ Sex * RH * PolViews + AbAny * Sex + AbAny * RH + AbAny * PolViews
Model 2: ~ Sex * RH * PolViews * AbAny
  resid.df -2*logP df change   pval
1       34  -23654
2        0  -23692 34 38.647 0.2677
```

With `cvamPrior`, you can also introduce more targeted prior information that applies prior frequencies to subsets of cells in the complete-data table. These frequencies are called nuggets. They resemble coarsened data, and we provide them to `cvamPrior` as a list. Prior nuggets should be used sparingly, but can be very helpful for certain types of models, especially those involving latent classes, which we discuss in Section 8.

4.7 Running `cvam` on a `cvam` object

Thus far, we have called the `cvam` function by supplying a model formula as its first argument. We may also call `cvam` with the first argument being a `cvam` object. Loosely speaking, running `cvam` on a `cvam` object means, "Carry on; do more of the same, unless I specifically tell you otherwise." For example, if the `cvam` object holds the results from an EM run, running `cvam` on that object will restart EM from where it stopped.

```
> # fit the saturated model to the crime data
> result <- cvam( ~ V1 * V2, data=crime, freq=n)
> # run it again, starting from the previous result
> result <- cvam(result)
> summary(result, showCoef=FALSE)

~ V1 * V2

Prior:
      Flattening frequency = 0
Total nuggets + flattening = 0
              Ridge factor = 0
          Intensity factor = 1

Sample size:
              total N in supplied data = 756
N from supplied data used in model fit = 641
           prior effective sample size =   0

Degrees of freedom:
    patterns of coarsened data = 9
  cells in complete-data table = 4
cells without latent variables = 4
         structural zero cells = 0
   parameters in Poisson model = 4
                            df = 0

Starting values:
```

```
supplied by user
jitter SD = 0.000000

EM algorithm:
Converged at iteration 1
Gradient length = 0.000000

  Final logP = 2939.299
Final loglik = 2939.299
```

In this example, when `cvam` was called the second time, EM started to run again, but stopped after just one iteration because it had already converged.

Running `cvam` on a `cvam` object is helpful in problems with high rates of missing information, which causes EM to converge slowly. By default, `cvam` stops if EM has not converged by 500 cycles. That limit is set by the control parameter `iterMaxEM`. If EM fails to converge, we could increase that limit and start over, hoping that it will converge by the new limit. A better option is to continue the EM run from where it stopped, using the final parameter values from the first run as the starting values for the next run.

Running `cvam` on a `cvam` object is also helpful for Markov chain Monte Carlo (MCMC), which is discussed in Section 6. Results from EM can be good starting values for MCMC, as can the results from another MCMC run. Simulated values from MCMC can also serve as random starting values for EM, which can help us to detect multiple modes.

5 Estimates, predictions, imputations, and likelihoods

5.1 Estimated marginal and conditional probabilities

Coefficients of a log-linear model can be difficult to interpret and depend on how the $\boldsymbol{X}$ matrix is coded. For those who are brave enough to try, the model matrix used in a `cvam` fit can be examined using `get.modelMatrix`. However, many analysts prefer to work with probabilities, and after fitting a `cvam` model, you can request a wide variety of estimated marginal and conditional probabilities with the function `cvamEstimate`.

This function has two required arguments. The first is a one-sided formula that specifies the desired probabilities, and the second is a `cvam` object containing the results from a model fit. In the example below, we request the estimated probabilities for the 2×3 marginal table that classifies persons by `Sex` and `AbAny`.

```
> cvamEstimate( ~ Sex + AbAny, myMod )
```

```
Estimates and SE's from EM, linearized
~ Sex + AbAny
     Sex AbAny   prob     SE prob.lower prob.upper
1 Female   Yes 0.2153 0.0091     0.1979     0.2337
2   Male   Yes 0.1638 0.0081     0.1485     0.1803
3 Female    No 0.3233 0.0101     0.3038     0.3434
4   Male    No 0.2497 0.0092     0.2321     0.2681
5 Female    DK 0.0250 0.0036     0.0188     0.0332
6   Male    DK 0.0230 0.0034     0.0171     0.0308
```

The factors in the formula are separated by '+', which does not imply any kind of additive structure, but merely signals that we are adding more dimensions to the table of requested probabilities. The result of a call to `cvamEstimate` is a data frame with estimated probabilities and standard errors. The latter are computed by the delta method, a first-order Taylor approximation based on the estimated covariance matrix for $\hat{\boldsymbol{\beta}}$, as described in Appendix J. A symmetric confidence interval for a probability based on a normal approximation may work poorly, especially when the estimate lies close to zero or one; the endpoints may even stray outside the parameter space. The data frame from `cvamEstimate` also gives lower and upper limits for approximate confidence intervals based on a logistic transformation. These alternative intervals are asymmetric, and the limits stay between 0 and 1. By default, the level of confidence is 95%, but this can be changed through the `confidence` argument.

For this example, it seems more natural to examine the conditional probabilities for `AbAny` given `Sex`. To ask for conditional probabilities, put the symbol '|' into the formula.

```
> # estimated conditional probabilities for AbAny given Sex
> cvamEstimate( ~ AbAny | Sex, myMod )

Estimates and SE's from EM, linearized
~ AbAny | Sex
     Sex AbAny   prob     SE prob.lower prob.upper
1 Female   Yes 0.3820 0.0149     0.3532     0.4117
2 Female    No 0.5736 0.0152     0.5436     0.6032
3 Female    DK 0.0444 0.0064     0.0334     0.0587
4   Male   Yes 0.3752 0.0168     0.3430     0.4086
5   Male    No 0.5721 0.0171     0.5382     0.6053
6   Male    DK 0.0527 0.0078     0.0394     0.0703
```

To condition on two or more variables, put them after '|' and separate them with '+'. Conditional probabilities may be easier to understand when displayed as a flat table.

```
> # conditional probabilities for AbAny given RH and PolViews
> est <- cvamEstimate( ~ AbAny | RH + PolViews, myMod )
> # reshape the probabilities into a three-dimensional array
> xtab <- xtabs( prob ~ AbAny + RH + PolViews, data = est )
> # display the array as a flat table
> ftable( xtab, row.vars=c("PolViews", "RH"), col.vars="AbAny" )
```

```
                      AbAny    Yes     No     DK
PolViews RH
Con      nonHispWhite       0.2722 0.6803 0.0475
         nonHispBlack       0.1865 0.7744 0.0391
         nonHispOther       0.2987 0.5693 0.1320
         Hisp               0.1701 0.7848 0.0451
Mod      nonHispWhite       0.4165 0.5337 0.0498
         nonHispBlack       0.3067 0.6472 0.0461
         nonHispOther       0.4394 0.4319 0.1287
         Hisp               0.2821 0.6663 0.0516
Lib      nonHispWhite       0.5495 0.4120 0.0385
         nonHispBlack       0.4305 0.5316 0.0379
         nonHispOther       0.5719 0.3275 0.1007
         Hisp               0.4010 0.5573 0.0418
```

5.2 Prediction

Predictions from a fitted model are computed by the function `cvamPredict`. In `cvam`, prediction has a different meaning from the way it is used in regression analysis. In regression, prediction is to compute the estimated mean response at specific values for the covariates. In `cvam`, prediction is to compute the predictive distribution for one or more variables, given specific values for the possibly incomplete and coarsened data. Let

- $\boldsymbol{V}_i = (V_{i1}, \ldots, V_{ij})$ denote the vector of true (uncoarsened) variables for observational unit i, with possible value $\boldsymbol{v} = (v_1, \ldots, v_J)$; let
- $\boldsymbol{A}_i$ denote a subset of these variables, with possible value $\boldsymbol{a}$; and let
- $\boldsymbol{V}_i^* = (V_{i1}^*, \ldots, V_{iJ}^*)$ denote the coarsened version of $\boldsymbol{V}_i$.

When calling `cvamPredict`, you supply a one-sided formula that specifies the variables to be predicted (the variables in $\boldsymbol{A}_i$) and a `cvam` object that contains a fitted model. You must also supply a prediction frame, a data frame whose rows are the $\boldsymbol{V}_i^*$'s for which predictions are desired. The prediction frame is not necessarily the dataset that was used to fit the model. The result from `cvamPredict` is the set of probabilities $P(\boldsymbol{A}_i = \boldsymbol{a} \,|\, \boldsymbol{V}_i^*, \boldsymbol{\mu} = \hat{\boldsymbol{\mu}})$ for all possible $\boldsymbol{a}$, where $\hat{\boldsymbol{\mu}}$ is the set of estimated parameters from the fitted model. These probabilities are returned as a matrix with the same number of rows as the prediction frame, and one column for each of the possible $\boldsymbol{a}$. Each row of the result sums to one.

To see how this works, recall the `crime` dataset introduced in Section 4.1. In the code below, we fit the non-independence model and generate predictions for `V1` using `crime` as the prediction frame.

```
> # display the crime data
> crime
```

```
    V1   V2   n
1   no   no 392
2  yes   no  76
3 <NA>   no  31
4   no  yes  55
5  yes  yes  38
6 <NA>  yes   7
7   no <NA>  33
8  yes <NA>   9
9 <NA> <NA> 115

> # fit the model of non-independence
> fit <- cvam( ~ V1 * V2, data=crime, freq=n )
> # display predictions for V1
> cvamPredict( ~ V1, fit, data=crime )

         no       yes
1 1.0000000 0.0000000
2 0.0000000 1.0000000
3 0.8369768 0.1630232
4 1.0000000 0.0000000
5 0.0000000 1.0000000
6 0.5902705 0.4097295
7 1.0000000 0.0000000
8 0.0000000 1.0000000
9 0.7957538 0.2042462
```

In the first row of the result, the predicted probabilities of `V1="no"` and `"yes"` are `1.0000000` and `0.0000000`, because the first row of `crime` has `V1` observed to be `"no"`. In the second row of the result, the predicted probabilities are `0.0000000` and `1.0000000`, because the second row of `crime` has `V1` observed to be `"yes"` In the third row of the result, the predicted probabilities are `0.8369768` and `0.1630232`. Notice that that third row of `crime` has `V1=NA` and `V2="no"`. To predict `V1` for that row, `cvamPredict` takes the estimated parameters from the fitted model and computes from them the conditional probabilities for `V1="no"` and `V1="yes"` given `V1=NA` and `V2="no"`.

By default, `cvamPredict` interprets the prediction frame as microdata. If frequencies are present, these can be supplied through the argument `freq`. In that case, `cvamPredict` returns a matrix of frequencies, with each row of the result summing to the corresponding frequency in the prediction frame.

```
> # display predicted frequencies for V1
> cvamPredict( ~ V1, fit, data=crime, freq=n )

          no       yes
1 392.000000  0.000000
2   0.000000 76.000000
3  25.946282  5.053718
4  55.000000  0.000000
5   0.000000 38.000000
6   4.131894  2.868106
7  33.000000  0.000000
8   0.000000  9.000000
9  91.511685 23.488315
```

If the `cvamPredict` formula has more than one variable, the result will have one column for each possible combination of those variables.

```
> # display predicted frequencies for V1 and V2
> cvamPredict( ~ V1 + V2, fit, data=crime, freq=n )

      no.no    yes.no    no.yes   yes.yes
1 392.00000  0.000000  0.000000  0.000000
2   0.00000 76.000000  0.000000  0.000000
3  25.94628  5.053718  0.000000  0.000000
4   0.00000  0.000000 55.000000  0.000000
5   0.00000  0.000000  0.000000 38.000000
6   0.00000  0.000000  4.131894  2.868106
7  28.90978  0.000000  4.090215  0.000000
8   0.00000  5.983207  0.000000  3.016793
9  80.16919 15.615049 11.342500  7.873266
```

Variables in the formula should be separated by '+'. The conditioning symbol '|' should not appear in the formula, because `cvamPredict` automatically conditions on all data supplied in the prediction frame. As variables are added to the formula, the output from `cvamPredict` becomes wider and more unwieldly. An alternative to prediction that avoids this problem is imputation.

5.3 Imputation

The function `cvamImpute` generates random imputations for all variables in a `cvam` model. When calling this function, the user supplies a `cvam` object containing a fitted model, and an imputation frame containing possibly incomplete or coarsened data. For each row $\boldsymbol{V}_i^*$ of the imputation frame, the true variables $\boldsymbol{V}_i$ are drawn from their joint predictive distribution $P(\boldsymbol{V}_i \,|\, \boldsymbol{V}_i^*, \boldsymbol{\mu} = \hat{\boldsymbol{\mu}})$, where $\hat{\boldsymbol{\mu}}$ is the set of estimated parameters from the fitted model. If the imputation frame has frequencies, these should be declared through the argument `freq`. No formula is needed, because `cvamImpute` automatically imputes all variables in the model given all the information in the imputation frame.

The `cvamImpute` uses R's internal random number generators. To make your results reproducible, set the random generator seeds beforehand using `set.seed`.

```
> set.seed(69852)
> cvamImpute( fit, data=crime )

   V1  V2
1  no  no
2 yes  no
3  no  no
4  no yes
5 yes yes
6  no yes
7  no  no
8 yes yes
9  no  no
```

In this example, we did not supply frequencies, so `cvamImpute` interpreted the imputation frame as microdata and returned a dataset with one row for every row of `crime`. If you supply frequencies, the result will have one row for every cell of the complete-data table and a variable named `freq` that holds the imputed frequencies.

```
> cvamImpute( fit, data=crime, freq=n )

   V1  V2 freq
1  no  no  527
2 yes  no  100
3  no yes   76
4 yes yes   53
```

By repeatedly calling `cvamImpute` with the same imputation frame, you can generate multiple versions of the true data, any of which is plausible under the fitted model. If these repeated calls use the same `cvam` object, however, then they will not be multiple imputations in the sense defined by Rubin (1987), because they will have all been generated under the same set of estimated parameters $\hat{\boldsymbol{\mu}}$. Proper multiple imputations must reflect uncertainty in model parameters along with uncertainty due to missing and coarsened values. Imputations that reflect parameter uncertainty can be generated by Markov chain Monte Carlo, which we will discuss in Section 6.

5.4 Likelihood values

The function `cvamLik` computes likelihood values for rows of a user-supplied data under a fitted model. These should not be confused with the value of the overall loglikelihood function achieved by the model provided by `get.loglik`. Let $\boldsymbol{V}_i^*$ denote coarsened data for a single observational unit i, and let $\boldsymbol{V}_i$ denote the underlying true data for tht unit. Let $\boldsymbol{A}_i^*$ denote a subset of the variables in $\boldsymbol{V}_i^*$ for which the likelihood is desired, and let $\boldsymbol{B}_i$ denote a subset of the variables in $\boldsymbol{V}_i$ to be condition on and treated as fixed. Variables in $\boldsymbol{A}_i^*$ and $\boldsymbol{B}_i$ must not overlap. The likelihood computed by `cvamLik` is

$$L(\boldsymbol{A}_i^* = \boldsymbol{a}^* \mid \boldsymbol{B}_i = \boldsymbol{b}, \boldsymbol{\mu} = \hat{\boldsymbol{\mu}}) \propto P(\boldsymbol{A}_i^* = \boldsymbol{a}^* \mid \boldsymbol{B}_i = \boldsymbol{b}, \boldsymbol{\mu} = \hat{\boldsymbol{\mu}}), \tag{8}$$

where $\boldsymbol{a}^*$ and $\boldsymbol{b}$ are the specific values for $\boldsymbol{A}_i^*$ and $\boldsymbol{B}_i$ seen in a row of the user-supplied data frame, and $\hat{\boldsymbol{\mu}}$ are estimated parameters in a `cvam` object. The likelihoods are computed as though they were probabilities, by summing the conditional probabilities $P(\boldsymbol{A}_i \mid \boldsymbol{B}_i = \boldsymbol{b}, \boldsymbol{\mu} = \hat{\boldsymbol{\mu}})$ over all possible values of the true variables $\boldsymbol{A}_i$ that are consistent with $\boldsymbol{A}_i^* = \boldsymbol{a}^*$. They are not probabilities, however, because `cvam` does not model the coarsened-data mechanism but regards it as ignorable. The probability of $\boldsymbol{A}_i^* = \boldsymbol{a}^*$ depends on the unmodeled mechanism and differs from the

likelihood by an unknown multiplicative constant. Likelihoods from `cvamLik` are useful for computing the odds of observing $\boldsymbol{A}_i^* = \boldsymbol{a}^*$ under alternative and possibly counterfactual versions of $\boldsymbol{b}$, because unknown proportionality constants cancel out in the the ratios. Arguments to `cvamLik` include

- a one-sided formula that specifies the variables in $\boldsymbol{A}_i^*$ and the variables in $\boldsymbol{B}_i$, with the two groups of variables separated by '|', and variables within each group separated by '+';
- a `cvam` object from a fitted model to provide parameter estimates; and
- a data frame whose rows contain the specific values $\boldsymbol{A}_i^* = \boldsymbol{a}^*$ and $\boldsymbol{B}_i = \boldsymbol{b}$ for computing the likelihoods.

The result is a data frame identical to the one supplied by the user, with an additional numeric variable `likVal` holding the likelihood values.

6 Bayesian methods and Markov chain Monte Carlo

6.1 Simulating draws from an approximate posterior distribution

When the number of sampled observations N is large relative to the number of parameters being estimated, the difference between the estimated coefficients $\hat{\boldsymbol{\beta}}$ from EM and the true coefficients $\boldsymbol{\beta}$ is approximately normally distributed with a covariance matrix $\hat{V}(\hat{\boldsymbol{\beta}})$. From a Bayesian perspective, we can treat $N(\hat{\boldsymbol{\beta}}, \hat{V}(\hat{\boldsymbol{\beta}}))$ as an approximate posterior distribution for $\boldsymbol{\beta}$ given the data used to fit the model. By simulating random draws of $\boldsymbol{\beta}$ from this distribution, we can perform approximate Bayesian inference for any parameter that can be expressed as a function of $\boldsymbol{\beta}$ or $\boldsymbol{\pi}$.

To simulate random draws from this approximate posterior distribution, apply the `cvam` function to a `cvam` object as described in Section 4.7, with the additional argument `method="approxBayes"`. The `cvam` object must hold the results from an EM run, otherwise the software won't have access to $\hat{\boldsymbol{\beta}}$ or $\hat{V}(\hat{\boldsymbol{\beta}})$. The simulation uses R's internal random number generators. To make your results reproducible, set the random generator seeds beforehand with `set.seed`.

```
> # fit the non-independence model to the crime data
> fitML <- cvam( ~ V1 * V2, data=crime, freq=n )
> # display the ML estimate for beta and pi
> get.coef( fitML )
```

```
(Intercept)         V11         V21     V11:V21
  4.6241983   0.5002470   0.6600862   0.3177051

> get.fitted( fitML, type="prob" )$fit

[1] 0.69712335 0.13578303 0.09863044 0.06846318

> # draw from the approximate posterior, display new beta and pi
> set.seed(83425)
> obj <- cvam(fitML, method="approxBayes")
> get.coef( obj )

(Intercept)         V11         V21     V11:V21
  4.6401653   0.6145345   0.6493728   0.2478188

> get.fitted( obj, type="prob" )$fit

[1] 0.70210490 0.12513288 0.11671087 0.05605134
```

By default, calling `cvam` with `method="approxBayes"` will draw one value of $\boldsymbol{\beta}$. This can be changed through the control parameter `iterApproxBayes`. In the example below, we simulate $5,000$ values of $\boldsymbol{\beta}$. These are stored in the `cvam` object and can be retrieved with `get.coefSeries`. By setting the control parameter `saveProbSeries` to `TRUE`, we instruct `cvam` to also store the $5,000$ simulated $\boldsymbol{\pi}$ vectors, which can then be retrieved with `get.probSeries`.

```
> # produce 5,000 draws of beta, saving also the resulting pi vectors
> obj <- cvam(fitML, method="approxBayes",
+    control=list(iterApproxBayes=5000, saveProbSeries=TRUE) )
> # display the first few beta and pi vectors
> head( get.coefSeries(obj) )

     (Intercept)       V11       V21   V11:V21
[1,]    4.621281 0.4904044 0.6591652 0.3329766
[2,]    4.579132 0.5171364 0.6917306 0.3486476
[3,]    4.593146 0.4306763 0.6814657 0.3789800
[4,]    4.632846 0.5294656 0.6763896 0.3347828
[5,]    4.570942 0.5765167 0.6637533 0.2605799
[6,]    4.547775 0.4621952 0.6455106 0.4112865

> head( get.probSeries(obj) )

         no.no    yes.no     no.yes    yes.yes
[1,] 0.6990577 0.1346894 0.09610582 0.07014708
[2,] 0.7189250 0.1272545 0.08974754 0.06407299
[3,] 0.7012110 0.1388642 0.08409251 0.07583234
[4,] 0.7145115 0.1268623 0.09456186 0.06406439
[5,] 0.6999839 0.1312185 0.11021036 0.05858726
[6,] 0.7121421 0.1241281 0.08602891 0.07770083
```

One parameter of interest is the change in victimization rate from the first time period to the second, $\delta = P(\texttt{V2="yes"}) - P(\texttt{V1="yes"})$, which is equivalent to the off-diagonal difference, $\delta = P(\texttt{V1="no", V2="yes"}) - P(\texttt{V1="yes", V2="no"})$. From the saved series, we can easily compute and summarize the $5,000$ values of δ.

```
> pi.series <- get.probSeries(obj)
> delta <- pi.series[,3] - pi.series[,2]
> summary(delta)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.12002 -0.05050 -0.03760 -0.03727 -0.02382  0.03553

> sum( delta > 0 )

[1] 151
```

Most of the δ values are negative, suggesting that the victimization rate has dropped over time. Only 151 are positive, so a simulated Bayesian p-value for testing the null hypothesis $\delta = 0$ against the two-sided alternative is $2 \times (151/5,000) = 0.0604$.

6.2 Markov chain Monte Carlo

With Markov chain Monte Carlo (MCMC), we can put aside the normal approximation and obtain Bayesian answers that are approximately exact. The answers are exact in the sense that, if the MCMC algorithm has run long enough to become independent of the starting values, the simulated parameters are drawn from their actual posterior distribution, with no large-sample approximations. The answers are approximate in the sense that, because MCMC is a type of Monte Carlo simulation, summaries of a posterior distribution from MCMC always contain random noise. That noise can be reduced by performing more iterations, so at least in principle, the MCMC summaries can be made arbitrarily precise. For general-purpose reviews of MCMC and its role in Bayesian inference, see Gamerman and Lopes (2006) or Gelman et al. (2013).

To run MCMC on a `cvam` model, call `cvam` with `method="MCMC"`. For log-linear models fit with `saturated=FALSE`, two versions of MCMC are available. The default version is data augmentation (DA), a two-step procedure that bears a strong resemblance to EM. Within each cycle of DA, any missing or coarsened values in the dataset are imputed under the current value for $\boldsymbol{\beta}$, and $\boldsymbol{\beta}$ is then updated by one step of a Metropolis-Hastings procedure. The other version is a random-walk Metropolis (RWM) algorithm that does not impute the missing or coarsened values at each cycle. For saturated models fit with `saturated=TRUE`, MCMC is always implemented as DA. The choice of algorithm, the number of iterations performed, and many other options pertaining to MCMC can be selected through control parameters. Details of these MCMC procedures and their control parameters are given in Appendix K.

In a moment, we will recreate our simulation of $\delta = \pi_{12} - \pi_{21}$ with the `crime` dataset using MCMC. Before that, let's simply invoke `cvam` with `method="MCMC"` and see what happens.

```
> set.seed(4358)
> fit <- cvam( ~ V1 * V2, data=crime, freq=n, method="MCMC")
> summary(fit)

~ V1 * V2

Prior:
      Flattening frequency = 0
Total nuggets + flattening = 0
              Ridge factor = 0
          Intensity factor = 1

Sample size:
              total N in supplied data = 756
N from supplied data used in model fit = 641
           prior effective sample size =   0

Degrees of freedom:
    patterns of coarsened data = 9
  cells in complete-data table = 4
cells without latent variables = 4
         structural zero cells = 0
   parameters in Poisson model = 4
                            df = 0

Starting values:
default, center
jitter SD = 0.000000

MCMC: Data augumentation (DA) with Metropolis-Hastings

Tuning parameters:
 proposal df = 10
   step size = 0.8
scale factor = 0.8

Accept rate = 0.7936

              Iterations performed = 5000
   Iterations discarded as burn-in =    0
          Iterations after burn-in = 5000
Thinning interval for saved series =    1
           Samples in saved series = 5000
               Imputation interval =    0
      Number of imputations stored =    0

Direct estimates and SE's from 5000 successive MCMC samples
              coef      SE zstat pval
(Intercept) 4.6104 0.05779 79.77    0
V11         0.5015 0.06015  8.34    0
V21         0.6657 0.06018 11.06    0
V11:V21     0.3209 0.06277  5.11    0
```

The output from `summary` explains that `cvam` ran the DA algorithm for 5,000 iterations. The number of iterations can be changed through the control parameter `iterMCMC`. The coefficients and standard errors displayed in the summary are "direct estimates," which means that they were computed from a running average and standard deviation of the simulated β values over the iterations of MCMC. The precision of these Monte Carlo summaries depends on the number of iterations and on the convergence behavior of the Markov chain. The series of β values was

stored in the `cvam` object and can be retrieved with `get.coefSeries`. By default, `get.coefSeries` returns the series as an `mcmc` object from the package `coda` (Plummer et al., 2006), which provides a variety of tools for assessing convergence and analyzing the output from MCMC runs. For example, the `summary` method displays means, standard deviations and quantiles.

```
> betaSeries <- get.coefSeries( fit )
> library(coda)
> summary( betaSeries )

Iterations = 1:5000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 5000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

              Mean      SD  Naive SE Time-series SE
(Intercept) 4.6104 0.05780 0.0008174       0.001691
V11         0.5015 0.06016 0.0008507       0.001395
V21         0.6657 0.06019 0.0008512       0.001413
V11:V21     0.3209 0.06277 0.0008877       0.001531

2. Quantiles for each variable:

              2.5%    25%    50%    75%  97.5%
(Intercept) 4.4974 4.5725 4.6105 4.6495 4.7202
V11         0.3807 0.4624 0.5018 0.5397 0.6213
V21         0.5516 0.6251 0.6645 0.7066 0.7865
V11:V21     0.1982 0.2792 0.3208 0.3657 0.4396
```

The `plot` method creates trace plots and density estimates,

```
> # display trace plots and density estimates
> plot( betaSeries )
```

with the result shown in Figure 2. The `acfplot` method creates autocorrelation plots,

```
> # display autocorrelation plots
> acfplot( betaSeries )
```

with the result shown in Figure 3.

By default, `cvam` does not store the series of simulated π vectors from an MCMC run, but it does store the final value of π and the running average of π across the iterations. This running average provides the fitted values in the result from `get.fitted`. The result from `get.fitted` also includes predicted true frequencies, which are a running average of the simulated true frequencies across the iterations.

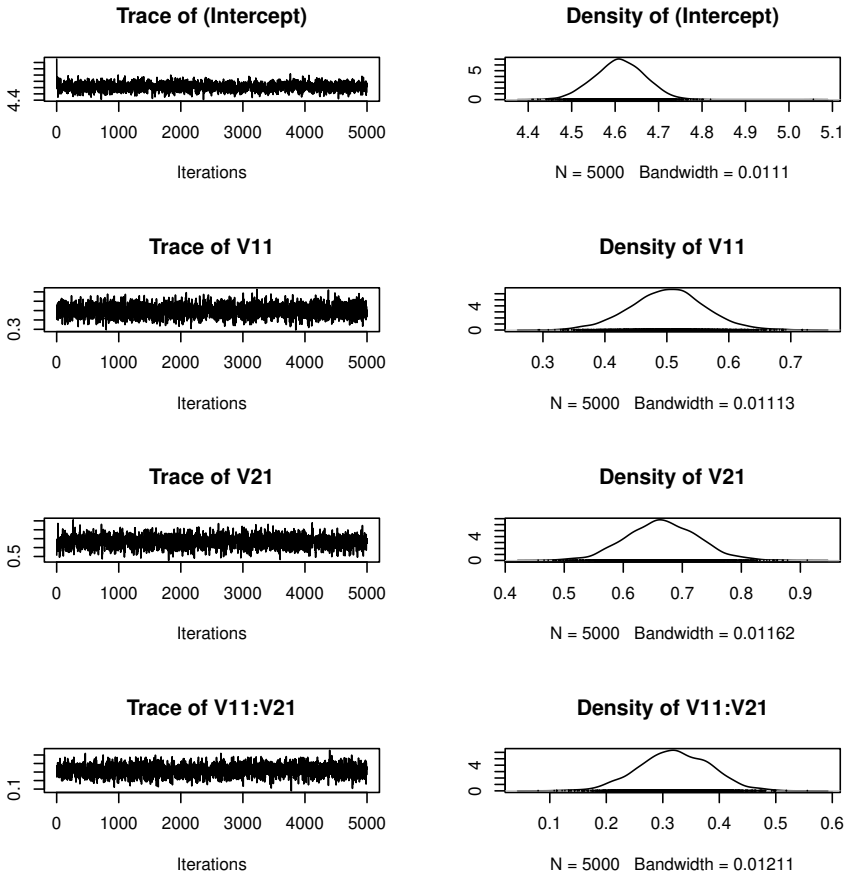

Figure 2: Trace plots and density estimates for log-linear coefficients from the `crime` dataset, produced by `coda`.

45

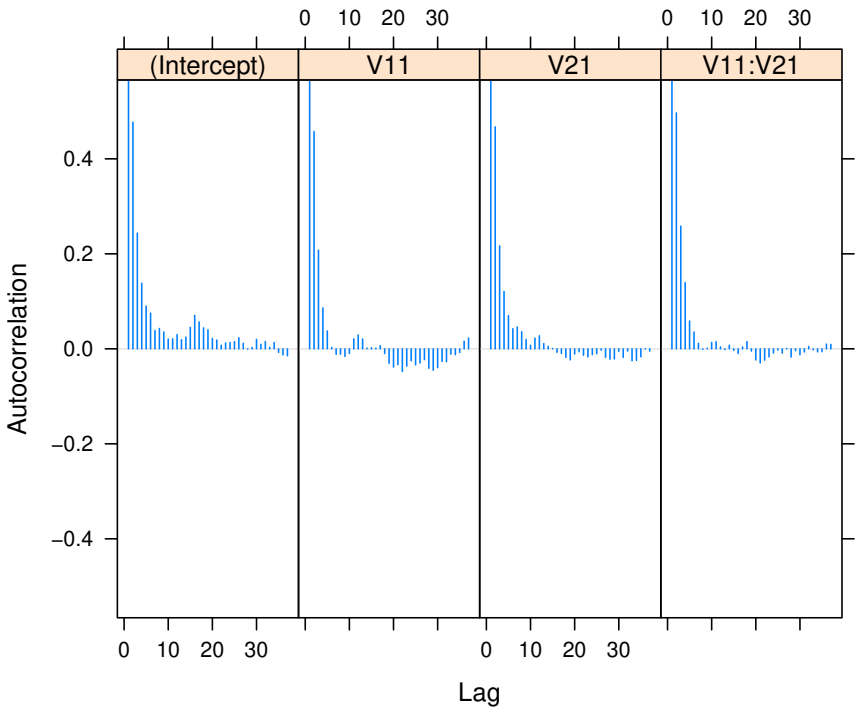

Figure 3: Plots of the autocorrelation function (ACF) for log-linear coefficients from the `crime` dataset, produced by `coda`.

46

```
> get.fitted(fit, type="prob")

   V1  V2     freq       fit
1  no  no 527.3670 0.69829289
2 yes  no 102.4814 0.13547757
3  no yes  74.4200 0.09780841
4 yes yes  51.7316 0.06842113
```

To access the series of π vectors, we will have to run the simulation again with the control parameter `saveProbSeries` set to `TRUE`. In the example below, we repeat the simulation with `saveProbSeries=TRUE`, then we compute and summarize the $5,000$ simulated values of δ.

```
> set.seed(4358)
> fit <- cvam( ~ V1 * V2, data=crime, freq=n, method="MCMC",
+    control=list( saveProbSeries=TRUE ) )
> piSeries <- get.probSeries(fit)
> delta <- piSeries[,3] - piSeries[,2]
> summary(delta)

Iterations = 1:5000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 5000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
    -0.0376692      0.0197574      0.0002794      0.0004454

2. Quantiles for each variable:

     2.5%       25%       50%       75%     97.5%
-0.077161 -0.050420 -0.038159 -0.024072  0.001025

> sum( delta > 0 )

[1] 138
```

The simulated Bayesian p-value of $2 \times (138/5,000) = 0.0552$ is very close to the result we obtained from the approximate Bayesian method.

7 Multiple imputation

7.1 Multiple imputation with MCMC

Multiple imputation (MI) (Rubin, 1987, 1996) is an increasingly popular method for analyzing datasets with missing values. With MI, we can perform the computations that handle the missing values ahead of time, transforming the task into a

series of repeated complete-data analyses. In the notation that we have been using, $\boldsymbol{f}$ denotes the true frequencies that we would want to analyze if there were no missing or coarsened values, the frequencies for the table that cross-classifies sample units by variables $\boldsymbol{V} = (V_1, \ldots, V_J)$. The information available to us is $\boldsymbol{f}^*$, the seen frequencies for patterns of coarsened data. MI requires us to specify an imputation model, a model that is rich enough to preserve the aspects of the joint distribution of $V_1, \ldots, V_J$ that are important for subsequent analyses. With MI, we simulate M independent random draws of $\boldsymbol{f}$ from its posterior predictive distribution under the imputation model,

$$\boldsymbol{f}^{(m)} \sim P(\boldsymbol{f} \mid \boldsymbol{f}^*) \quad \text{independently for } m = 1, \ldots, M. \tag{9}$$

The variability among $\boldsymbol{f}^{(1)}, \ldots, \boldsymbol{f}^{(M)}$ should not only reflect uncertainty due to missing and coarsened values, but also the uncertainty due to the fact that the parameters of the imputation model are unknown. After creating the imputed datasets $\mathrm{f}^{(1)}, \ldots, \mathrm{f}^{(M)}$, we analyze each one as if it were the true $\boldsymbol{f}$, saving the estimates and measures of uncertainty, and then combine the results using procedures described by Rubin (1987), Barnard and Rubin (1999), and others.

With `cvam`, we can create the imputations in (9) by first drawing M independent values of the imputation model parameters using an approximate Bayes or MCMC procedure described in Section 6, and then, from each set of parameters, generating an imputed dataset using `cvamImpute`. If the imputation frame supplied to `cvamImpute` contains aggregated data and frequencies, the result will be M versions of a true data frame, each having its own version of $\boldsymbol{f}$. If the imputation frame supplied to `cvamImpute` contains microdata, the result will be M different versions of the microdata with no missing or coarsened values. In the example below, we use the `crime` dataset to create $M = 10$ imputed versions of the 2×2 table.

```
> impList <- as.list(1:10) # a list to store the imputed datasets
> set.seed(769090)         # for reproducibility
> for(m in 1:10) {
+    # run MCMC under the non-independence model
+    tmp <- cvam( ~ V1 * V2, data=crime, freq=n, method="MCMC")
+    # impute under the simulated parameters
+    impList[[m]] <- cvamImpute( tmp, crime, freq=n)
+ }
> # display the first two imputations
> impList[1:2]

[[1]]
   V1  V2 freq
1  no  no  530
2 yes  no  100
3  no yes   73
4 yes yes   53

[[2]]
   V1  V2 freq
```

```
1  no  no  534
2 yes  no   99
3  no yes   69
4 yes yes   54
```

An easier way to generate multiple imputations is to perform a single run of MCMC and save simulated values of $\boldsymbol{f}$ along the way, spacing them far enough apart in the iteration sequence to be reasonably sure that they are independent. The number of iterations between successive imputations, which we call the imputation interval, is set by the control parameter `imputeEvery`. Setting `imputeEvery` and the number of iterations `iterMCMC` will determine the number of imputations that are saved. After the MCMC run, the imputed frequencies are retrieved with `get.imputedFreq`.

```
> # run MCMC for 5,000 iterations, saving an imputation at every 500th
> result <- cvam( ~ V1 * V2, data=crime, freq=n, method="MCMC",
+    control=list( iterMCMC=5000, imputeEvery=500 ) )
> get.imputedFreq(result)

   V1  V2 imp.1 imp.2 imp.3 imp.4 imp.5 imp.6 imp.7 imp.8 imp.9 imp.10
1  no  no   528   527   541   523   529   522   530   515   531    528
2 yes  no   102   104    90   105   103   110   103   109    96    104
3  no yes    72    71    70    70    69    73    73    77    73     67
4 yes yes    54    54    55    58    55    51    50    55    56     57
```

7.2 Multiple imputation with an approximate posterior

We can also perform MI by taking M draws of $\boldsymbol{\beta}$ from the approximate posterior distribution using `method="approxBayes"` and generating an imputation for each one. Draws from the approximate posterior are independent, so an imputation interval greater than one is unnecessary. Setting the control parameter `imputeApproxBayes` to `TRUE` will instruct `cvam` to create and store an imputation for every draw.

```
> #  run EM, then create ten imputations with approxBayes
> fitML <- cvam( ~ V1 * V2, data=crime, freq=n )
> result <- cvam( fitML, method="approxBayes",
+    control=list( iterApproxBayes=10, imputeApproxBayes=TRUE ) )
> get.imputedFreq(result)

   V1  V2 imp.1 imp.2 imp.3 imp.4 imp.5 imp.6 imp.7 imp.8 imp.9 imp.10
1  no  no   522   522   529   534   528   528   529   525   530    530
2 yes  no   105   112   107   101    93   101   104   104   104     99
3  no yes    78    72    70    68    80    75    72    72    73     80
4 yes yes    51    50    50    53    55    52    51    55    49     47
```

7.3 Combining results from repeated-imputation inferences

Rules for consolidating the results from a multiply-imputed data analysis are implemented in the function `miInference`. This function has two required arguments: `est.list`, a list of estimates to be combined, and `std.err.list`, a list of corresponding standard errors. Each list should have M components, where M in the number of imputations. Each component may be a scalar or a vector, and they should all have the same length. For example, suppose that the imputed datasets are analyzed by fitting a logistic regression model with 12 coefficients. Each component of `est.list` will be a vector of 12 estimated coefficients, and each component of `std.err.list` will be a vector of 12 standard errors.

For a very simple example, we will use the imputed versions of `crime` to create an MI-based confidence interval for the odds ratio relating `V1` to `V2`. Given the frequencies for a 2×2 table $\boldsymbol{f} = (f_{11}, f_{12}, f_{21}, f_{22})$, the estimated odds ratio is

$$\hat{\omega} = \frac{f_{11}\, f_{22}}{f_{12}\, f_{21}}.$$

In large samples, the estimated log-odds ratio $\hat{\theta} = \log \hat{\omega}$ is approximately normally distributed around $\theta = \log \omega$, with estimated variance

$$\hat{V}(\hat{\theta}) = \frac{1}{f_{11}} + \frac{1}{f_{12}} + \frac{1}{f_{21}} + \frac{1}{f_{22}}$$

(Agresti, 2013). In the code below, we impute the `crime` dataset $M = 10$ times, compute $\hat{\theta}$ and $\hat{V}(\hat{\theta})$ from each one, and combine the results.

```
> set.seed(54981)
> result <- cvam( fitML, method="MCMC",
+    control=list( iterMCMC=5000, imputeEvery=500 ) )
> impData <- get.imputedFreq(result)[-(1:2)] # just the frequencies
> est.list <- std.err.list <- as.list(1:10)  # to hold the estimates and SEs
> for( m in 1:10 ) {
+    f <- impData[,m]
+    est.list[[m]] <- log( (f[1] * f[4]) / (f[2] * f[3]) )
+    std.err.list[[m]] <- sqrt( sum(1/f) )
+ }
> miInference( est.list, std.err.list )

       Est      SE Est/SE   df p Pct.mis
[1,] 1.2773 0.26669  4.789 64.1 0    37.5
```

The combined estimate of the log-odds ratio is 1.28 with a standard error of 0.27. The test of the null hypothesis $\theta = 0$ against a two-sided alternative yields a p-value that is essentially zero, and the estimated rate of missing information is 37.5%. For more information on these quantities, see `?miInference`.

7.4 A more detailed example of multiple imputation

For a more elaborate illustration, we return to the four-variable example of Section 4.4 and produce multiple imputations for the microdata. Our imputation model will be saturated, which will preserve all possible associations among the variables and avoid introducing bias into post-imputation analyses. To start, we fit the saturated model using EM, then try a test run of MCMC starting from the EM estimate.

```
> # put the four variables into a data frame
> dF <- data.frame( Sex = abortion2000$Sex, RH = RH,
+    PolViews = abortion2000$PolViews, AbAny = abortion2000$AbAny )
> # fit the saturated model with EM, then do a test run of MCMC
> fitEM <- cvam( ~ Sex * RH * PolViews * AbAny, data=dF )
> set.seed(598902)
> fitMCMC <- cvam( fitEM, method="MCMC")

Note: Overflow; cell mean became too large
Cell 46
OCCURRED IN: compute_mu_from_beta in MOD cvam_engine
OCCURRED IN: run_mh_step_beta_da in MOD cvam_engine
MCMC procedure aborted
Iteration 1
OCCURRED IN: run_da_log_linear in MOD cvam_engine
```

To understand why MCMC failed, let's examine some of the results from EM.

```
> # display fitted cell probs, rounded to five decimal places
> round( get.fitted(fitEM, type="prob", mfTrue=FALSE ), 5)

 [1] 0.02907 0.03923 0.00753 0.00245 0.00160 0.00304 0.00296 0.00050 0.06468
[10] 0.05462 0.01667 0.00602 0.00403 0.00191 0.00372 0.00337 0.06647 0.04504
[19] 0.00921 0.00369 0.00156 0.00171 0.00785 0.00312 0.09541 0.09529 0.01836
[28] 0.00518 0.00146 0.00000 0.00757 0.01045 0.09560 0.05246 0.02298 0.02268
[37] 0.00731 0.00361 0.01053 0.00919 0.04222 0.03682 0.01061 0.00988 0.00285
[46] 0.00000 0.00934 0.00360 0.00507 0.00624 0.00132 0.00117 0.00000 0.00098
[55] 0.00057 0.00101 0.00887 0.00754 0.00162 0.00090 0.00000 0.00000 0.00150
[64] 0.00000 0.00516 0.00209 0.00000 0.00083 0.00000 0.00166 0.00000 0.00000

> # display some of the coefs and SEs
> head( get.coef(fitEM, withSE=TRUE) )

                 coef       SE zstat   pval
(Intercept)  0.4681699 119.6099  0.00 0.9969
Sex1         0.1245196 119.6099  0.00 0.9992
RH1          3.8026873 119.6099  0.03 0.9746
RH2          1.1959860 159.5182  0.01 0.9940
RH3         -3.9601814 286.8639 -0.01 0.9890
PolViews.L  -1.4433772 199.2738 -0.01 0.9942
```

Many of the fitted probabilities are close to zero. Standard errors for the coefficients are huge, a telltale sign that the loglikelihood function is poorly shaped and almost non-concave where EM stopped. This example has $N = 2,817$ observations and 72 cells, but many of the cells are empty, causing some aspects of $\boldsymbol{\beta}$ to be poorly

estimated or inestimable. To address this problem, we could simplfy the model by omitting some of the higher-way associations, or we could apply a flattening constant as described in Section 4.6. We could also introduce a ridge factor, a term that penalizes the fit for β values that are far from zero. A ridge factor shrinks the estimated coefficients and moves the estimates away from the boundary, much as a flattening constant does, and it reshapes the fitting function to become more concave. Let's apply a mild ridge factor of 0.5 and see what happens to the fitted probabilities and coefficients.

```
> # re-run EM with a ridge factor of 0.5
> fitEM.ridge <- cvam( ~ Sex * RH * PolViews * AbAny, data=dF,
+    prior=cvamPrior( ridge=.5 ) )
> round( get.fitted(fitEM.ridge, type="prob", mfTrue=FALSE ), 5)

 [1] 0.02909 0.03937 0.00755 0.00252 0.00157 0.00280 0.00297 0.00053 0.06469
[10] 0.05455 0.01666 0.00601 0.00403 0.00202 0.00373 0.00336 0.06641 0.04503
[19] 0.00919 0.00374 0.00165 0.00166 0.00783 0.00310 0.09536 0.09503 0.01834
[28] 0.00515 0.00155 0.00045 0.00757 0.01043 0.09566 0.05265 0.02302 0.02269
[37] 0.00721 0.00324 0.01052 0.00917 0.04230 0.03674 0.01059 0.00979 0.00278
[46] 0.00025 0.00933 0.00361 0.00505 0.00622 0.00125 0.00118 0.00006 0.00096
[55] 0.00057 0.00096 0.00881 0.00744 0.00155 0.00089 0.00013 0.00012 0.00147
[64] 0.00004 0.00507 0.00213 0.00009 0.00085 0.00004 0.00159 0.00004 0.00003

> head( get.coef(fitEM.ridge, withSE=TRUE) )

                 coef        SE zstat   pval
(Intercept)  2.2664510 0.1590153 14.25 0.0000
Sex1         0.1349898 0.1590153  0.85 0.3959
RH1          2.0025760 0.1675005 11.96 0.0000
RH2          0.1415260 0.2112464  0.67 0.5029
RH3         -1.3848754 0.3319072 -4.17 0.0000
PolViews.L  -0.2525115 0.2610683 -0.97 0.3334
```

The fitted probabilities thet were close to zero are slightly larger, the standard errors for the coefficients are dramatically smaller, and the coefficients have changed. Whenever we use prior information to stabilize a model, it is worth asking whether the results have changed too much, and whether the extra information is contradicted by the data. To understand this, let's compare the new and old estimates in terms of -2 times the loglikelihood difference and the likelihood ratio.

```
> -2 * ( get.loglik(fitEM.ridge) - get.loglik(fitEM) )

[1] 1.666639

> exp( get.loglik(fitEM) - get.loglik(fitEM.ridge) )

[1] 2.300944
```

For rough guidance, we may compare -2 times the loglikelihood difference to a chi-squared distribution with 72 degrees of freedom, because the saturated Poisson model has 72 free parameters (one per cell). With $P(\chi^2_{72} \geq 1.67) \approx 1$, there is

essentially no evidence to reject the new estimates in favor of the old. The likelihood ratio can be viewed as a Bayes factor. By a widely used criterion, a Bayes factor between 1 and 3 means that the evidence to prefer the old estimates to the new is "not worth more than a bare mention" (Jeffreys, 1961; Kass and Raftery, 1995).

Adding prior information effectively stabilized the estimates from EM, but the MCMC algorithm still does not work.

```
> set.seed(87900)
> fitMCMC <- cvam( fitEM.ridge, method="MCMC" )

Note: Metropolis-Hastings got stuck
MCMC procedure aborted
Iteration 26
OCCURRED IN: run_da_log_linear in MOD cvam_engine
```

The procedure aborted after 25 iterations, because the Metropolis-Hastings jumping rule failed to generate any plausible new values for β. The default settings for MCMC do not always work, especially when some areas of the complete-data table are sparsely populated. When this happens, we can address the problem by:

- using stronger prior information, either by increasing the ridge factor or introducing a flattening constant. There is a danger of adding too much, so this should be done cautiously and sparingly.
- changing the tuning constants for the proposal distribution through the control parameters `tuneDA`, as described in Appendix K.
- changing the algorithm from data augmentation (DA) to random-walk Metropolis (RWM), by setting the control parameter `typeMCMC` to `"RWM"` and fiddling with the tuning constants `tuneRWM`.

Because this example involves a saturated model, we can also

- run the procedure with `saturated=TRUE`, which uses a different DA algorithm that is more stable and never gets stuck.

Finally, if all else fails, we can still

- switch to the approximate Bayesian procedure described in Section 7.2.

For this example, we switched to RWM, which solved the problem of getting stuck. With the default tuning constants, the acceptance rate was about 61%, which is higher than optimal. In general, RWM performs best when the acceptance rate is 20–40%. Increasing the scale factor for the random-walk proposal will bring down

the acceptace rate, but if we raise the scale factor too much, the algorithm gets stuck. With a little experimentation and a few more test runs, we found that a scale factor of 0.17 struck a nice balance.

```
> set.seed(87900)
> fitMCMC <- cvam( fitEM.ridge, method="MCMC",
+    control=list( typeMCMC="RWM", tuneRWM=c(1000,.17) ) )
```

Plots of the coefficient series created with `coda` (not shown) revealed that this chain mixes poorly, and it takes 500 or more iterations for the ACF functions to die down. Collecting accurate posterior summaries from a parameter series would require extremely long runs of MCMC. In settings like these, multiple imputation has a powerful advantage over direct simulation of the posterior summaries, because with MI, we only need a small number of independent draws. In the example below, we run MCMC for 2,500 cycles, impute the microdata with `cvamImpute`, then repeat the process $M = 25$ times.

```
> M <- 25
> impList <- as.list(1:M)  # dummy list to hold the imputed datasets
> set.seed(2343)
> for( m in 1:M ) {
+    # take 2,500 steps of MCMC, then impute
+    fitMCMC <- cvam( fitMCMC, control=list(iterMCMC=2500) )
+    impList[[m]] <- cvamImpute( fitMCMC, data=dF )
+ }
> # display the first few rows of the original data and
> # the first imputed dataset
> head( dF )
```

```
     Sex           RH PolViews AbAny
1   Male    nonHispNA      Con    No
2 Female         Hisp      Con    No
3 Female    nonHispNA      Con  <NA>
4 Female nonHispWhite      Con  <NA>
5 Female nonHispWhite      Lib  <NA>
6 Female nonHispWhite      Lib   Yes
```

```
> head( impList[[1]] )
```

```
     Sex           RH PolViews AbAny
1   Male nonHispWhite      Con    No
2 Female         Hisp      Con    No
3 Female nonHispWhite      Con    No
4 Female nonHispWhite      Con    No
5 Female nonHispWhite      Lib    No
6 Female nonHispWhite      Lib   Yes
```

We finish this illustration by fitting a logistic regression model to each imputed dataset and combining the results. The response is a binary indicator for `AbAny="Yes"`, and the predictors include main effects for `Sex`, `RH` and `PolViews`.

```
> est.list <- SE.list <- as.list(1:M)
> for( m in 1:M ) {
+    # extract the imputed dataset
+    impData <- impList[[m]]
+    # create the binary response and fit the logit model
+    impData$y <- 1 * ( impData$AbAny == "Yes" )
+    logitFit <- glm( y ~ Sex + RH + PolViews, data=impData,
+       family=binomial() )
+    # extract matrix of coefficients and SEs
+    coefMat <- summary(logitFit)$coef
+    est.list[[m]] <- coefMat[,1]
+    SE.list[[m]]  <- coefMat[,2]
+ }
> # combine the results with Rubin's rules
> miInference( est.list, SE.list )
```

```
                     Est       SE Est/SE    df     p Pct.mis
(Intercept)    -0.3762900 0.076647 -4.909 138.6 0.000    41.6
SexMale         0.0038774 0.101040  0.038 187.1 0.969    35.8
RHnonHispBlack -0.3659200 0.248680 -1.471  40.5 0.149    77.0
RHnonHispOther  0.0729550 0.479760  0.152  39.9 0.880    77.6
RHHisp         -0.5976000 0.205170 -2.913 172.5 0.004    37.3
PolViews.L      0.8583100 0.090432  9.491 237.3 0.000    31.8
PolViews.Q     -0.0234460 0.078080 -0.300 316.2 0.764    27.6
```

The results are consistent with our log-linear analyses in Section 4.4: `RH` and `PolViews` are strong predictors of `AbAny`, but `Sex` is not.

7.5 Creating synthetic data

The function `cvamImpute` can also generate synthetic data, with applications to statistical disclosure limitation (Raghunathan et al., 2003; Reiter, 2004), parametric bootstrapping (Efron, 2012), and Bayesian model criticism through posterior predictive checks (Rubin, 1984; Gelman et al., 2013). To generate synthetic data, call `cvamImpute` with the argument `synthetic=TRUE`. This will cause the function to wipe out all observed values in the imputation frame, replacing them with missing values and then imputing them under the model and parameters in the supplied `cvam` object. If the imputation frame contains microdata, the result will be a dataset of the same size filled with synthetic data. If the imputation frame has grouped data and frequencies, the result is a grouped data frame with one row per cell of the complete-data table and synthetic integer frequencies that add up to the total sample size.

```
> # take 2,500 more steps of MCMC and draw a synthetic dataset
> fitMCMC <- cvam( fitMCMC )
> synthData <- cvamImpute( fitMCMC, data=dF)
> head( synthData )
```

```
     Sex           RH PolViews AbAny
1   Male nonHispWhite      Con    No
2 Female         Hisp      Con    No
3 Female nonHispWhite      Con    No
4 Female nonHispWhite      Con    No
5 Female nonHispWhite      Lib   Yes
6 Female nonHispWhite      Lib   Yes
```

8 Latent-class analysis

8.1 Background

Latent-class (LC) analysis has a long history of use in the social sciences (Lazarsfeld and Henry, 1968; Clogg and Goodman, 1984), medical and psychiatric diagnosis (Formann and Kohlmann, 1996; Bandeen-Roche et al., 1997), analysis of response errors in censuses and surveys (Biemer et al., 2001) and elsewhere. Specialized routines for LC modeling are available in SAS (Collins and Lanza, 2010; Lanza et al., 2015), R (Linzer and Lewis, 2011) Mplus (Muthén and Muthén, 2017) and Latent GOLD (Vermunt and Magidson, 2016). Because LC models are an example of log-linear models with incomplete data (Hagenaars, 1993), we can fit them with `cvam`. Special care is needed, however, because LC models have unique features that cause them to behave differently from other types of log-linear models.

8.2 A simple example with two latent classes

An LC model posits a categorical variable that is completely missing and relies on multiple observable variables to measure it. For example, Yang and Becker (1997) examined the results from diagnostic tests for HIV infection. Four tests, which are labeled `A`, `B`, `C` and `D`, were given to $N = 428$ high-risk patients. The report from each test was either `"neg"` (negative) or `"pos"` (positive). The aggregated results are distributed with `cvam` in a dataset called `hivtest`.

```
> hivtest
```

```
    A   B   C   D COUNT
1 neg neg neg neg   170
2 neg neg neg pos    15
3 neg pos neg neg     6
4 pos neg neg neg     4
5 pos neg neg pos    17
6 pos neg pos pos    83
7 pos pos neg neg     1
8 pos pos neg pos     4
9 pos pos pos pos   128
```

None of these tests is a gold standard; any of them can produce false positives or false negatives. Suppose that each patient has a true infection status, a two-level factor that we will call `L`. That factor is not found in the data frame, but we can create it with the function `latentFactor`. This function accepts two arguments; the first is the length of the latent factor, and the second is its number of levels.

```
> hivtest$L <- latentFactor( NROW(hivtest), 2 )
> hivtest

    A   B   C   D COUNT    L
1 neg neg neg neg   170 <NA>
2 neg neg neg pos    15 <NA>
3 neg pos neg neg     6 <NA>
4 pos neg neg neg     4 <NA>
5 pos neg neg pos    17 <NA>
6 pos neg pos pos    83 <NA>
7 pos pos neg neg     1 <NA>
8 pos pos neg pos     4 <NA>
9 pos pos pos pos   128 <NA>
```

A traditional LC model assumes that the items measuring the latent variable are conditionally independent given the latent variable. That assumption, known as local independence, does not necessarily hold, but it provides a useful place to begin, and we will evaluate the assumption later. For this example, the formula is:

$$\sim \texttt{L*A + L*B + L*C + L*D}$$

Before fitting this model, we need to make two important points. The first point is that *the default starting-value procedure should not be used with an LC model*. By default, `cvam` starts by assigning equal probabilities to all cells of the complete-data table. For an LC model, this happens to be a saddlepoint of the loglikelihood function, a stationary value that is not a maximum, but where EM will stop changing from one iteration to the next. To avoid getting stuck at this saddlepoint, we can set the control parameter `startValJitter` to a small positive value, which adds random Gaussian noise to the starting values.

A second important point about LC models is that *their ML estimates are not unique*. For a model with C classes, there are $C!$ equivalent solutions corresponding to all possible ways that the classes can be ordered from 1 to C. With randomly jittered starting values, we can set the random number generator seed to ensure that, if we need to run the procedure again, we will not converge to a different solution.

```
> # set the RNG seed and fit the model of local independence
> set.seed(125)
> fit <- cvam( ~ L*A + L*B + L*C + L*D, data=hivtest, freq=COUNT,
+    control = list( startValJitter=.1 ) )
```

```
Note: Estimate at or near boundary
Estimated variances may be unreliable
```

The ML estimates lie on a boundary, which is quite common for LC models.

8.3 Parameter estimates

Parameter estimates that are traditionally reported for an LC model include the class prevalences, which are the marginal probabilities for the latent variable, and the measurement parameters, which are the conditional probabilities for each item given the latent variable, We can obtain these with a single call to `cvamEstimate`, putting the formulas into a list.

```
> cvamEstimate( list( ~L, ~A|L, ~B|L, ~C|L, ~D|L ), fit )

Estimates and SE's from EM, linearized
~ L
  L   prob     SE prob.lower prob.upper
1 1 0.5401 0.0242     0.4924     0.5870
2 2 0.4599 0.0242     0.4130     0.5076
~ A | L
  L   A   prob     SE prob.lower prob.upper
1 1 neg 0.0000 0.0000     0.0000     1.0000
2 1 pos 1.0000 0.0000     0.0000     1.0000
3 2 neg 0.9703 0.0131     0.9304     0.9876
4 2 pos 0.0297 0.0131     0.0124     0.0696
~ B | L
  L   B   prob     SE prob.lower prob.upper
1 1 neg 0.4290 0.0327     0.3665     0.4938
2 1 pos 0.5710 0.0327     0.5062     0.6335
3 2 neg 0.9644 0.0132     0.9272     0.9829
4 2 pos 0.0356 0.0132     0.0171     0.0728
~ C | L
  L   C   prob    SE prob.lower prob.upper
1 1 neg 0.0871 0.019     0.0564     0.1323
2 1 pos 0.9129 0.019     0.8677     0.9436
3 2 neg 1.0000 0.000     0.0000     1.0000
4 2 pos 0.0000 0.000     0.0000     1.0000
~ D | L
  L   D   prob   SE prob.lower prob.upper
1 1 neg 0.0000 0.00     0.0000     1.0000
2 1 pos 1.0000 0.00     0.0000     1.0000
3 2 neg 0.9195 0.02     0.8706     0.9509
4 2 pos 0.0805 0.02     0.0491     0.1294
```

For the estimates on a boundary, the reported standard errors are zero, suggesting that the estimates have no uncertainty. That is implausible, and it is one of the reasons why `cvam` issued a warning. Adding even a small amount of prior information through a flattening constant or ridge factor can address that problem. Examining the pattern of conditional probabilities, we see that at this solution, latent class `L=1` contains individuals who are likely to test positive, and class `L=2` contains individuals who are likely to test negative. With different random generator seeds, these

class labels could easily be reversed. Interpreting `L=1` as actual HIV positive and `L=2` as actual HIV negative, the estimated sensitivities are

$$\begin{aligned} P(\texttt{A="pos"} \,|\, \texttt{L=1}) &= 1.000, \\ P(\texttt{B="pos"} \,|\, \texttt{L=1}) &= 0.571, \\ P(\texttt{C="pos"} \,|\, \texttt{L=1}) &= 0.913, \\ P(\texttt{D="pos"} \,|\, \texttt{L=1}) &= 1.000, \end{aligned}$$

and the estimated specificities are

$$\begin{aligned} P(\texttt{A="neg"} \,|\, \texttt{L=2}) &= 0.970, \\ P(\texttt{B="neg"} \,|\, \texttt{L=2}) &= 0.964, \\ P(\texttt{C="neg"} \,|\, \texttt{L=2}) &= 1.000, \\ P(\texttt{D="neg"} \,|\, \texttt{L=2}) &= 0.919. \end{aligned}$$

8.4 Lack-of-fit testing and residuals

The standard lack-of-fit test for LC model compares its loglikelihood to that of a saturated model without the latent variable.

```
> # perform the lack-of-fit test
> fitSat <- cvam( ~ A*B*C*D, data=hivtest, freq=COUNT )
> anova( fit, fitSat, pval=TRUE )

Model 1: ~ L * A + L * B + L * C + L * D
Model 2: ~ A * B * C * D
  resid.df -2*loglik df change   pval
1        6   -3070.8
2        0   -3087.1  6 16.227 0.0126
```

The accuracy of the reported p-value is dubious, because some of the fitted cell means are at or near zero. With many LC models, the lack-of-fit test is at best a rough guide. Nevertheless, the result suggests that this model could be improved.

When computing residuals for an LC model, we must address the fact that the model of interest and the saturated model have different numbers of cells. First, we extract from the saturated model the frame containing the predicted true frequencies.

```
> satFrame <- get.fitted( fitSat, type="mean" )
> # this frame has 16 rows; display the first few
> head(satFrame)

    A   B   C   D freq        fit
1 neg neg neg neg  170 1.70000e+02
2 pos neg neg neg    4 4.00000e+00
```

```
3 neg pos neg neg    6 6.00000e+00
4 pos pos neg neg    1 1.00000e+00
5 neg neg pos neg    0 4.07402e-07
6 pos neg pos neg    0 4.07402e-07
```

```
> # get rid of the fitted values, because they are redundant
> satFrame$fit <- NULL
```

Next, we extract the fitted values from the LC model.

```
> LCFrame <-  get.fitted( fit, type="mean" )
> # this frame has 32 rows; display the first few
> head(LCFrame)
```

```
  L   A   B   C   D        freq         fit
1 1 neg neg neg neg 5.927605e-17 5.905489e-17
2 2 neg neg neg neg 1.700000e+02 1.693657e+02
3 1 pos neg neg neg 1.909359e-08 2.478698e-08
4 2 pos neg neg neg 4.000000e+00 5.192732e+00
5 1 neg pos neg neg 7.547378e-17 7.861742e-17
6 2 neg pos neg neg 6.000000e+00 6.249913e+00
```

To compare these fitted means to the predicted true frequencies, we must sum them over the levels of the latent variable and arrange them in the same order as the cells of the saturated model.

```
> muHatTable <- xtabs( fit ~ A + B + C + D, data=LCFrame )
> muHatFrame <- as.data.frame( muHatTable, responseName = "muHat" )
> # diplay the first few rows to make sure that the
> # cell order is correct
> head( muHatFrame)
```

```
   A   B   C   D       muHat
1 neg neg neg neg 1.693657e+02
2 pos neg neg neg 5.192732e+00
3 neg pos neg neg 6.249913e+00
4 pos pos neg neg 1.916216e-01
5 neg neg pos neg 5.087254e-07
6 pos neg pos neg 2.752312e-07
```

Finally, we compute the quasi-Pearson residuals, and put the fitted values and residuals into the frame that holds the predicted true frequencies.

```
> muHat <- muHatFrame$muHat
> quasiPearson <- ( satFrame$freq - muHat ) / sqrt( muHat )
> satFrame$muHat <- round( muHat, 3 )
> satFrame$quasiPearson <- round( quasiPearson, 2 )
> satFrame
```

```
    A   B   C   D freq   muHat quasiPearson
1  neg neg neg neg  170 169.366        0.05
2  pos neg neg neg    4   5.193       -0.52
3  neg pos neg neg    6   6.250       -0.10
```

```
4  pos pos neg neg    1   0.192         1.85
5  neg neg pos neg    0   0.000         0.00
6  pos neg pos neg    0   0.000         0.00
7  neg pos pos neg    0   0.000         0.00
8  pos pos pos neg    0   0.000         0.00
9  neg neg neg pos   15  14.837         0.04
10 pos neg neg pos   17   9.096         2.62
11 neg pos neg pos    0   0.548        -0.74
12 pos pos neg pos    4  11.520        -2.22
13 neg neg pos pos    0   0.000         0.00
14 pos neg pos pos   83  90.509        -0.79
15 neg pos pos pos    0   0.000         0.00
16 pos pos pos pos  128 120.491         0.68
```

The residuals in rows 10 and 12 are a bit large, suggesting again that the model can be improved.

8.5 Departures from local independence

One way to improve the fit of an LC model is to increase the number of latent classes. For this example, that is not really an option, because we believe HIV infection status is a binary condition, and because the tests `A`, `B`, `C` and `D` were designed for binary diagnosis. Fit may also be improved by relaxing the assumption of local independence. This should be done cautiously and sparingly, because LC models use up degrees of freedom very quickly. With binary items and a binary latent class, a three-way association such as `L:A:B` adds two more parameters to the model. In the code below, we add each of these three-way associations without the others and compare the fit to local independence and the saturated model.

```
> set.seed(85657)
> fitLAB <- cvam( ~ L*A + L*B + L*C + L*D + L*A*B,
+     data=hivtest, freq=COUNT,
+     control = list(startValJitter=.1) )

Note: Estimate at or near boundary
Estimated variances may be unreliable

> anova(fit, fitLAB, fitSat, pval=TRUE)

Model 1: ~ L * A + L * B + L * C + L * D
Model 2: ~ L * A + L * B + L * C + L * D + L * A * B
Model 3: ~ A * B * C * D
  resid.df -2*loglik df  change   pval
1        6   -3070.8
2        4   -3072.7  2  1.8402 0.3985
3        0   -3087.1  4 14.3871 0.0062

> fitLAC <- cvam( ~ L*A + L*B + L*C + L*D + L*A*C,
+     data=hivtest, freq=COUNT,
+     control = list(startValJitter=.1) )

Note: Estimate at or near boundary
Estimated variances may be unreliable
```

```
> anova(fit, fitLAC, fitSat, pval=TRUE)

Model 1: ~ L * A + L * B + L * C + L * D
Model 2: ~ L * A + L * B + L * C + L * D + L * A * C
Model 3: ~ A * B * C * D
  resid.df -2*loglik df change   pval
1        6   -3070.8
2        4   -3070.8  2  0.000 1.0000
3        0   -3087.1  4 16.227 0.0027

> fitLAD <- cvam( ~ L*A + L*B + L*C + L*D + L*A*D,
+     data=hivtest, freq=COUNT,
+     control = list(startValJitter=.1) )

Note: Estimate at or near boundary
Estimated variances may be unreliable

> anova(fit, fitLAD, fitSat, pval=TRUE)

Model 1: ~ L * A + L * B + L * C + L * D
Model 2: ~ L * A + L * B + L * C + L * D + L * A * D
Model 3: ~ A * B * C * D
  resid.df -2*loglik df change   pval
1        6   -3070.8
2        4   -3084.0  2 13.171 0.0014
3        0   -3087.1  4  3.056 0.5485

> fitLBC <- cvam( ~ L*A + L*B + L*C + L*D + L*B*C,
+     data=hivtest, freq=COUNT,
+     control = list(startValJitter=.1) )

Note: Estimate at or near boundary
Estimated variances may be unreliable

> anova(fit, fitLBC, fitSat, pval=TRUE)

Model 1: ~ L * A + L * B + L * C + L * D
Model 2: ~ L * A + L * B + L * C + L * D + L * B * C
Model 3: ~ A * B * C * D
  resid.df -2*loglik df change   pval
1        6   -3070.8
2        4   -3084.0  2 13.171 0.0014
3        0   -3087.1  4  3.056 0.5485

> fitLBD <- cvam( ~ L*A + L*B + L*C + L*D + L*B*D,
+     data=hivtest, freq=COUNT,
+     control = list(startValJitter=.1) )

Note: Estimate at or near boundary
Estimated variances may be unreliable

> anova(fit, fitLBD, fitSat, pval=TRUE)

Model 1: ~ L * A + L * B + L * C + L * D
Model 2: ~ L * A + L * B + L * C + L * D + L * B * D
Model 3: ~ A * B * C * D
  resid.df -2*loglik df  change   pval
1        6   -3070.8
2        4   -3072.0  2  1.1872 0.5523
3        0   -3087.1  4 15.0400 0.0046
```

```
> fitLCD <- cvam( ~ L*A + L*B + L*C + L*D + L*C*D,
+    data=hivtest, freq=COUNT,
+    control = list(startValJitter=.1) )

Note: Estimate at or near boundary
Estimated variances may be unreliable

> anova(fit, fitLCD, fitSat, pval=TRUE)

Model 1: ~ L * A + L * B + L * C + L * D
Model 2: ~ L * A + L * B + L * C + L * D + L * C * D
Model 3: ~ A * B * C * D
  resid.df -2*loglik df  change   pval
1        6   -3070.8
2        4   -3072.7  2  1.8402 0.3985
3        0   -3087.1  4 14.3871 0.0062
```

The associations `L:A:D` and `L:B:C` are both statistically significant. Adding either of them without the other produces a model that fits well, but adding both of them would be of no value.

```
> fitBoth <- cvam( ~ L*A + L*B + L*C + L*D + L*A*D + L*B*C,
+    data=hivtest, freq=COUNT,
+    control = list(startValJitter=.1) )

Note: Estimate at or near boundary
Estimated variances may be unreliable

> anova(fitLAD, fitBoth)

Model 1: ~ L * A + L * B + L * C + L * D + L * A * D
Model 2: ~ L * A + L * B + L * C + L * D + L * A * D + L * B * C
  resid.df -2*loglik df      change
1        4     -3084
2        2     -3084  2 -1.2725e-06

> anova(fitLBC, fitBoth)

Model 1: ~ L * A + L * B + L * C + L * D + L * B * C
Model 2: ~ L * A + L * B + L * C + L * D + L * A * D + L * B * C
  resid.df -2*loglik df    change
1        4     -3084
2        2     -3084  2 7.856e-07
```

8.6 Posterior predictions

With two plausible models and no compelling reason to choose one over the other, we arbitrarily pick the model with `L:B:C` and examine its implications. Using the function `cvamPredict`, we can obtain the estimated posterior probabilities of `L=1` and `L=2` for each row of the original dataset.

```
> # get predicted probabilities and display them with the dataset
> pred <- cvamPredict( ~L, fitLBC, data=hivtest )
> cbind( hivtest, round(pred, 3) )
```

```
    A   B   C   D COUNT    L     1     2
1 neg neg neg neg   170 <NA> 1.000 0.000
2 neg neg neg pos    15 <NA> 1.000 0.000
3 neg pos neg neg     6 <NA> 1.000 0.000
4 pos neg neg neg     4 <NA> 1.000 0.000
5 pos neg neg pos    17 <NA> 0.024 0.976
6 pos neg pos pos    83 <NA> 0.000 1.000
7 pos pos neg neg     1 <NA> 1.000 0.000
8 pos pos neg pos     4 <NA> 0.004 0.996
9 pos pos pos pos   128 <NA> 0.000 1.000
```

Notice that in this fitted model, the labels for the latent classes have switched, so L=1 now represents HIV-negative and L=2 represents HIV-positive. These predictions apply to the patients in this study, each of whom was given all four tests. In actual medical practice, it is more likely for a patient to receive one of the four tests. In the example below, we create a new prediction frame that shows the posterior probabilities given a positive or negative result for each test apart from the others.

```
> predFrame <- hivtest[1:8,]
> predFrame$COUNT <- NULL
> predFrame[["A"]][] <- NA
> predFrame[["B"]][] <- NA
> predFrame[["C"]][] <- NA
> predFrame[["D"]][] <- NA
> predFrame[["A"]][1] <- "pos"; predFrame[["A"]][2] <- "neg"
> predFrame[["B"]][3] <- "pos"; predFrame[["B"]][4] <- "neg"
> predFrame[["C"]][5] <- "pos"; predFrame[["C"]][6] <- "neg"
> predFrame[["D"]][7] <- "pos"; predFrame[["D"]][8] <- "neg"
> predFrame[["A"]] <- coarsened( predFrame[["A"]] )
> predFrame[["B"]] <- coarsened( predFrame[["B"]] )
> predFrame[["C"]] <- coarsened( predFrame[["C"]] )
> predFrame[["D"]] <- coarsened( predFrame[["D"]] )
> pred <- cvamPredict( ~L, fitLBC, data=predFrame )
> cbind( predFrame, round(pred, 3) )
```

```
     A    B    C    D    L     1     2
1  pos <NA> <NA> <NA> <NA> 0.023 0.977
2  neg <NA> <NA> <NA> <NA> 1.000 0.000
3 <NA>  pos <NA> <NA> <NA> 0.050 0.950
4 <NA>  neg <NA> <NA> <NA> 0.655 0.345
5 <NA> <NA>  pos <NA> <NA> 0.000 1.000
6 <NA> <NA>  neg <NA> <NA> 0.905 0.095
7 <NA> <NA> <NA>  pos <NA> 0.062 0.938
8 <NA> <NA> <NA>  neg <NA> 1.000 0.000
```

And in the code below, we produce posterior predictions under the model of local independence.

```
> pred <- cvamPredict( ~L, fit, data=predFrame )
> cbind( predFrame, round(pred, 3) )
```

```
     A    B    C    D    L     1     2
1  pos <NA> <NA> <NA> <NA> 0.975 0.025
2  neg <NA> <NA> <NA> <NA> 0.000 1.000
3 <NA>  pos <NA> <NA> <NA> 0.950 0.050
4 <NA>  neg <NA> <NA> <NA> 0.343 0.657
5 <NA> <NA>  pos <NA> <NA> 1.000 0.000
6 <NA> <NA>  neg <NA> <NA> 0.093 0.907
7 <NA> <NA> <NA>  pos <NA> 0.936 0.064
8 <NA> <NA> <NA>  neg <NA> 0.000 1.000
```

Aside from the fact that the class labels are different, the predicted values from the two models are essentially identical. Including the `L:B:C` association produces a better fitting model but does not change the diagnostic implications.

8.7 MCMC for latent-class models

Running MCMC on an LC model is straightforward, but first we must address the fact that the ML estimates lie on a boundary. Adding a small bit of prior information through a ridge factor solves the problem.

```
> # re-fit the model with EM using a small ridge factor
> set.seed(7666)
> fitLBC <- cvam( ~ L*A + L*B + L*C + L*D + L*B*C,
+    data=hivtest, freq=COUNT, prior=cvamPrior( ridge=.1 ),
+    control = list(startValJitter=.1) )
```

The default DA procedure gets stuck, but with a little experimentation, we found that RWM with tuning parameters `c(1000,.5)` works well for this problem.

```
> # do a long run of MCMC and save ten imputed datasets
> fitMCMC <- cvam(fitLBC, method="MCMC",
+    control=list( typeMCMC="RWM", tuneRWM=c(1000,.5),
+       iterMCMC=25000, imputeEvery=2500 ) )
```

It is possible for the latent-class labels to permute during an MCMC run, which complicates the task of extracting posterior summaries of parameters from the output stream. Many solutions to the label-switching problem have been proposed (Richardson and Green, 1997; Celeux et al., 2000; Chung et al., 2004; Papastamoulis, 2014). Here we suggest a pragmatic and simple method: instead of working with the parameter series, collect and analyze multiple imputations of the complete-data table. It is easy to examine a few imputed datasets to see if label switching has occurred, and if it has, the problem can be solved by a simple relabeling of the latent factor in the affected datasets.

```
> # check to see if any label switching has occurred
> impData <- get.imputedFreq(fitMCMC)
> head(impData)

  L   A   B   C   D imp.1 imp.2 imp.3 imp.4 imp.5 imp.6 imp.7 imp.8 imp.9
1 1 neg neg neg neg     0     0     0     0     0     0     0     0     0
2 2 neg neg neg neg   170   170   170   170   170   170   170   170   170
3 1 pos neg neg neg     0     0     0     0     0     0     0     0     0
4 2 pos neg neg neg     4     4     4     4     4     4     4     4     4
5 1 neg pos neg neg     0     0     0     0     0     0     0     0     0
6 2 neg pos neg neg     6     6     6     6     6     6     6     6     6
  imp.10
1      0
2    170
3      0
4      4
5      0
6      6
```

In each of these imputed tables, the 170 patients with negative results on all four tests were all assigned to class L=2. If label switching had occurred, we would have occasionally seen them assigned to L=1.

With these imputed datasets, we can see why the model of local independence did not fit. Taking the first imputation and collapsing it down to the B $\times$ C $\times$ L margins, we examine the conditional B $\times$ C tables for L=1 and L=2.

```
> impData$freq <- impData[["imp.1"]] # first imputation
> BCL <- xtabs( freq ~ B + C + L, data=impData )
> BCL

, , L = 1

     C
B     neg pos
  neg  17  83
  pos   4 128

, , L = 2

     C
B     neg pos
  neg 189   0
  pos   7   0
```

The results from tests B and C are strongly correlated for the L=1 group, which now represents patients who are HIV-positive. For the HIV-negative patients (L=2), the data are not sufficient to compute an odds ratio. Using the formulas from Section 7.3, we compute the conditional log-odds ratios and their standard errors from each imputed dataset, then combine the results. To avoid problems due to zero cells, we add $1/2$ to each cell before computing the odds ratio.

```
> # use multiple imputations to examine the conditional
> # BC odds ratios given L=1 and L=2
```

```
> est.list <- SE.list <- as.list(1:10)
> for( m in 1:10 ) {
+    # get the imputed marginal table BxCxL
+    impName <- paste( "imp", format(m), sep="." )
+    impData$freq <- impData[[impName]]
+    BCL <- xtabs( freq ~ B + C + L, data=impData )
+    # add 1/2 to every cell to avoid problems
+    BCL <- BCL + .5
+    # get BC log-odds ratio and SE for L=1
+    BCL.1 <- BCL[,,"1"]
+    logOR.1 <- log( ( BCL.1[1,1] * BCL.1[2,2] ) /
+       ( BCL.1[1,2] * BCL.1[2,1] ) )
+    SE.1 <- sqrt( sum( 1/BCL.1 ) )
+    # get BC log-odds ratio and SE for L=2
+    BCL.2 <- BCL[,,"2"]
+    logOR.2 <- log( ( BCL.2[1,1] * BCL.2[2,2] ) /
+       ( BCL.2[1,2] * BCL.2[2,1] ) )
+    SE.2 <- sqrt( sum( 1/BCL.2 ) )
+    # save the estimates and SEs
+    est.list[[m]] <- c( logOR.1, logOR.2 )
+    SE.list[[m]] <- c( SE.1, SE.2 )
+ }
> miInference( est.list, SE.list )

       Est      SE Est/SE         df     p Pct.mis
[1,] 1.7849 0.55982  3.188    23011.2 0.001       2
[2,] 3.2196 2.03440  1.583 56542162.4 0.114       0
```

References

Agresti, A. (2013). *Categorical Data Analysis, Third Edition*. John Wiley & Sons, Hoboken, NJ.

Baker, S. G. (1994). The multinomial-Poisson transformation. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 43(4):495–504.

Bandeen-Roche, K., Miglioretti, D. L., Zeger, S. L., and Rathouz, P. J. (1997). Latent variable regression for multiple discrete outcomes. *Journal of the American Statistical Association*, 92(440):1375–1386.

Barnard, J. and Rubin, D. B. (1999). Miscellanea: Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86(4):948–955.

Bedrick, E. J., Christensen, R., and Johnson, W. (1996). A new perspective on priors for generalized linear models. *Journal of the American Statistical Association*, 91(436):1450–1460.

Bickel, P., Hammel, E., and O'Connell, J. (1975). Sex bias in graduate admissions: Data from Berkeley. *Science*, 187(4175):398–404.

Biemer, P. P., Woltmann, H., Raglin, D., and Hill, J. (2001). Enumeration accuracy in a population census: An evaluation using latent class analysis. *Journal of Official Statistics*, 17(1):129.

Bishop, Y. M., Fienberg, S. E., and Holland, P. W. (1975). *Discrete Multivariate Analysis: Theory and Practice*. MIT Press, Cambridge, MA.

Celeux, G., Hurn, M., and Robert, C. P. (2000). Computational and inferential difficulties with mixture posterior distributions. *Journal of the American Statistical Association*, 95(451):957–970.

Christensen, R. (2006). *Log-Linear Models and Logistic Regression*. Springer Science & Business Media, New York.

Chung, H., Loken, E., and Schafer, J. L. (2004). Difficulties in drawing inferences with finite-mixture models: a simple example with a simple solution. *The American Statistician*, 58(2):152–158.

Clogg, C. C. and Goodman, L. A. (1984). Latent structure analysis of a set of multidimensional contingency tables. *Journal of the American Statistical Association*, 79(388):762–771.

Clogg, C. C., Rubin, D. B., Schenker, N., Schultz, B., and Weidman, L. (1991). Multiple imputation of industry and occupation codes in census public-use samples using Bayesian logistic regression. *Journal of the American Statistical Association*, 86(413):68–78.

Collins, L. M. and Lanza, S. T. (2010). *Latent Class and Latent Transition Analysis: With Applications in the Social, Behavioral, and Health Sciences*, volume 718. John Wiley & Sons, New York.

Efron, B. (2012). Bayesian inference and the parametric bootstrap. *The Annals of Applied Statistics*, 6(4):1971.

Formann, A. K. and Kohlmann, T. (1996). Latent class analysis in medical research. *Statistical Methods in Medical Research*, 5(2):179–211.

Fuchs, C. (1982). Maximum likelihood estimation and model selection in contingency tables with missing data. *Journal of the American Statistical Association*, 77(378):270–278.

Gamerman, D. (1997). Sampling from the posterior distribution in generalized linear mixed models. *Statistics and Computing*, 7(1):57–68.

Gamerman, D. and Lopes, H. F. (2006). *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman & Hall/CRC Press, Boca Raton, FL.

Gelman, A., Stern, H. S., Carlin, J. B., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian Data Analysis, Third Edition*. Chapman & Hall/CRC Press, Boca Raton, FL.

Hagenaars, J. A. (1993). *Loglinear models with latent variables*. Number 94. Sage Publications, Thousand Oaks, CA.

Heitjan, D. F. and Rubin, D. B. (1991). Ignorability and coarse data. *The Annals of Statistics*, 19(4):2244–2253.

Jeffreys, H. (1961). *The Theory of Probability, Third Edition*. Oxford University Press, Oxford, UK.

Kadane, J. B. (1985). Is victimization chronic? a Bayesian analysis of multinomial missing data. *Journal of Econometrics*, 29(1-2):47–67.

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795.

Lanza, S. T., Dziak, J. J., Huang, L., Wagner, A., and Collins, L. M. (2015). *Proc LCA and Proc LTA Users' Guide, Version 1.3.2*. The Pennsylvania State University, University Park, PA.

Lauritzen, S. L. (1996). *Graphical Models*. Clarendon Press, Oxford.

Lazarsfeld, P. F. and Henry, N. W. (1968). *Latent Structure Analysis*. Houghton Mifflin Co.

Linzer, D. A. and Lewis, J. B. (2011). polca: An r package for polytomous variable latent class analysis. *Journal of Statistical Software*, 42(10):1–29.

Little, R. J. and Rubin, D. B. (2002). *Statistical Analysis with Missing Data, Second Edition*. John Wiley & Sons, New York.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models, Second edition*. Chapman & Hall/CRC Press, London.

Muthén, L. K. and Muthén, B. O. (2017). *Mplus User's Guide, Version 8*. Muthén & Muthén, Los Angeles, CA.

Papastamoulis, P. (2014). Handling the label switching problem in latent class models via the ECR algorithm. *Communications in Statistics: Simulation and Computation*, 43(4):913–927.

Pitt, M., Chan, D., and Kohn, R. (2006). Efficient Bayesian inference for Gaussian copula regression models. *Biometrika*, 93(3):537–554.

Plummer, M., Best, N., Cowles, K., and Vines, K. (2006). coda: Convergence diagnosis and output analysis for mcmc. *R News*, 6(1):7–11.

Raghunathan, T. E., Reiter, J. P., and Rubin, D. B. (2003). Multiple imputation for statistical disclosure limitation. *Journal of official statistics*, 19(1):1.

Reiter, J. P. (2004). Simultaneous use of multiple imputation for missing data and disclosure limitation. *Survey Methodology*, 30(2):235–242.

Richards, F. S. (1961). A method of maximum-likelihood estimation. *Journal of the Royal Statistical Society: Series B (Methodological)*, 23(2):469–475.

Richardson, S. and Green, P. J. (1997). On Bayesian analysis of mixtures with an unknown number of components (with discussion). *Journal of the Royal Statistical Society, Series B: Statistical Methodology*, 59(4):731–792.

Rubin, D. B. (1984). Bayesianly justifiable and relevant frequency calculations for the applies statistician. *The Annals of Statistics*, pages 1151–1172.

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys*. John Wiley & Sons, New York.

Rubin, D. B. (1996). Multiple imputation after 18+ years. *Journal of the American statistical Association*, 91(434):473–489.

Schafer, J. L. (1997). *Analysis of Incomplete Multivariate Data*. Chapman & Hall/CRC press, London.

Smith, T. W., Davern, M., Freese, J., and Morgan, S. L. (2019). *General Social Surveys, 1972–2018*. National Data Program for the Social Sciences, No. 25. NORC, Chicago. 1 data file (64,814 logical records) + 1 codebook (3,758 pp.).

Venables, W. N. and Ripley, B. D. (2013). *Modern Applied Statistics with S*. Springer Science & Business Media, New York, fourth edition.

Vermunt, J. K. and Magidson, J. (2016). *Technical Guide for Latent GOLD 5.1: Basic, Advanced, and Syntax*. Statistical Innovations, Inc., Belmont, MA.

Whittaker, J. (2009). *Graphical Models in Applied Multivariate Statistics*. John Wiley & Sons, New York.

Wickham, H. (2007). Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20.

Yang, I. and Becker, M. P. (1997). Latent variable modeling of diagnostic accuracy. *Biometrics*, pages 948–958.

Appendix A Notation for multivariate categorical data

Notation for describing multivariate categorical models can be complicated, because the data can be expressed in so many different forms; here we strive for brevity and generality. Scalars will be written in lightface, vectors and other arrays in boldface. Let $\boldsymbol{V}_i = (V_{i1}, \ldots, V_{iJ})$ denote a vector of J categorical random variables for sample unit i, where $i = 1, \ldots, N$ indexes units in the microdata sense, and N is the total sample size. The vectors $\boldsymbol{V}_1, \ldots, \boldsymbol{V}_N$ are not necessarily seen, because data may arrive in a tabulated or grouped formats, and even when microdata are given, some of the V_{ij}'s could be missing or coarsened. The set of possible values taken by V_{ij} is

$$\mathbb{V}_j \;=\; \{1, 2, \ldots, \#\mathbb{V}_j\}.$$

(The symbol '#' is the cardinality operator. When applied to a set, it returns the number of elements in the set.) Let $\boldsymbol{v} = (v_1, \ldots, v_J)$ denote a possible value of $\boldsymbol{V}_i$. For now, we suppose that $\boldsymbol{v}$ lies within the set

$$\mathbb{V} \;=\; \mathbb{V}_1 \;\times\; \mathbb{V}_2 \;\times\; \cdots \;\times\; \mathbb{V}_J \;=\; \prod_{j=1}^{J} \mathbb{V}_j,$$

where '$\times$' and '$\prod$' denote the Cartesian product. In Appendix C, we will discuss the possibility of structural zeros, cells within $\mathbb{V}$ that are disallowed and whose probabilities are fixed at zero. Denote the probability of the event $\boldsymbol{V}_i \;=\; \boldsymbol{v}$ by

$$\pi_{\boldsymbol{v}} \;=\; P(V_{i1} = v_1, \ldots, V_{iJ} = v_J),$$

and the vector of probabilities for all cells by

$$\boldsymbol{\pi} \;=\; (\pi_{\boldsymbol{v}} \;:\; \boldsymbol{v} \in \mathbb{V}).$$

We assume that $\boldsymbol{\pi}$ vector lies within the $(\#\mathbb{V} - 1)$-dimensional open simplex

$$\mathbb{S} \;=\; \left\{ \boldsymbol{\pi} \;:\; 0 < \pi_{\boldsymbol{v}} < 1 \,, \boldsymbol{v} \in \mathbb{V} \cap \sum_{\boldsymbol{v} \in \mathbb{V}} \pi_{\boldsymbol{v}} = 1 \right\}.$$

Under multinomial sampling, in which units are independently drawn from a common population, the loglikelihood function for $\boldsymbol{\pi}$ based on $\boldsymbol{V}_1, \ldots, \boldsymbol{V}_N$ is

$$l_\pi(\boldsymbol{\pi} \,|\, \boldsymbol{V}_1, \ldots, \boldsymbol{V}_N) \;=\; c \;+\; \sum_{i=1}^{N} \log P(\boldsymbol{V}_i \,|\, \boldsymbol{\pi}),$$

where c is an arbitrary constant. But

$$\begin{aligned} l_\pi(\boldsymbol{\pi} \mid \boldsymbol{V}_1, \ldots, \boldsymbol{V}_N) &= \sum_{i=1}^{N} \sum_{\boldsymbol{v} \in \mathbb{V}} I(\boldsymbol{V}_i = \boldsymbol{v}) \log \pi_{\boldsymbol{v}} \\ &= \sum_{\boldsymbol{v} \in \mathbb{V}} f_{\boldsymbol{v}} \log \pi_{\boldsymbol{v}}, \end{aligned} \tag{10}$$

where $f_{\boldsymbol{v}} = \sum_{i=1}^{N} I(\boldsymbol{V}_i = \boldsymbol{v})$ is the frequency in cell $\boldsymbol{v}$, and $I(\cdot)$ is the indicator function equal to one if its argument is true and zero otherwise. Because this function depends on $\boldsymbol{V}_1, \ldots, \boldsymbol{V}_N$ only through the sufficient statistic $\boldsymbol{f} = (f_{\boldsymbol{v}} : \boldsymbol{v} \in \mathbb{V})$, we will write it as $l_\pi(\boldsymbol{\pi} \mid \boldsymbol{f})$. We will also call it the complete-data multinomial loglikelihood, because it is the function we would use to estimate $\boldsymbol{\pi}$ under multinomial sampling if $\boldsymbol{f}$ were fully observed. Without further restrictions on $\boldsymbol{\pi}$, and without any missing or coarsened values, the maximum-likelihood (ML) estimate for $\boldsymbol{\pi}$ under multinomial sampling given by the sample proportions

$$\underset{\boldsymbol{\pi} \in \mathbb{S}}{\arg\max}\ l_\pi(\boldsymbol{\pi} \mid \boldsymbol{f}) = N^{-1} \boldsymbol{f} \tag{11}$$

(Agresti, 2013).

Appendix B Alternative sampling distributions

Appendix B.1 Poisson sampling

Under multinomial sampling, N is regarded as fixed, $\boldsymbol{f}$ has a multinomial distribution

$$\boldsymbol{f} \mid N, \boldsymbol{\pi} \sim \text{Mult}(N, \boldsymbol{\pi}), \tag{12}$$

and the elements of $\boldsymbol{f}$ are negatively intercorrelated because of the constraint $f_{\boldsymbol{+}} = \sum_{\boldsymbol{v} \in \mathbb{V}} f_{\boldsymbol{v}} = N$. (Whenever a vector subscript is replaced by '+', it denotes summation over the subscript.) Even when N is fixed by the study design, it may be convenient to treat the elements of $\boldsymbol{f}$ as independent Poisson variates with means $\boldsymbol{\mu} = (\mu_{\boldsymbol{v}} : \boldsymbol{v} \in \mathbb{V})$,

$$f_{\boldsymbol{v}} \mid \boldsymbol{\mu} \sim \text{Poisson}(\mu_{\boldsymbol{v}}) \text{ independently for } \boldsymbol{v} \in \mathbb{V}, \tag{13}$$

which makes N a random variable. Using a well known relationship between the Poisson and multinomial distributions, the joint distribution function for $\boldsymbol{f}$ implied by Equation (13) can be factored as

- a Poisson distribution function for N, with mean $\phi = \mu_{\boldsymbol{+}} = \sum_{\boldsymbol{v} \in \mathbb{V}} \mu_{\boldsymbol{v}}$, multiplied by

- the multinomial distribution function implied by Equation (12), with $\boldsymbol{\pi} = \phi^{-1}\boldsymbol{\mu}$

(Agresti, 2013). Consequently, the Poisson-induced loglikelihood function for $\boldsymbol{\mu}$,

$$l_\mu(\boldsymbol{\mu} \mid \boldsymbol{f}) \;=\; \sum_{\boldsymbol{v}\in\mathbb{V}} \left(f_{\boldsymbol{v}} \log \mu_{\boldsymbol{v}} \;-\; \mu_{\boldsymbol{v}} \right), \tag{14}$$

can be written as

$$l_\mu(\phi\boldsymbol{\pi} \mid \boldsymbol{f}) \;=\; l_\phi(\phi \mid N) \;+\; l_\pi(\boldsymbol{\pi} \mid \boldsymbol{f}), \tag{15}$$

where $l_\phi(\phi \mid N) \;=\; N \log \phi \;-\; \phi$ and $l_\pi(\boldsymbol{\pi} \mid \boldsymbol{f})$ is the multinomial loglikelihood. At any fixed value of ϕ, $l_\pi(\boldsymbol{\pi} \mid \boldsymbol{f})$ and $l_\mu(\phi\boldsymbol{\pi} \mid \boldsymbol{f})$ differ by an additive constant. It follows that an ML estimate for $\boldsymbol{\pi}$ based on the multinomial model can be obtained by computing the ML estimate for $\boldsymbol{\mu}$ based on the Poisson model, fixing ϕ at its estimated value, and projecting the estimated $\boldsymbol{\mu}$ into $\mathbb{S}$ by $\boldsymbol{\pi} \;=\; \phi^{-1}\boldsymbol{\mu}$ (Richards, 1961; Baker, 1994). The nuisance quantity ϕ is an expansion parameter which, after it has been used to convert $\boldsymbol{\mu}$ to $\boldsymbol{\pi}$, carries no further information about $\boldsymbol{\pi}$ from a likelihood perspective. Estimating multinomial probabilities by fitting a surrogate Poisson model, which is sometimes called the Poisson trick, is often used in categorical data analysis to circumvent the sum-to-one constraint on $\boldsymbol{\pi}$ (Venables and Ripley, 2013).

When using the Poisson trick, the parameter spaces for $\boldsymbol{\mu}$ and $\boldsymbol{\pi}$ must conform in the following sense: if we assume $\boldsymbol{\pi} \in \mathbb{S}_0 \subset \mathbb{S}$, then the surrogate Poisson model must be fit over the augmented space

$$\mathbb{M}_0 \;=\; \{\boldsymbol{\mu} : \boldsymbol{\mu} = \phi\boldsymbol{\pi}, \boldsymbol{\pi} \in \mathbb{S}_0, \phi \in (0, +\infty)\}.$$

This condition, which is automatically satisfied for log-linear models, ensures that maximizing l_μ is equivalent to separately maximizing l_ϕ and l_π. In particular, when no restrictions are placed on $\boldsymbol{\pi}$ other than $\pi_+ = 1$, the ML estimate for μ is

$$\underset{\boldsymbol{\mu}\,\in\,\mathbb{M}_0}{\arg\max}\; l_\mu(\boldsymbol{\mu} \mid \boldsymbol{f}) \;=\; \boldsymbol{f}, \tag{16}$$

in agreement with Equation (11).

Although the Poisson trick is usually described in terms of ML estimation, the equivalence between $l_\pi(\boldsymbol{\pi} \mid \boldsymbol{f})$ and $l_\mu(\phi\boldsymbol{\pi} \mid \boldsymbol{f})$ holds at any fixed value for ϕ, not just at the mode. In Bayesian analyses, inferences about $\boldsymbol{\pi}$ under Poisson and multinomial models will be equivalent, provided that the prior distribution for $\boldsymbol{\mu}$, when marginalized over ϕ, is the desired prior for $\boldsymbol{\pi}$.

When using the Poisson trick, it is crucial to include empty cells, because they are informative in the Poisson setting. A cell with $f_{\boldsymbol{v}} = 0$ contributes nothing to l_π,

so when fitting a multinomial model to grouped data, rows of a data frame with zero frequencies may be omitted. But an occurrence of $f_{\boldsymbol{v}} = 0$ does contribute to l_μ, so a surrogate Poisson model must be fit to the full contingency table, including any empty cells.

Appendix B.2 Product-multinomial sampling

The Poisson trick can be extended to situations where some categorical variables are regarded as fixed. Suppose we partition the variables as $\boldsymbol{V}_i = (\boldsymbol{A}_i, \boldsymbol{B}_i)$, where $\boldsymbol{A}_i = (V_{i1}, \ldots, V_{ij})$ and $\boldsymbol{B}_i = (V_{i,j+1}, \ldots, V_{iJ})$ for some j, and we are only interested in describing the conditional distribution of $\boldsymbol{B}_i$ given $\boldsymbol{A}_i$, treating $\boldsymbol{A}_i$ as unmodeled covariates. Partition the microdata as $\boldsymbol{V} = (\boldsymbol{A}, \boldsymbol{B})$, where $\boldsymbol{A} = (\boldsymbol{A}_i : i = 1, \ldots, N)$ and $\boldsymbol{B} = (\boldsymbol{B}_i : i = 1, \ldots, N)$. Denote possible values for $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ as $\boldsymbol{a} = (v_1, \ldots, v_j) \in \mathbb{A}$ and $\boldsymbol{b} = (v_{j+1}, \ldots, v_J) \in \mathbb{B}$, where $\mathbb{A} = \mathbb{V}_1 \times \cdots \times \mathbb{V}_j$ and $\mathbb{B} = \mathbb{V}_{j+1} \times \cdots \times \mathbb{V}_J$, so that $\pi_{\boldsymbol{v}} = \pi_{\boldsymbol{a},\boldsymbol{b}}$ and $f_{\boldsymbol{v}} = f_{\boldsymbol{a},\boldsymbol{b}}$. Denote the marginal probability of $\boldsymbol{A}_i = \boldsymbol{a}$ by

$$\pi^{(A)}_{\boldsymbol{a}} = \sum_{\boldsymbol{b} \in \mathbb{B}} f_{\boldsymbol{a},\boldsymbol{b}},$$

and the vector of all such marginal probabilities by $\boldsymbol{\pi}^{(A)} = (\pi^{(A)}_{\boldsymbol{a}} : \boldsymbol{a} \in \mathbb{A})$. Similarly, denote the marginal frequency of $\boldsymbol{A}_i = \boldsymbol{a}$ by

$$f^{(A)}_{\boldsymbol{a}} = \sum_{\boldsymbol{b} \in \mathbb{B}} f_{\boldsymbol{a},\boldsymbol{b}},$$

and let $\boldsymbol{f}^{(A)} = (f^{(A)}_{\boldsymbol{a}} : \boldsymbol{a} \in \mathbb{A})$. Finally, let

$$\boldsymbol{f}_{\boldsymbol{a},:} = (f_{\boldsymbol{a},\boldsymbol{b}} : \boldsymbol{b} \in \mathbb{B})$$

denote the slice of the frequency table $\boldsymbol{f}$ corresponding to $\boldsymbol{A}_i = \boldsymbol{a}$. The multinomial sampling model of Equation (12) implies that

$$\boldsymbol{f}^{(A)} \,|\, N, \boldsymbol{\pi} \sim \text{Mult}(N, \boldsymbol{\pi}^{(A)}), \tag{17}$$

and that

$$\boldsymbol{f}_{\boldsymbol{a},:} \,|\, \boldsymbol{f}^{(A)}, \boldsymbol{\pi} \sim \text{Mult}(f^{(A)}_{\boldsymbol{a}}, \boldsymbol{\pi}^{(B\,|\,A)}_{\boldsymbol{a},:}) \text{ independently for } \boldsymbol{a} \in \mathbb{A}, \tag{18}$$

where $\pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b}} = \pi_{\boldsymbol{a},\boldsymbol{b}} \,/\, \pi^{(A)}_{\boldsymbol{a}}$ denotes the conditional probability of $\boldsymbol{B}_i = \boldsymbol{b}$ given $\boldsymbol{A}_i = \boldsymbol{a}$, and

$$\boldsymbol{\pi}^{(B\,|\,A)}_{\boldsymbol{a},:} = (\pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b}} : \boldsymbol{b} \in \mathbb{B})$$

denotes the slice of all such probabilities corresponding to $\boldsymbol{A}_i = \boldsymbol{a}$. A set of independent multinomial distributions over slices of a contingency table is called a product-multinomial model.

It follows that the Poisson loglikelihood in Equation (15) can be written as

- a Poisson loglikelihood for ϕ based on $N =$, plus
- a multinomial loglikelihood for the marginal probabilities $\boldsymbol{\pi}^{(A)}$ based on the marginal frequencies $\boldsymbol{f}^{(A)}$, plus
- a multinomial loglikelihood for each slice of conditional probabilities $\boldsymbol{\pi}^{(B\,|\,A)}_{\boldsymbol{a},:}$ based on the corresponding slice of frequencies $\boldsymbol{f}_{\boldsymbol{a},:}$.

If we want to model only the conditional distribution of $\boldsymbol{B}_i$ given $\boldsymbol{A}_i$, because the marginal frequencies $\boldsymbol{f}^{(A)}$ are fixed by design or are otherwise not of interest, we can do so in three different ways.

- Fit separate multinomial models to each slice $f^{(A)}_{\boldsymbol{a}}$ to directly estimate each slice of conditional probabilities $\boldsymbol{\pi}^{(B\,|\,A)}_{\boldsymbol{a},:}$.
- Fit a surrogate multinomial model to $\boldsymbol{f}$, compute the conditonal probabilities $\boldsymbol{\pi}^{(B\,|\,A)}_{\boldsymbol{a},:}$ from the estimate of $\boldsymbol{\pi}$, and ignore the expansion parameters $\boldsymbol{\pi}^{(A)}$.
- Fit a surrogate Poisson model to $\boldsymbol{f}$, compute the conditonal probabilities $\boldsymbol{\pi}^{(B\,|\,A)}_{\boldsymbol{a},:}$ from the estimate of $\boldsymbol{\mu}$, and ignore the expansion parameters ϕ and $\boldsymbol{\pi}^{(A)}$.

Answers from the three methods will be identical, provided that the surrogate models impose no restrictions on ϕ other than $\phi \in (0, +\infty)$, and no restrictions on $\boldsymbol{\pi}^{(A)}$ other than $\sum_{\boldsymbol{a} \in \mathbb{A}} \pi^{(A)}_{\boldsymbol{a}} = 1$. In practice, this can be satisfied by fitting a surrogate model that includes all possible associations among the variables in $\boldsymbol{A}_i$ (Venables and Ripley, 2013).

Appendix C Defining a log-linear model

For our purposes, a log-linear model is a restriction on $\boldsymbol{\pi}$ of the form

$$\log \boldsymbol{\pi} = \boldsymbol{o} + \boldsymbol{X}\boldsymbol{\lambda}, \tag{19}$$

where $\boldsymbol{X}$ is a known $\#\mathbb{V} \times p$ model matrix, $\boldsymbol{o} = (o_1, o_2, \ldots)^\top$ is a known vector of length $\#\mathbb{V}$, and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)^\top$ is a vector of unknown coefficients to be estimated (the superscript '$\top$' denotes transpose). The $\boldsymbol{o}$ vector is called an offset,

and it appears in applications where the expected cell frequencies are thought to be proportional to a given variable (called exposure) that varies across cells. Except in those settings, $\boldsymbol{o}$ is usually set to $\mathbf{0} = (0, \ldots, 0)^\top$. We assume that $\boldsymbol{X}$ has full rank. We also require $\mathbb{C}(\boldsymbol{X})$, the linear space spanned by the columns of $\boldsymbol{X}$, to include $\mathbf{1} = (1, 1, \ldots,)^\top$. If $\mathbf{1}$ were not in $\mathbb{C}(\boldsymbol{X})$, then it might be impossible to find a $\boldsymbol{\lambda}$ for which $\pi_+ = 1$. Following standard practice, we will usually satisfy this requirement by choosing an $\boldsymbol{X}$ with $\mathbf{1}$ as its first column. The remaining columns of $\boldsymbol{X}$ will correspond to terms for main effects of $V_{i1}, \ldots, V_{iJ}$ and interactions among them. The log-linear model for $\boldsymbol{\mu} = N\boldsymbol{\pi}$ implied by Equation (19) is

$$\log \boldsymbol{\mu} \;=\; \boldsymbol{o} \;+\; \boldsymbol{X}\boldsymbol{\beta}. \tag{20}$$

In the computations performed by `cvam`, the ordering of the cells in the log-linear model is not specified directly by the user. Rather, it is determined by R's formula mechanism and the anti-lexicographical ordering of records in a data frame generated by the `aggregate` function.

To make sure that the logarithms in (19) and (20) are defined, we require $\pi_{\boldsymbol{v}} > 0$ and $\mu_{\boldsymbol{v}} > 0$ for every cell. However, we do allow for the possibility of structural zeros, particular values of $\boldsymbol{v} = (v_1, \ldots, v_J)$ that are deemed by the user to be impossible. Structural zero cells are not removed from $\boldsymbol{\pi}$ or $\boldsymbol{\mu}$ or from the rows of $\boldsymbol{X}$, but they are skipped over in all computations that iterate over the cells of the table. For example, if structural zeros are present, then every sum over $\boldsymbol{v} \in \mathbb{V}$ in any formula should be understood as as summation over the cells that are not structural zeros.

Appendix D Newton-Raphson with complete data

Under the Poisson surrogate model defined by (13) and (20), we maximize the log-likelihood over the expanded parameter space $\boldsymbol{\beta} \in \mathbb{R}^p$ using the Newton-Raphson (NR) method. Writing the Poisson-induced loglikelihood as

$$l(\boldsymbol{\beta}) \;=\; \sum_{\boldsymbol{v} \in \mathbb{V}} (f_{\boldsymbol{v}} \log \mu_{\boldsymbol{v}} \;-\; \mu_{\boldsymbol{v}}),$$

one iteration of NR updates the current estimate $\hat{\boldsymbol{\beta}}^{(t)}$ by

$$\hat{\boldsymbol{\beta}}^{(t+1)} \;=\; \hat{\boldsymbol{\beta}}^{(t)} + \;=\; \left[- \left(\frac{\partial^2 l}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top} \right) \right]^{-1} \left(\frac{\partial l}{\partial \boldsymbol{\beta}} \right),$$

where the derivatives on the right-hand side are evaluated at $\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}^{(t)}$. Letting $\eta_{\boldsymbol{v}} = \log \mu_{\boldsymbol{v}}$ and applying the chain rule

$$\frac{\partial l}{\partial \beta_j} \;=\; \sum_{\boldsymbol{v} \in \mathbb{V}} \left(\frac{\partial l}{\partial \mu_{\boldsymbol{v}}} \right) \left(\frac{\partial \mu_{\boldsymbol{v}}}{\partial \eta_{\boldsymbol{v}}} \right) \left(\frac{\partial \eta_{\boldsymbol{v}}}{\partial \beta_j} \right),$$

we obtain

$$\frac{\partial l}{\partial \beta_j} \;=\; \sum_{\boldsymbol{v}\in\mathbb{V}} (f_{\boldsymbol{v}} - \mu_{\boldsymbol{v}})\, x_{\boldsymbol{v},j},$$

where $x_{\boldsymbol{v},j}$ denotes the jth element of the row of the model matrix corresponding to cell $\boldsymbol{v}$. Applying the chain rule again, we get

$$\frac{\partial^2 l}{\partial \beta_j \partial \beta_k} \;=\; -\sum_{\boldsymbol{v}\in\mathbb{V}} \mu_{\boldsymbol{v}}\, x_{\boldsymbol{v},j}\, x_{\boldsymbol{v},k}.$$

In vector form, the score (first derivative) vector and Hessian (second derivative) matrix are

$$\frac{\partial l}{\partial \boldsymbol{\beta}} \;=\; \boldsymbol{X}^\top(\boldsymbol{f} - \boldsymbol{\mu}) \quad \text{and} \quad \frac{\partial^2 l}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^\top} \;=\; -\,\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X},$$

where $\boldsymbol{W} = \text{diag}(\boldsymbol{\mu})$ is the square matrix with elements $\mu_{\boldsymbol{v}}$, $\boldsymbol{v} \in \mathbb{V}$ on the diagonal and zeros elsewhere.

In well-behaved problems, NR converges reliably and quickly. Failure to converge generally indicates that the ML estimate is not unique or that it lies on a boundary of the parameter space where one or more cell probabilities are zero. If we judge convergence by observing whether changes in the elements of the estimated $\boldsymbol{\beta}$ are sufficiently small, then ML estimates on a boundary will cause some elements of $\boldsymbol{\beta}$ to run away toward $+\infty$ or $-\infty$, depending on how $\boldsymbol{X}$ is coded, and the test will fail. But if we examine changes in the elements of $\boldsymbol{\pi}$ or $\boldsymbol{\mu}$, an estimate on the boundary may not be problematic, as the changes from one iteration to the next on that scale will become smaller and smaller. In that case, there is a potential for numerical overflow or underflow when exponentiating the elements of $\boldsymbol{X\beta}$, and the estimated covariance matrix for $\boldsymbol{\beta}$ may not be reliable.

Appendix E Notation for coarsened factors

Returning to the notation for categorical variables in Appendix A, we now expand the notation to allow missing and coarsened values. As before, let V_{ij} denote the jth categorical variable for observational unit i. The possible values for V_{ij} are

$$\mathbb{V}_j \;=\; \{1, 2, \ldots, \#\mathbb{V}_j\}.$$

Members of $\mathbb{V}_j$ are called base-level codes; these are all the possible responses that would be seen if there were no missing or coarsened data.

Let V^*_{ij} denote a coarsened version of V_{ij}. The possible values of V^*_{ij} lie in the expanded set

$$\mathbb{V}^*_j \;=\; \{1, 2, \ldots, \#\mathbb{V}_j, \ldots, \#\mathbb{V}^*_j\},$$

where $\#\mathbb{V}_j^* > \#\mathbb{V}_j$. The extra codes not found in $\mathbb{V}_j$,

$$\mathbb{V}_j^* \setminus \mathbb{V}_j = \{\#\mathbb{V} + 1, \ldots, \#\mathbb{V}_j^*\},$$

are called coarse-level codes.

If V_{ij}^* happens to be one of the base-level codes, then V_{ij} is equal to V_{ij}^*,

$$\begin{aligned} V_{ij}^* = 1 &\Rightarrow V_{ij} = 1, \\ &\vdots \\ V_{ij}^* = \#\mathbb{V}_j &\Rightarrow V_{ij} = \#\mathbb{V}_j. \end{aligned}$$

However, if V_{ij}^* happens to be one of the coarse-level codes, the exact value of V_{ij} cannot be deduced from it. In that case, V_{ij} is known to lie within a subset of the base-level codes, a set denoted by $\mathcal{M}_j(V_{ij}^*)$. That is,

$$V_{ij}^* = v^* \Rightarrow V_{ij} \in \mathcal{M}_j(v^*),$$

where $\mathcal{M}_j$ is the mapping, a one-to-many relation that maps elements of $\mathbb{V}^*$ onto non-empty subsets of $\mathbb{V}_j$. We use the last coarse-level code to denote a traditional missing value,

$$\mathcal{M}_j(v^*) = \mathbb{V}_j \text{ when } v^* = \#\mathbb{V}_j^*.$$

As the number of base-level codes increases, the number of possible coarse-level codes expands rapidly. In practice, we do not need to create a coarse-level code for every possible subset of the base-level codes, but only for groupings that arise in a given application.

For examples of coarsened categorical variables and information on how the `cvam` package creates and stores them, see the accompanying vignette *Understanding Coarsened Factors in `cvam`*.

Consider now a vector of J coarsened variables $\boldsymbol{V}_i^* = (V_{i1}^*, \ldots, V_{iJ}^*)$, a coarsened version of $\boldsymbol{V}_i = (V_{i1}^*, \ldots, V_{iJ})$. Let $\boldsymbol{v}^* = (v_1^*, \ldots, v_J^*)$ denote a possible value for $\boldsymbol{V}_i^*$. The set of all possible values for $\boldsymbol{V}_i^*$ is

$$\mathbb{V}^* = \mathbb{V}_1^* \times \mathbb{V}_2^* \times \cdots \times \mathbb{V}_J^* = \prod_{j=1}^{J} \mathbb{V}_j^*.$$

This set can become very large, with $\#\mathbb{V}^* >> \#\mathbb{V}$. The `cvam` function never enumerates $\mathbb{V}^*$ or creates any tables corresponding to $\mathbb{V}^*$ but only work with the elements of $\mathbb{V}^*$ that are seen in the rows of the grouped or ungrouped data frame supplied to `cvam` through its `data` argument. Observing $\boldsymbol{V}_i^* = \boldsymbol{v}^*$ implies that the uncoarsened version $\boldsymbol{V}_i$ lies within a known subset of $\mathbb{V}$. Specifically,

$$\boldsymbol{V}_i^* = \boldsymbol{v}^* \Rightarrow \boldsymbol{V}_i \in \mathcal{M}(\boldsymbol{v}^*),$$

where $\mathcal{M}$ is the one-to-many mapping function

$$\mathcal{M}(\boldsymbol{v}^*) = \mathcal{M}_1(v_1^*) \times \mathcal{M}_2(v_2^*) \times \cdots \times \mathcal{M}_J(v_J^*)$$

Appendix F Seen, true, and augmented data

The data supplied to `cvam`, whether given as microdata or in a grouped data frame, are aggregated into a vector of coarsened-data integer frequencies

$$\boldsymbol{f}^* \;=\; (f^*_{\boldsymbol{v}^*} \;:\; \boldsymbol{v}^* \in \mathbb{V}^*),$$

where $f^*_{\boldsymbol{v}^*}$ is the number of sample units $i = 1, \ldots, N$ for which $\boldsymbol{V}^*_i = \boldsymbol{v}^*$. These frequencies, and the their corresponding coarsened-data response patterns (i.e., the unique values of $\boldsymbol{v}^*$ appearing in the supplied dataset), are called the seen data and are stored by `cvam` in an object called `mfSeen`. This model frame has $\#\mathbb{V}^*$ rows, whose ordering is determined automatically by the ordering of the factor variables appearing in the model formula. The columns of `mfSeen` include each of the variables in the model as a coarsened factor, plus a variable named `freq` to store $\boldsymbol{f}^*$. The seen data can be retrieved by calling `cvam` with the argument `method="mfSeen"`.

Another model frame, called the true data or `mfTrue`, has $\#\mathbb{V}$ rows. This frame enumerates the full set of complete-data patterns $\boldsymbol{v} \in \mathbb{V}$, including any structural-zero cells. The columns of `mfTrue` include each of the variables in the model as an ordinary non-coarsened factor, plus a variable named `freq` to store $\boldsymbol{f}$. If the data supplied to `cvam` has missing or coarsened values, the elements of $\boldsymbol{f}$ are unknown and must be predicted, which happens during a model-fitting procedure. Before a model is fit, `mfTrue` can be retrieved by calling `cvam` with the argument `method="mfTrue"`; this returns a data frame in which `freq` is filled with `NA` values. After a model has been fit, the true data can be retrieved from the resulting `cvam` object by calling `get.mfTrue`; in that case, `freq` will be populated with predicted true frequencies.

The relationship between the seen data and the true data is determined by another set of frequencies. Let $F_{\boldsymbol{v}^*,\boldsymbol{v}}$ denote the unobserved portion of $f^*_{\boldsymbol{v}^*}$ that belongs to cell $\boldsymbol{v}$ of the complete-data table, and let

$$\boldsymbol{F} \;=\; (F_{\boldsymbol{v}^*,\boldsymbol{v}} \;:\; \boldsymbol{v}^* \in \mathbb{V}^*, \boldsymbol{v} \in \mathbb{V})$$

be the array of frequencies in the table that cross-classifies the N sample units by their values of $\boldsymbol{V}^*_i$ and $\boldsymbol{V}_i$. This array is called the augmented data. When summed over its first subscript, it generates the true frequencies,

$$\sum_{\boldsymbol{v}^* \in \mathbb{V}^*} F_{\boldsymbol{v}^*,\boldsymbol{v}} \;=\; f_{\boldsymbol{v}},$$

and when summed over its second subscript, it reproduces the seen frequencies,

$$\sum_{\boldsymbol{v} \in \mathbb{V}} F_{\boldsymbol{v}^*,\boldsymbol{v}} \;=\; \sum_{\boldsymbol{v} \in \mathcal{M}(\boldsymbol{v}^*)} F_{\boldsymbol{v}^*,\boldsymbol{v}} \;=\; f_{\boldsymbol{v}^*}.$$

79

With $\#\mathbb{V}^* \times \#\mathbb{V}$ cells, $\boldsymbol{F}$ is potentially huge. It is also very sparse, because $F_{\boldsymbol{v}^*,\boldsymbol{v}} = 0$ whenever $\boldsymbol{v} \notin \mathcal{M}(\boldsymbol{v}^*)$. The array is never actually formed by `cvam`, but individual nonzero elements are repeatedly predicted and discarded during a modeling run.

The augmented-data array was introduced by Baker (1994), who discussed applying product-multinomial and Poisson models to its elements. In `cvam`, we model $\boldsymbol{f}$ rather than $\boldsymbol{F}$. This strategy is appropriate if the coarsened data are coarsened at random (CAR), and if the parameters of the unspecified coarsening mechanism and the true-data model are distinct (Heitjan and Rubin, 1991).

Appendix G EM algorithm for coarsened categorical data

If there were no missing or coarsened values, we would fit log-linear models directly to the true frequencies $\boldsymbol{f}$. Omitting constants that do not involve $\boldsymbol{\mu}$, the surrogate-Poisson loglikelihood function based on the true data is

$$l^A(\boldsymbol{\mu} \mid \boldsymbol{f}) \;=\; -\,\mu_+ \;+\; \sum_{\boldsymbol{v} \in \mathbb{V}} f_{\boldsymbol{v}} \log \mu_{\boldsymbol{v}}, \tag{21}$$

where $\mu_+ = \sum_{\boldsymbol{v} \in \mathbb{V}}$. The superscipt '$A$' stands for augmented. This function depends on the augmented data $\boldsymbol{F}$ through its margin $\boldsymbol{f}$, and we refer to it as the augmented-data loglikelihood.

When $\boldsymbol{f}$ is not fully observed, inferences must be based on the seen frequencies $\boldsymbol{f}^*$. Baker (1994) describes surrogate-Poisson likelihood functions for incomplete-data problems based on a multinomial distribution over the non-structural zero cells of $\boldsymbol{F}$. Using results from Baker (1994), and assuming the coarsening mechanism is ignorable, an appropriate surrogate-Poisson loglikelihood function based on $\boldsymbol{f}^*$ is

$$l(\boldsymbol{\mu} \mid \boldsymbol{f}^*) \;=\; -\,\mu_+ \;+\; \sum_{\boldsymbol{v}^* \in \mathbb{V}^*} f^*_{\boldsymbol{v}^*} \log \tau_{\boldsymbol{v}^*}, \tag{22}$$

where

$$\tau_{\boldsymbol{v}^*} \;=\; \sum_{\boldsymbol{v} \in \mathcal{M}(\boldsymbol{v}^*)} \mu_{\boldsymbol{v}}.$$

We refer to $l(\boldsymbol{\mu} \mid \boldsymbol{f}^*)$ as the observed-data loglikelihood.

The EM algorithm is an iterative procedure for maximizing $l(\boldsymbol{\mu} \mid \boldsymbol{f}^*)$ by repeatedly maximizing a function that looks like $l^A(\boldsymbol{\mu} \mid \boldsymbol{f})$. At each iteration, the current estimate of the log-linear coefficients $\hat{\boldsymbol{\beta}}^{(t)}$ is updated in two steps. In the Expectation or E-step, we obtain the function

$$Q^{(t)}(\boldsymbol{\beta}) \;=\; E\,[\,l^A(\boldsymbol{\mu} \mid \boldsymbol{f}) \mid \boldsymbol{f}^*, \boldsymbol{\beta} = \hat{\boldsymbol{\beta}}^{(t)}\,], \tag{23}$$

and in the M-step, we maximize that function to obtain the new estimate,

$$\hat{\boldsymbol{\beta}}^{(t+1)} \;=\; \underset{\boldsymbol{\beta}}{\arg\max}\; Q^{(t)}(\boldsymbol{\beta}) \tag{24}$$

To perform the E-step, note that the augmented-data loglikelihood is a linear function of the elements of $\boldsymbol{f}$, so its expectation is obtained by replacing $\boldsymbol{f}$ with its expected value given $\boldsymbol{f}^*$ under the assumed value for $\boldsymbol{\beta}$ or $\boldsymbol{\mu}$. Under a multinomial model for the non-structural zero elements of $\boldsymbol{F}$, the conditional distribution of $\boldsymbol{F}$ given $\boldsymbol{f}^*$ becomes product-multinomial. That is, each slice

$$\boldsymbol{F}_{\boldsymbol{v}^*,:} \;=\; (\, \boldsymbol{F}_{\boldsymbol{v}^*,\boldsymbol{v}} : \boldsymbol{v} \in \mathcal{M}(\boldsymbol{v}^*) \,)$$

is distributed as

$$\boldsymbol{F}_{\boldsymbol{v}^*,:} \,|\, \boldsymbol{f}^*_{\boldsymbol{v}^*}, \boldsymbol{\mu} \;\sim\; \text{Mult}(\boldsymbol{f}^*_{\boldsymbol{v}^*}, \boldsymbol{\xi}_{\boldsymbol{v}^*}) \tag{25}$$

independently for $\boldsymbol{v}^* \in \mathbb{V}^*$, where $\boldsymbol{\xi}_{\boldsymbol{v}^*}$ is the vector with elements $\xi_{\boldsymbol{v}^*,\boldsymbol{v}} = \mu_{\boldsymbol{v}}/\tau_{\boldsymbol{v}^*}$ for $\boldsymbol{v} \in \mathcal{M}(\boldsymbol{v}^*)$. The expected value of $\boldsymbol{f}$ under this product-multinomial distribution, which we denote by $\hat{\boldsymbol{f}} = E(\boldsymbol{f} \,|\, \boldsymbol{f}^*, \boldsymbol{\mu})$, has elements $\hat{f}_{\boldsymbol{v}} = \sum_{\boldsymbol{v}^* \in \mathbb{V}^*} \hat{F}_{\boldsymbol{v}^*,\boldsymbol{v}}$, where

$$\hat{F}_{\boldsymbol{v}^*,\boldsymbol{v}} \;=\; I(\, \boldsymbol{v} \in \mathcal{M}(\boldsymbol{v}^*) \,)\, f^*_{\boldsymbol{v}^*} \xi_{\boldsymbol{v}^*,\boldsymbol{v}}. \tag{26}$$

The E-step accumulates $\hat{\boldsymbol{f}}$ by cycling over the rows of the seen data, computing $\tau_{\boldsymbol{v}^*}$ for that row, then incrementing the each element $\hat{f}_{\boldsymbol{v}}$ for $\boldsymbol{v} \in \mathcal{M}(\boldsymbol{v}^*)$ by the amount $f^*_{\boldsymbol{v}^*}\mu_{\boldsymbol{v}}/\tau_{\boldsymbol{v}^*}$. While performing this E-step, `cvam` also computes the observed-data loglikelihood (22) under the current estimate of $\boldsymbol{\beta}$ or $\boldsymbol{\mu}$, which comes at essentially no cost.

Once the E-step has been completed, the M-step fits the log-linear model by the NR procedure described in Appendix D, with $\boldsymbol{f}$ replaced by $\hat{\boldsymbol{f}}$, to obtain the new estimate for $\boldsymbol{\beta}$. If the model is saturated and `cvam` is called with `saturated=TRUE`, the coefficients $\boldsymbol{\beta}$ are not defined, and the M-step sets the new estimate of $\boldsymbol{\mu}$ to $\hat{\boldsymbol{f}}$.

Observations that are entirely missing (i.e., rows of the seen data that have missing values for all modeled variables) contribute no information to the observed-data loglikelihood function (22), except for driving up the estimate of μ_+. Including these observations can increase the rates of missing information and slow the convergence of EM (Schafer, 1997). By default, `cvam` excludes these observations from the model-fitting procedure but restores them afterward when reporting the predicted true frequencies $\hat{\boldsymbol{f}}$ in the model frame `mfTrue`. To include these cases in the model fitting, set the control parameter `excludeAllNA` to `FALSE` in the `control` argument.

Appendix H Prior information

Prior distributions implemented in `cvamPrior` allow the user to incorporate prior information as

- a flattening constant, a positive value that is divided equally among all non-structural zero cells of the complete-data table, and
- prior nuggets, which take the form of coarsened-data frequencies and are ascribed to groups of cells (slices of the table).

The result is a kind of data-augmentation prior (DAP) in which the flattening constant and nuggets contribute additively to the objective function in the same way that seen frequencies $\boldsymbol{f}^*$ contribute to the observed-data loglikelihood. Including prior information, the objective function becomes

$$\log P(\boldsymbol{\mu}) \;=\; \sum_{\boldsymbol{v}\in\mathbb{V}} k \log \mu_{\boldsymbol{v}} \;+\; \sum_{\boldsymbol{v}^*\in\mathbb{P}} f^*_{\boldsymbol{v}^*} \log \tau_{\boldsymbol{v}^*} \;+\; l(\boldsymbol{\mu} \,|\, \boldsymbol{f}^*), \tag{27}$$

where k is the per-cell flattening constant, $\mathbb{P}$ is a set of coarsened-data cells over which the prior nuggets are defined, and $f^*_{\boldsymbol{v}^*}$ denotes a prior nugget if $\boldsymbol{v}^* \in \mathbb{P}$ or a seen frequency if $\boldsymbol{v}^* \in \mathbb{V}^*$. The indices for cells in $\mathbb{P}$ and their associated prior nuggets are stored in a model frame called `mfPrior`, which can be accessed by calling `cvam` with `method="mfPrior"`.

For log-linear models fit with `saturated=FALSE`, the user can also specify a ridge factor, which acts upon the coefficients in a manner similar to ridge regression, shrinking the estimated $\boldsymbol{\beta}$ toward $\mathbf{0} = (0, \ldots, 0)^\top$ and stabilizing its estimated covariance matrix. A ridge factor $r > 0$ adds information equivalent to a multivariate normal prior density for $\boldsymbol{\beta}$ centered at $\mathbf{0}$ with prior covariance matrix $r^{-1}\boldsymbol{I}$. With a ridge factor, the objective function becomes

$$\log P(\boldsymbol{\beta}) \;=\; -\frac{r}{2}\,\boldsymbol{\beta}^\top\boldsymbol{\beta} \;+\; \sum_{\boldsymbol{v}\in\mathbb{V}} k \log \mu_{\boldsymbol{v}} \;+\; \sum_{\boldsymbol{v}^*\in\mathbb{P}} f^*_{\boldsymbol{v}^*} \log \tau_{\boldsymbol{v}^*} \;+\; l(\boldsymbol{\mu} \,|\, \boldsymbol{f}^*). \tag{28}$$

If a saturated model if fit with the option `saturated=TRUE`, the coefficients $\boldsymbol{\beta}$ are not defined, and the ridge factor is ignored.

In the EM algorithm, (27) and (28) are not treated as the logarithms of an actual density, but as a scale-invariant objective function that takes the same value regardless of whether the parameters are expressed as $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ or $\boldsymbol{\beta}$. The E-step treats the flattening constant and prior nuggets as actual data, apportioning them to the cells of the complete-data table and making the elements of $\hat{\boldsymbol{f}}$ larger. The ridge factor is applied in the M-step, contributing a term $-r\,\boldsymbol{\beta}$ to the score vector and $-r\,\boldsymbol{I}$ to the Hessian matrix described in Appendix D. After the EM run, $\hat{\boldsymbol{f}}$ is recomputed under the final parameter estimates without the flattening constant or nuggets and reported as the variable `freq` in `get.mfTrue`.

Appendix I Standard errors for coefficients

After running the EM algorithm for a log-linear model, `cvam` computes an estimated covariance matrix for $\boldsymbol{\beta}$ based on the observed second derivatives of $\log P$,

$$\hat{V}(\hat{\boldsymbol{\beta}}) \;=\; \left(-\,\frac{\partial^2 \log P}{\partial \boldsymbol{\beta}\, \partial \boldsymbol{\beta}^\top} \right)^{-1},$$

where the derivatives are evaluated at the final estimate for $\boldsymbol{\beta}$. The inverse is computed using a Cholesky factorization and fails if $\log P$ is not concave. To compute the derivatives, we temporarily put aside the ridge factor and use a chain rule

$$\frac{\partial \log P}{\partial \boldsymbol{\beta}} \;=\; \left(\frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\beta}^\top} \right)^\top \left(\frac{\partial \log P}{\partial \boldsymbol{\mu}^\top} \right).$$

But $\partial \mu_{\boldsymbol{v}} / \partial \beta_j \;=\; \mu_{\boldsymbol{v}}\, x_{\boldsymbol{v},j}$, so $\partial \boldsymbol{\mu} / \partial \boldsymbol{\beta}^\top \;=\; \boldsymbol{W}\boldsymbol{X}$, where $\boldsymbol{W} \;=\; \text{diag}(\boldsymbol{\mu})$. With some algebra, we can show that $\partial \log P / \partial \mu_{\boldsymbol{v}} = (\hat{f}_{\boldsymbol{v}} - \mu_{\boldsymbol{v}})/\mu_{\boldsymbol{v}}$, where $\hat{f}_{\boldsymbol{v}}$ is the predicted true frequency for cell $\boldsymbol{v}$ from the E-step of EM, including the flattening constant and any contributions from prior nuggets. It follows that $\partial \log P / \partial \boldsymbol{\mu} \;=\; \boldsymbol{W}^{-1}(\hat{\boldsymbol{f}} - \boldsymbol{\mu})$ and

$$\frac{\partial \log P}{\partial \boldsymbol{\beta}} \;=\; \boldsymbol{X}^\top (\hat{\boldsymbol{f}} - \boldsymbol{\mu}), \tag{29}$$

which vanishes at the EM solution. Applying the chain rule again, the second derivative can be written as

$$\frac{\partial^2 \log P}{\partial \boldsymbol{\beta}\, \partial \boldsymbol{\beta}^\top} \;=\; \sum_{\boldsymbol{v} \in \mathbb{V}} \left(\frac{\partial \log P}{\partial \mu_{\boldsymbol{v}}} \right) \left(\frac{\partial^2 \mu_{\boldsymbol{v}}}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top} \right) \;+\; \left(\frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\beta}^\top} \right)^\top \left(\frac{\partial^2 \log P}{\partial \boldsymbol{\mu} \partial \boldsymbol{\mu}^\top} \right) \left(\frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\beta}^\top} \right). \tag{30}$$

But $\partial^2 \log \mu_{\boldsymbol{v}} / \partial \beta_j\, \partial \beta_k = \mu_{\boldsymbol{v}}\, x_{\boldsymbol{v},j}\, x_{\boldsymbol{v},k}$, so $\partial^2 \mu_{\boldsymbol{v}} / \partial \boldsymbol{\beta}\, \partial \boldsymbol{\beta}^\top = \mu_{\boldsymbol{v}}\, \boldsymbol{x}_{\boldsymbol{v}}\, \boldsymbol{x}_{\boldsymbol{v}}^\top$, and the first term in (30) becomes

$$\sum_{\boldsymbol{v} \in \mathbb{V}} \left(\frac{\partial \log P}{\partial \mu_{\boldsymbol{v}}} \right) \left(\frac{\partial^2 \mu_{\boldsymbol{v}}}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top} \right) \;=\; \boldsymbol{X}^\top \text{diag}(\hat{\boldsymbol{f}} - \boldsymbol{\mu})\, \boldsymbol{X}.$$

With some algebra, the second term in (30) can be written as

$$\left(\frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\beta}^\top} \right)^\top \left(\frac{\partial^2 \log P}{\partial \boldsymbol{\mu} \partial \boldsymbol{\mu}^\top} \right) \left(\frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\beta}^\top} \right) \;=\; \boldsymbol{X}^\top \boldsymbol{M} \boldsymbol{X},$$

where

$$\boldsymbol{M} \;=\; -\,k\, \boldsymbol{I} \;-\; \sum_{\boldsymbol{v}^* \in \mathbb{P}} \boldsymbol{M}_{\boldsymbol{v}^*} \;-\; \sum_{\boldsymbol{v}^* \in \mathbb{V}^*} \boldsymbol{M}_{\boldsymbol{v}^*},$$

where $\boldsymbol{I}$ is the $\#\mathbb{V} \times \#\mathbb{V}$ identity matrix, and $\boldsymbol{M}_{\boldsymbol{v}^*}$ is the matrix with element

$$\hat{F}_{\boldsymbol{v}^*, \boldsymbol{v}}\, \hat{F}_{\boldsymbol{v}^*, \boldsymbol{v}'} / f^*_{\boldsymbol{v}^*} \tag{31}$$

in position $(\boldsymbol{v}, \boldsymbol{v}')$ and zeros elsewhere. The matrices $\boldsymbol{M}_{\boldsymbol{v}^*}$ are large and very sparse, because (31) becomes zero whenever $\boldsymbol{v} \notin \mathcal{M}(\boldsymbol{v}^*)$ or $\boldsymbol{v}' \notin \mathcal{M}(\boldsymbol{v}^*)$. In `cvam`, these matrices are never formed; rather, we cycle over the cells of $\mathbb{P}$ and $\mathbb{V}^*$ and accumulate $\boldsymbol{MX}$ in a workspace of the same size as $\boldsymbol{X}$.

These formulas do not account for a ridge factor. If a ridge factor $r > 0$ is present, the term $-\,r\,\boldsymbol{\beta}$ is added to the first derivative, and the term $-\,r\,\boldsymbol{I}$ is added to the second derivative.

Appendix J Estimated probabilities and standard errors

The function `cvamEstimate` computes tables of marginal and conditional probabilities from the estimated cell means $\hat{\boldsymbol{\mu}}$. The formula given to `cvamEstimate` partitions the model variables into $\boldsymbol{V}_i = (\boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{C}_i)$, where $\boldsymbol{A}_i$ denotes the variables to be conditioned on, $\boldsymbol{B}_i$ denotes the variables for which probabilities are requested, and $\boldsymbol{C}_i$ denotes variables to be marginalized over. With respect to the `cvamEstimate` formula,

- $\boldsymbol{A}_i$ consists of all variables on the right-hand side of '|',
- $\boldsymbol{B}_i$ consists of all variables on the left-hand side of '|', and
- $\boldsymbol{C}_i$ consists of all variables from the model formula that are absent from the `cvamEstimate` formula.

The set $\boldsymbol{B}_i$ must include any variables that the model regarded as fixed, otherwise `cvamEstimate` reports an error. Writing a possible value for $\boldsymbol{V}_i$ as $\boldsymbol{v} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$, we may identify cells of true data frame by this triple index, as in

$$\pi_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}} \;=\; P(\boldsymbol{A}_i = \boldsymbol{a}, \boldsymbol{B}_i = \boldsymbol{b}, \boldsymbol{C}_i = \boldsymbol{c}).$$

The probabilities requested by `cvamEstimate` are $P(\boldsymbol{B}_i = \boldsymbol{b} \,|\, \boldsymbol{A}_i = \boldsymbol{a})$ for all $\boldsymbol{a} \in \mathbb{A}$ and $\boldsymbol{b} \in \mathbb{B}$, which can be written as

$$\pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+} \;=\; \sum_{\boldsymbol{c} \in \mathbb{C}} \frac{\mu_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}}}{\mu_{\boldsymbol{a},+,+}}.$$

Using the delta method (Agresti, 2013), an estimated variance for $\hat{\pi}^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}$ is

$$\hat{V}(\hat{\pi}^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}) \;=\; \left(\frac{\partial\, \pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}}{\partial \boldsymbol{\beta}^\top} \right)^\top \hat{V}(\hat{\boldsymbol{\beta}}) \left(\frac{\partial\, \pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}}{\partial \boldsymbol{\beta}^\top} \right), \tag{32}$$

84

where the derivatives are evaluated at $\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}$. We compute this as $\hat{V}(\hat{\pi}^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}) = \|\boldsymbol{D}^{-1}\boldsymbol{d}_{\boldsymbol{a},\boldsymbol{b}}\|^2$, where $\boldsymbol{D}$ is the lower-triangular Cholesky factor of $-\,\partial^2 \log P/\partial\boldsymbol{\beta}\,\partial\boldsymbol{\beta}^\top$, and

$$\boldsymbol{d}_{\boldsymbol{a},\boldsymbol{b}} \;=\; \left(\frac{\partial\boldsymbol{\mu}}{\partial\boldsymbol{\beta}^\top}\right)^\top \left(\frac{\partial\,\pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}}{\partial\boldsymbol{\mu}}\right). \tag{33}$$

The first term in (33) is $\partial\boldsymbol{\mu}/\partial\boldsymbol{\beta}^\top = \boldsymbol{W}\boldsymbol{X}$, where $\boldsymbol{W} = \mathrm{diag}(\boldsymbol{\mu})$. For the second term, note that

$$\frac{\partial\,\pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}}{\partial\mu_{\boldsymbol{a}',\boldsymbol{b}',\boldsymbol{c}'}} \;=\; 0 \quad \text{if} \quad \boldsymbol{a}' \neq \boldsymbol{a}.$$

If $\boldsymbol{a}' = \boldsymbol{a}$, the derivative is

$$\frac{\partial\,\pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}}{\partial\mu_{\boldsymbol{a},\boldsymbol{b}',\boldsymbol{c}'}} \;=\; \frac{1}{\mu_{\boldsymbol{a},+,+}}\left(I(\boldsymbol{b}' = \boldsymbol{b}) \;-\; \pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}\right),$$

which is the same for all $\boldsymbol{c}'$. Putting these together, the elements of $\boldsymbol{d}_{\boldsymbol{a},\boldsymbol{b}}$ become

$$\boldsymbol{d}_{\boldsymbol{a},\boldsymbol{b},j} \;=\; \sum_{\boldsymbol{b}'\in\mathbb{B}}\sum_{\boldsymbol{c}'\in\mathbb{C}} x_{\boldsymbol{a},\boldsymbol{b}',\boldsymbol{c}',j}\,\pi^{(B,C\,|\,A)}_{\boldsymbol{a},\boldsymbol{b}',\boldsymbol{c}'}\left(I(\boldsymbol{b}' = \boldsymbol{b}) \;-\; \pi^{(B\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},+}\right)$$

for $j = 1, \ldots, p$, where $\pi^{(B,C\,|\,A)}_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}} = \mu_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}}/\mu_{\boldsymbol{a},+,+}$.

Appendix K Markov chain Monte Carlo

Appendix K.1 Data augmentation

MCMC algorithms for incomplete multivariate categorical data were described by Schafer (1997), including a stochastic version of iterative proportional fitting called Bayesian IPF. For our purposes, it is more natural to use methods that are not related to IPF but focus on the model matrix $\boldsymbol{X}$ and the log-linear coefficients $\boldsymbol{\beta}$, except for models that are specifically fit as saturated with the option `saturated=TRUE`.

By default, when the `cvam` function is called with `method="MCMC"`, it runs a data-augmentation (DA) procedure that resembles EM. Let $\boldsymbol{\beta}^{(t)}$ denote the simulated value of $\boldsymbol{\beta}$ at iteration t, and let $\boldsymbol{\mu}^{(t)}$ denote the corresponding value of $\boldsymbol{\mu}$. In DA, generating the next iterate $\boldsymbol{\beta}^{(t+1)}$ involves

- an Imputation or I-step, which draws a table of true frequencies from its predictive distribution given the seen frequencies and the current simulated parameters,

  $$\boldsymbol{f}^{(t)} \;\sim\; P(\boldsymbol{f}\,|\,\boldsymbol{f}^*,\,\boldsymbol{\mu} = \boldsymbol{\mu}^{(t)}), \tag{34}$$

  followed by

- a Posterior or P-step, which draws a new set of parameters given the simulated true frequencies,

$$\boldsymbol{\beta}^{(t+1)} \sim P(\boldsymbol{\beta} \mid \boldsymbol{f} = \boldsymbol{f}^{(t)}). \tag{35}$$

Choosing a starting value $\boldsymbol{\beta}^{(0)}$ and repeating these two steps creates a Markov chain $\boldsymbol{\beta}^{(1)}, \boldsymbol{\beta}^{(2)}, \ldots$ whose stationary distribution is the desired posterior $P(\boldsymbol{\beta} \mid \boldsymbol{f}^*)$. The default $\boldsymbol{\beta}^{(0)}$ depends on which S3 method is invoked. If the `cvam` function is called with a model formula as its first argument, then `cvam.formula` chooses default starting values in the center of the parameter space, consistent with cell means that are equal across cells. If the `cvam` is applied to a `cvam` object, then `cvam.cvam` sets $\boldsymbol{\beta}^{(0)}$ to the final value of $\boldsymbol{\beta}$ held in the `cvam` object, regardless of whether that value came from `method="EM"`, `"MCMC"`, or `"approxBayes"`.

The I-step of DA is computationally similar to the E-step described in Appendix G and Appendix H, except that the prior nuggets and seen frequencies are apportioned to cells of the complete-data table in a random fashion. The true frequency in cell $\boldsymbol{v}$, including the contributions of the data-augmentation prior (DAP), is

$$f_{\boldsymbol{v}} = k + \sum_{\boldsymbol{v}^* \in \mathbb{P}} F_{\boldsymbol{v}^*,\boldsymbol{v}} + \sum_{\boldsymbol{v}^* \in \mathbb{V}^*} F_{\boldsymbol{v}^*,\boldsymbol{v}}, \tag{36}$$

where k is the flattening constant, $F_{\boldsymbol{v}^*,\boldsymbol{v}}$ is the portion of the coarsened-data frequency $f^*_{\boldsymbol{v}^*}$ that belongs to cell $\boldsymbol{v}$, $\mathbb{P}$ is the set of coarsened-data cells in the frame `mfPrior` of prior nuggets, and $\mathbb{V}^*$ is the set of coarsened-data cells in the frame `mfSeen` of aggregated user-supplied data. For each $\boldsymbol{v}^*$ in $\mathbb{P}$ and $\mathbb{V}^*$, the integer vector $\boldsymbol{F}_{\boldsymbol{v}^*,:}$ is drawn from the multinomial distribution (25), and the contributions are accumulated into $f_{\boldsymbol{v}}$.

For the P-step, we simulate a draw of $\boldsymbol{\beta}$ from the augmented-data posterior

$$P(\boldsymbol{\beta} \mid \boldsymbol{f}) \propto \exp\left(-\frac{r}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta} - \mu_+ + \sum_{\boldsymbol{v} \in \mathbb{V}} f_{\boldsymbol{v}} \log \mu_{\boldsymbol{v}} \right), \tag{37}$$

which is interpreted as a density for $\boldsymbol{\beta}$ with respect to Lebesgue measure over $\mathbb{R}^p$. Because $P(\boldsymbol{\beta} \mid \boldsymbol{f})$ is difficult to simulate directly, we replace an exact draw from this distribution with one step of a Metropolis-Hastings (MH) algorithm that has $P(\boldsymbol{\beta} \mid \boldsymbol{f})$ as its target, so that the DA algorithm becomes an example of Metropolis-Hastings within Gibbs (Gamerman and Lopes, 2006). Our MH procedure draws a candidate $\boldsymbol{\beta}^*$ from the multivariate t proposal

$$\boldsymbol{\beta}^* \sim t_\nu(\, c(\boldsymbol{\beta}^{(t)}),\, \boldsymbol{S}(\boldsymbol{\beta}^{(t)})\,), \tag{38}$$

where

$$\begin{aligned} c(\boldsymbol{\beta}) &= \boldsymbol{\beta} + \delta\, [\,-\boldsymbol{H}(\boldsymbol{\beta})\,]^{-1} \nabla \log P(\boldsymbol{\beta} \mid \boldsymbol{f}), \\ \boldsymbol{S}(\boldsymbol{\beta}) &= \epsilon^2 \left(\frac{\nu + p}{\nu} \right) [\,-\boldsymbol{H}(\boldsymbol{\beta})\,]^{-1}, \end{aligned}$$

where ∇ denotes the gradient, and H is the second derivative or Hessian matrix of $\log P(\boldsymbol{\beta} \mid \boldsymbol{f})$ with respect to $\boldsymbol{\beta}$. In this notation, $t_\nu(\boldsymbol{c}, \boldsymbol{S})$ denotes a multivariate t distribution centered at $\boldsymbol{c}$ with scale matrix $\boldsymbol{S}$ and ν degrees of freedom, The proposal density is is

$$q(\boldsymbol{\beta}^* \mid \boldsymbol{\beta}^{(t)}) \propto \left[1 + \frac{1}{\nu}(\boldsymbol{\beta}^* - \boldsymbol{c})^\top \boldsymbol{S}^{-1}(\boldsymbol{\beta}^* - \boldsymbol{c})\right]^{-\left(\frac{\nu+p}{2}\right)}$$

with $\boldsymbol{c} = \boldsymbol{c}(\boldsymbol{\beta}^{(t)})$ and $\boldsymbol{S} = \boldsymbol{S}(\boldsymbol{\beta}^{(t)})$.

The quantities ν, δ, and ϵ are tuning constants. When $\delta = 1$, the proposal is centered at the estimate of $\boldsymbol{\beta}$ obtained by taking one step of Newton-Raphson from $\boldsymbol{\beta}^{(t)}$, and when $\epsilon = 1$, we are matching the curvature of the log-proposal density to that of $\log P(\boldsymbol{\beta}^{(t)} \mid \boldsymbol{f})$. A version of this procedure with $(\nu, \delta, \epsilon) = (\infty, 1, 1)$ was applied by Gamerman (1997) for simulating coefficients in generalized linear mixed models; that method achieves an optimal acceptance rate of 100% when the target is multivariate normal. A similar procedure was also used by Pitt et al. (2006), who used a normal proposal centered at the target's mode, which in general requires multiple steps of Newton-Raphson, whereas ours needs only one step. The tuning constants should be chosen to produce low correlations between successive iterates of $\boldsymbol{\beta}$. In our experience, $(\nu, \delta, \epsilon) = (10, 0.8, 0.8)$ often performs well in problems where p is small, and these values the current `cvam` default. The tuning constants are set by the control parameter `tuneDA`. The integer control parameter `stuckLimit` instructs DA to abort if MH gets stuck, i.e., if the candidates are rejected more than `stuckLimit` times in a row, which can happen in higher-dimensional problems. If MH gets stuck, the problem can usually be solved by increasing the degrees of freedom and reducing the step size and scale factor, setting `tuneDA` to, say $(100, 0.4, 0.4)$, $(1000, 0.2, 0.2)$, $(1000, 0.1, 0.1)$, or even $(1000, 0, 0.1)$. Increasing ν and reducing δ and ϵ causes MH to take smaller steps, which decreases the chance of getting stuck but increases the correlation between successive values of $\boldsymbol{\beta}$.

When a saturated model is fit with the option `saturated=TRUE`, a different P-step is required, because the log-linear coefficients $\boldsymbol{\beta}$ are not defined. For that case, we omit the ridging term involving $\boldsymbol{\beta}$ from the right-hand side of (37) and interpret it as a non-normalized density for $\boldsymbol{\mu}$ with respect to Lebesgue measure over $\mathbb{R}^{\#\mathbb{V}}$, which implies that

$$\boldsymbol{\mu}_{\boldsymbol{v}} \mid \boldsymbol{f} \sim \text{Gamma}(f_{\boldsymbol{v}} + 1, 1)$$

independently for $\boldsymbol{v} \in \mathbb{V}$, where $\text{Gamma}(a, b)$ denotes a gamma distribution with shape a and rate b. This P-step does not rely on tuning parameters and has an acceptance rate of 100%. Results for a saturated model may be slightly different when run under `saturated=FALSE` and `saturated=TRUE`, because the target posterior distributions differ. These differences will become less noticeable as the sample size N grows.

Appendix K.2 Random-walk Metropolis

When `saturated=FALSE`, an alternative to DA is available that does not impute $\boldsymbol{f}$ at each cycle. That alternative is a random-walk Metropolis (RWM) algorithm with a non-normalized target density obtained by exponentiating the right-hand side of Equation (27). The proposal distribution is

$$\boldsymbol{\beta}^* \sim t_\nu(\boldsymbol{\beta}^{(t)}, \epsilon^2 \hat{V}(\hat{\boldsymbol{\beta}})),$$

where $\hat{V}(\hat{\boldsymbol{\beta}})$ is the asymptotic covariance matrix computed at the end of EM. To use this alternative procedure, set the control parameter `typeMCMC` to `"RWM"`. The tuning constants (ν, ϵ), which default to $(1000, 0.1)$, are set by the control parameter `tuneRWM`. If the algorithm gets stuck, the problem can usually be solved by decreasing ϵ, which will increase the acceptance rate. With RWM, an acceptance rate between 20% and 40% is desirable.

Appendix K.3 More details of MCMC

Four additional control parameters apply regardless of which MCMC algorithm is used. They are:

- `iterMCMC`, which sets the number of iterations to be performed after the burn-in period;
- `burnMCMC`, which sets the number of iterations to be treated as a burn-in period and discarded;
- `thinMCMC`, which sets the thinning interval; and
- `imputeEvery`, which sets the imputation interval.

After the burn-in period, `cvam` accumulates and saves:

- a running average and a running covariance matrix from the output stream of $\boldsymbol{\beta}$. These become the default source for the estimated coefficients and standard errors extracted by `summary` and `get.coef` and the estimated covariance matrix extracted by `get.covMat`.
- a running average of the cell probabilities $\boldsymbol{\pi}$, which become the source of fitted values extracted by `get.fitted`.
- a running average of the imputed true frequencies $\boldsymbol{f}$, without the flattening constant or any contributions from prior nuggets. These are reported as the variable `freq` in the data frame `mfTrue`.

- a series containing every kth value of -2 times the objective function in Equation (27), where k is the thinning interval set by `thinMCMC`. This series can be extracted with `get.minus2logPSeries`.
- a series containing every kth simulated value of $\boldsymbol{\beta}^{(t)}$, which can be extracted with `get.coefSeries`.
- if `saveProbSeries=TRUE`, a series containing every kth value of $\boldsymbol{\pi}^{(t)}$, which can be extracted with `get.probSeries`.
- every mth value of the imputed true frequencies $\boldsymbol{f}$, without the flattening constant or any contributions from prior nuggets, where m is the imputation interval set by `imputeEvery`. These can be extracted with `get.imputedFreq`. If m is sufficiently large, these can be regarded as proper Bayesian multiple imputations of $\boldsymbol{f}$ under the model.

89