Package Tutorial for pgmultinomr

Lauren Hoskovec

October 28, 2021

The R package pgmultinomr provides a function to fit a Bayesian categorical regression model with partially or fully missing outcomes, using polya-gamma (PG) data augmentation for efficient Gibbs inference. For more details on the method, see [Hoskovec et al.]. The following functions are available in the package pgmultinomr:

function	brief description of usage
pgmultinom	fit Bayesian categorical regression with K>2 using PG data augmentation
sim_dat	simulate data to fit pgmultinom
sim_study	reproduce simulation study in [Hoskovec et al.]

In this vignette, we show how to install the package, fit the model, and post-process and visualize the results for inference. We also demonstrate how to reproduce the simulation study in [Hoskovec et al.].

Install the Package

First, load the following package dependencies. Use install.packages() to install them, if needed.

```
library(Rcpp)
library(RcppArmadillo)
library(mvnfast)
library(pgdraw)
```

The package can then be installed from gitHub.

```
devtools::install_github("lvhoskovec/pgmultinomr", build_vignettes = TRUE)
library(pgmultinomr)
```

Overview

We briefly describe the model here. For more details, see [Hoskovec et al.].

For a sample $i = 1, \dots, n$, let $\mathbf{y}_i$ denote the K-dimensional vector indicating to which of K possible outcome categories individual i belong. We model $\mathbf{y}_i$ with a multinomial distribution where the number of trials is 1. Using the stick-breaking representation of the multinomial distribution, we model the complete data for individuals $i = 1, \dots, n$ by

$$\mathbf{y}_i \sim \prod_{k=1}^{K-1} \text{binom}(y_{ik} | N_{ik}, \tilde{\pi}_{ik}),$$

where $N_{i1} = 1$ and $N_{ik} = 1 - \sum_{j<k} y_{ij}$. We model each $\tilde{\pi}_{ik}$ for $i = 1, \ldots, n$ and $k = 1, \ldots, K-1$ using a logit link function of the exposures and covariates. The logit link for the stick-specific weights is given by

$$\begin{aligned} \tilde{\pi}_{ik} &= \frac{e^{\psi_{ik}}}{1 + e^{\psi_{ik}}} \\ \psi_{ik} &= \mathbf{x}_i^T \boldsymbol{\beta}_k + \mathbf{w}_i^T \boldsymbol{\gamma}_k, \end{aligned}$$

where $\boldsymbol{\beta}_k$ and $\boldsymbol{\gamma}_k$ are category-specific regression coefficients for the exposures and covariates, respectively. Latent polya-gamma random variables are introduced so that the conditional likelihood has a Gaussian kernel. Hence, we place multivariate normal priors on $\boldsymbol{\beta}_k$ and $\boldsymbol{\gamma}_k$ for efficient Gibbs sampling inference. For $k = 1, \ldots, K-1$,

$$\begin{aligned} \boldsymbol{\beta}_k &\sim N(\mathbf{b}_k, \mathbf{B}_k) \\ \boldsymbol{\gamma}_k &\sim N(\mathbf{g}_k, \mathbf{G}_k). \end{aligned}$$

The default priors are standard normal.

Simulate Data

We first use the `sim_dat` function included the package to simulate data. The `sim_dat` function includes the following parameters: `n` is the number of observations, `miss_prob` is the proportion of observations with partially or fully missing outcomes, `K` is the number of outcome categories, `p` is the number of exposures, `q` is the number of covariates, and `covX` is the covariance matrix of the exposure data. Three remaining parameters determine the simulation design: 1) `allmiss` is logical; if TRUE, observations with missing outcome data are missing values for all K categories (fully missing), if FALSE, they are missing values for between 2 and K-1 categories (partially missing), 2) null_scenario is logical; if TRUE, the regression coefficients are all set to 0 (null), if FALSE, the regression coefficients are all simulated from independent standard normal distributions (signal), and 3) equal_probs is logical; if TRUE, category-specific intercepts are set so each category has roughly equal amounts of data (equal probabilities), if FALSE, intercepts are set to mimic the outcome category probabilities from the data analysis in [Hoskovec et al.] (data probabilities).

We generate $n = 1000$ observations with $p = 3$ correlated exposures, $q = 5$ independent covariates plus an intercept, and $K = 6$ outcome variables. To mimic the simulation study in [Hoskovec et al.], we generate exposure data from a multivariate normal distribution with mean 0 and covariance matrix `covX` given by

```
covX = matrix(c(1, -0.13, 0.02, -0.13, 1, -0.86, 0.02, -0.86, 1), 3, 3)
```

Outcome data are simulated from the stick-breaking representation of the multinomial distribution. In this example, we simulate data for the scenario with data probabilities, partially missing outcomes, and a signal. We set the missing data level to 50%. The following code simulates the data for our example:

```
n = 1000
K = 6
p = 3
q = 5
dat = sim_dat(n=n, miss_prob = 0.5, K=K, p=p, q=q,
                    covX = covX, allmiss = FALSE, null_scenario = FALSE, equal_probs =
        FALSE)
```

The object `dat` is a list with elements: `y` the simulated data with missing values, `ycomplete` the complete data, `x` the exposure data, `w` the covariate data, `beta_true` the true exposure regression coefficients, and `gamma_true` the true covariate regression coefficients.

Fit Model

Next we demonstrate how to fit the Bayesian multinomial regression model using the `pgmultinom` function. The function `pgmultinom` takes the following parameters: `niter` is the total number of MCMC iterations, `priors` is a list of priors (see below for details), `y` is the data (with missing values), `ycomplete` is the (optional) complete data, `x` is the exposure data matrix, `w` is the (optional) covariate data matrix, `intercept` is logical and dictates if an intercept term should be estimated, `beta_true` is the (optional) matrix of true exposure regression coefficients, and gamma_true is the (optional) matrix of true covariate regression coefficients. We include the 3 exposures simulated above and all pairwise interactions. We fit the model using our simulated data by

```
niter = 1000
ints <- combn(1:ncol(dat$x), 2)
z <- apply(ints, 2, FUN = function(a) {
  dat$x[,a[1]] * dat$x[,a[2]]
})
x = cbind(dat$x,z)
fit = pgmultinom(niter=niter, priors=NULL, y = dat$y, ycomplete = dat$ycomplete, x = x,
                 w = dat$w, intercept = TRUE,
                 beta_true = dat$beta_true, gamma_true = dat$gamma_true)
```

```
## [1] 100
## [1] 200
## [1] 300
## [1] 400
## [1] 500
## [1] 600
## [1] 700
## [1] 800
## [1] 900
## [1] 1000
```

Post-Process Results

Interpretation of the regression coefficients in the stick-breaking representation of the multinomial distribution is challenging. We present a visualization approach and base inference on the exposure-response function. First, we plot the posterior distribution of the estimated exposure regression coefficients. The following code uses the R packages `ggplot2` and `cowplot`. Install if needed.

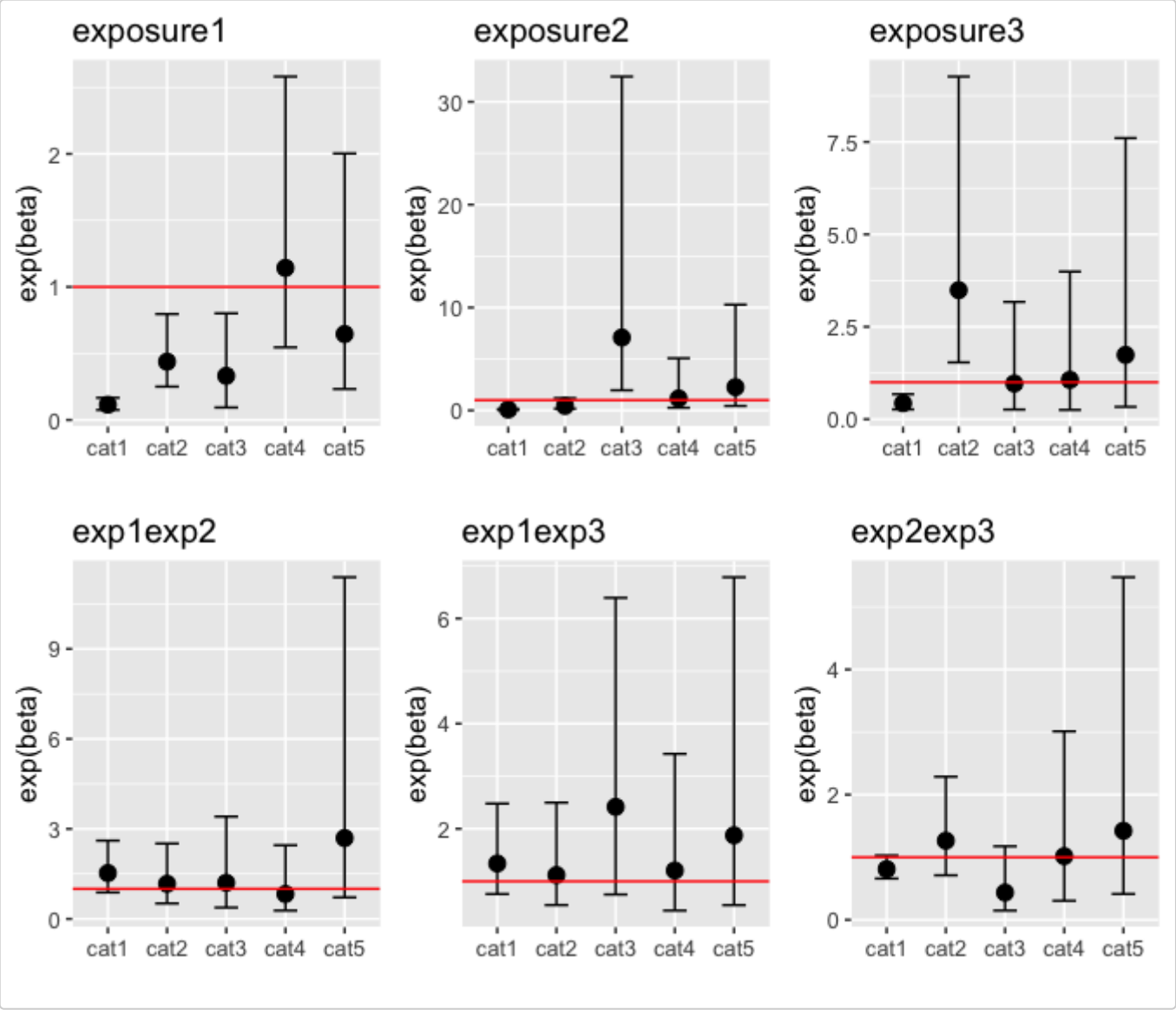
```
library(ggplot2)
library(cowplot)
```

```
nburn = niter/2
beta_quantiles = t(apply(fit$beta.vec[(nburn+1):niter,], 2, FUN = function(b){
    return(c(quantile(b, 0.025), mean(b), quantile(b, 0.975)))}))
namesX = c("exposure1", "exposure2", "exposure3", "exp1exp2", "exp1exp3", "exp2exp3")
namesY = c("cat1", "cat2", "cat3", "cat4", "cat5")
mix_df = cbind(data.frame(exp(beta_quantiles)), rep(namesX,K-1))
```

```r
colnames(mix_df) = c("lwr", "mean", "upr", "exposure")
plot_list = list()

for(p in 1:length(namesX)){
  data_plot = mix_df[which(mix_df$exposure==namesX[p]),]
  limits = c(min(data_plot$lwr), max(data_plot$upr))
  # make a single exposure plot
  mix_plot = ggplot() + geom_point(data = data_plot, mapping = aes(x = 1:(K-1), y = mean),
        size = 3) +
    geom_errorbar(data = data_plot, mapping = aes(x = 1:(K-1), ymin = lwr, ymax = upr),width
        = 0.4) +
    geom_hline(yintercept = 1, colour = "red") +
    scale_x_discrete(name = "", breaks = factor(1:(K-1)), limits = factor(1:(K-1)), label =
        namesY) +
    scale_y_continuous(name = "exp(beta)", limits = limits) +
    theme(text = element_text(size = 12)) + ggtitle(namesX[p])
  # save plots in a list
  plot_list[[p]] = mix_plot
}


# plot all exposures
plot_grid(
  plot_list[[1]] + theme(legend.position="none"),
  plot_list[[2]] + theme(legend.position="none"),
  plot_list[[3]] + theme(legend.position="none"),
  plot_list[[4]] + theme(legend.position="none"),
  plot_list[[5]] + theme(legend.position="none"),
  plot_list[[6]] + theme(legend.position="none"),
  align = 'h',
  labels = "",
  hjust = -1,
  nrow = 2
)
```



Visualize Odds Ratio

Next we show how to visualize the odds ratio for each category relative to a reference category as a function of each exposure. For more details on this inferential approach, see Section 3 in [Hoskovec et al.].

The matrix `x.star` includes a sequence of values for a primary exposure, and sets the secondary exposures to a fixed percentile. The matrix `x.base` includes the primary exposure set to its mean value, and the secondary exposures set to the same fixed percentile. Interactions between exposures are then calculated. The following is an example of how to make `x.star` and `xbase`. In this example, exposure 1 is the primary exposure. The matrix `x.star` includes a sequence of 100 values for exposure 1 ranging from -1 to 1. The secondary exposures, exposures 2 and 3, are set to their 50th percentiles. The matrix `x.base` includes the primary exposure set to its mean and the secondary exposures set to their 50th percentiles. The following code makes `x.star` and `x.base` for this example. The code can be adapted as needed.

```
len = 100
x.star = matrix(0, nrow = len, ncol = ncol(x))
x.star[,1] = seq(-1,1,length.out = len) # sequence of exposure 1 values
x.star[,2] = quantile(x[,2], 0.50) # set exposure 2 to a fixed percentile
x.star[,3] = quantile(x[,3], 0.50) # set exposure 3 to a fixed percentile
```

```
# calculate the interactions
x.star[,4] = x.star[,1]*x.star[,2]
x.star[,5] = x.star[,1]*x.star[,3]
x.star[,6] = x.star[,2]*x.star[,3]

# make x.base
x.base = matrix(0, nrow = 1, ncol = ncol(x))
x.base[,1] = mean(x[,1]) # set exposure 1 to 0
x.base[,2] = quantile(x[,2], 0.50) # set exposure 2 to a fixed percentile
x.base[,3] = quantile(x[,3], 0.50) # set exposure 3 to a fixed percentile

# calculate the interactions
x.base[,4] = x.base[,1]*x.base[,2]
x.base[,5] = x.base[,1]*x.base[,3]
x.base[,6] = x.base[,2]*x.base[,3]
```

Next, we use the function `get_odds_ratio`, provided in the R package, to create the posterior distribution of the odds ratio using `x.star` and `x.base`. The function `get_odds_ratio` takes the following parameters: `x.star` is `x.star` as described above, `x.base` is `x.base` as described above, `beta_posterior` is the list of posterior estimates of exposure regression coefficients (beta) from the model fit, `niter` is the total number of MCMC iterations, `nburn` is the number of burn-in iterations, `K` is the number of outcome categories, `refK` is the selected reference category. As described in Section 3 in [Hoskovec et al.], this method allows any category to be selected as the reference category. We select category 2 as the reference category in the example below.

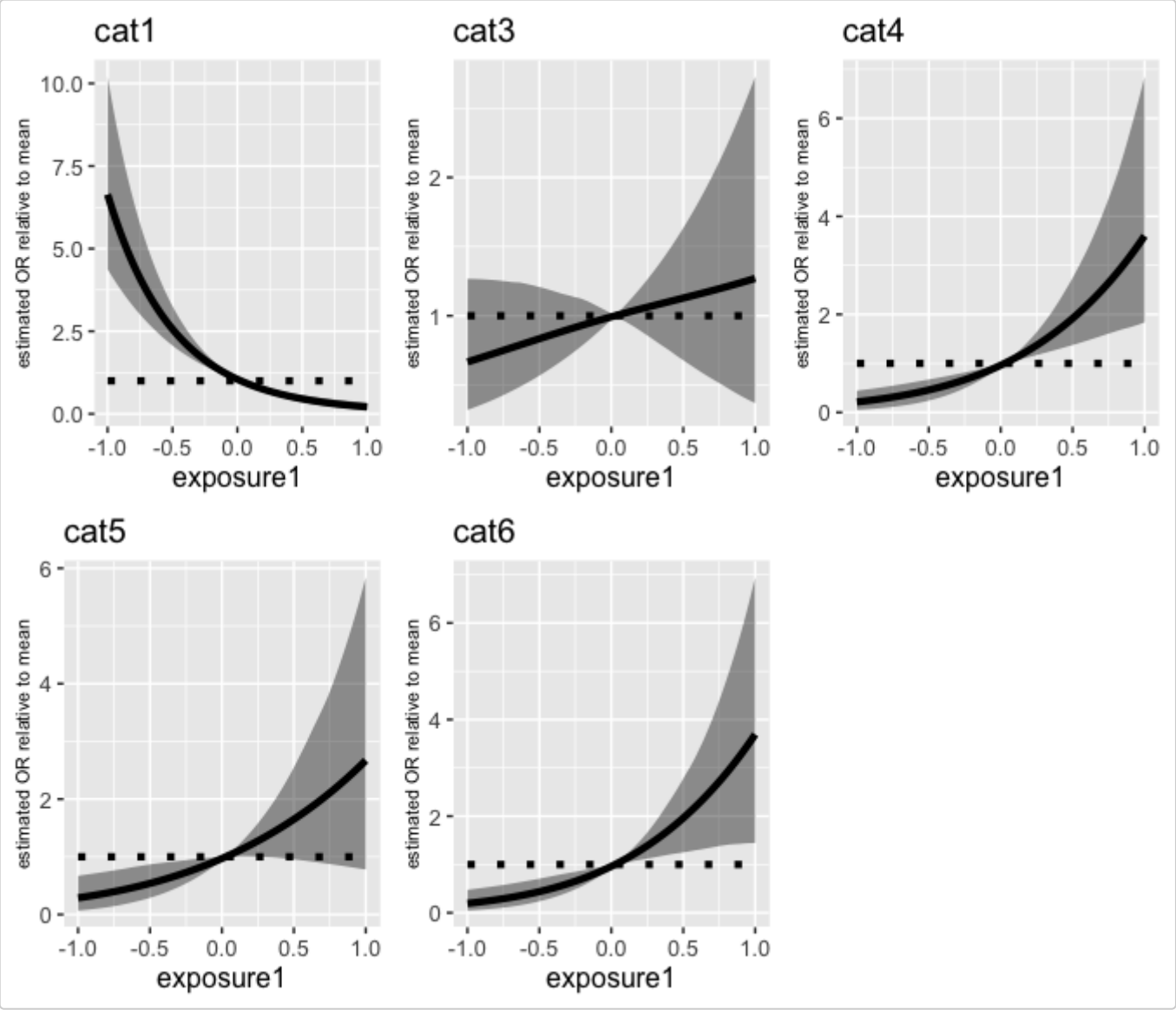
```
# calculate OR(x.star, x.base)
refK = 2
odds_ratio = get_odds_ratio(x.star = x.star, x.base = x.base,
                            beta_posterior = fit$beta,
                            niter = 1000, nburn = 500, K = 6,
                            refK = refK)
```

The object `odds_ratio` is a list of the posterior mean, .025 percentile, and .975 percentile of the odds ratio as a function of `x.star` and `x.base` for each of the $K-1$ categories relative to the reference category. We provide some code below to plot the posterior distribution of the estimated odds ratio as a function of exposure 1 relative to the mean of exposure 1, for each of the outcome categories, relative to the reference category.

```
# make data frame of odds ratios
or_df = data.frame(xvals = x.star[,1], matrix(unlist(odds_ratio), nrow = len))
# set column names
or_names = "xvals"
for(k in 1:(K-1)){
  k_names = c("lwr", "mean", "upr")
  or_names = c(or_names,k_names)
}
colnames(or_df) = or_names
# make K-1 plots of odds ratio for each category
plot_or = list()
for(k in 1:(K-1)){
  # get data for category k
  col_nums = 1 + (3*k-2):(3*k)
  df_k = or_df[,c(1,col_nums)]
```

```
  # plot OR for category k_num
  k_num = ((1:K)[-refK])[k]
  plotk = ggplot(data = df_k) + ggtitle(paste0("cat",k_num)) +
    geom_line(aes(x = xvals, y = mean), size = 1.5) +
    geom_ribbon(mapping =
                  aes(x = xvals, ymin = lwr, ymax = upr), fill = "black",
                alpha = .4, show.legend = FALSE) +
    geom_line(aes(x = xvals, y = 1), linetype = "dotted", size = 1.5) +
    scale_x_continuous(name = "exposure1") +
    scale_y_continuous(name = "estimated OR relative to mean") +
    theme(text = element_text(size = 12),
          axis.title.y = element_text(size = 8))
  plot_or[[k]] = plotk
}


# K-1 OR plots in a panel
plot_grid(
  plot_or[[1]] + theme(legend.position="none"),
  plot_or[[2]] + theme(legend.position="none"),
  plot_or[[3]] + theme(legend.position="none"),
  plot_or[[4]] + theme(legend.position="none"),
  plot_or[[5]] + theme(legend.position="none"),
  align = 'h',
  labels = "",
  hjust = -1,
  nrow = 2
)
```



Setting the Prior Distributions

The user can specify different prior distributions for the exposure and covariate regression coefficients. These are:

control parameter	model parameter	default value	description
betamean	$\mathbf{b}_k$	$\mathbf{0}$	mean parameter for multivariate Normal prior on $\boldsymbol{\beta}_k, k = 1, \ldots, K-1$
betavar	$\mathbf{B}_k$	$\mathbf{I}$	variance parameter for multivariate Normal prior on $\boldsymbol{\beta}_k, k = 1, \ldots, K-1$
gammamean	$\mathbf{g}_k$	$\mathbf{0}$	mean parameter for multivariate Normal prior on $\boldsymbol{\gamma}_k, k = 1, \ldots, K-1$

control parameter	model parameter	default value	description
gammavar	$\mathbf{G}_k$	$\mathbf{I}$	variance parameter for multivariate Normal prior on $\gamma_k, k = 1, \ldots, K-1$

Reproduce Simulation Study

The package includes the function `sim_study` to reproduce the simulation study in [Hoskovec et al.]. See Section 4 in [Hoskovec et al.] for more details on the simulation study. Here we demonstrate how to use `sim_study`.

The function `sim_study` takes in the following parameters: `simnum` is the simulation number (which is used to set a seed), `niter` is the total number of MCMC iterations to run the model, `nburn` is the number of burn-in iterations, `n` is the sample size, and `miss_prob` is the proportion of observations with missing outcome data. The three remaining parameters determine the simulation design as described above. To reiterate: 1) `allmiss` is logical; if TRUE, observations with missing outcome data are missing values for all K categories (fully missing), if FALSE, they are missing values for between 2 and K-1 categories (partially missing), 2) null_scenario is logical; if TRUE, the regression coefficients are all set to 0 (null), if FALSE, the regression coefficients are all simulated from independent standard normal distributions (signal), and 3) equal_probs is logical; if TRUE, category-specific intercepts are set so each category has roughly equal amounts of data (equal probabilities), if FALSE, intercepts are set to mimic the outcome category probabilities from the data analysis in [Hoskovec et al.] (data probabilities). Other parameters for the model fit are set to that described in Section 3 in [Hoskovec et al.].

Below we demonstrate running the simulation study for the scenario: data probabilities, partially missing outcomes, and a signal in the regression coefficients. We specify a missing data level of 50%. The function fits the `pgmultinom` model as described above and imputes data for missing outcomes. It also fits the model to the subset of complete cases.

```
sim_example = sim_study(simnum = 1, niter = 1000, nburn = 500, n = 1000,  miss_prob = 0.5,
                    allmiss = FALSE, null_scenario = FALSE, equal_probs = FALSE)
```

```
## [1] "simnum = 1"
## [1] 100
## [1] 200
## [1] 300
## [1] 400
## [1] 500
## [1] 600
## [1] 700
## [1] 800
## [1] 900
## [1] 1000
## [1] 100
## [1] 200
## [1] 300
## [1] 400
## [1] 500
## [1] 600
## [1] 700
## [1] 800
```

```
## [1] 900
## [1] 1000
```

The object `sim_example` is a list containing two elements: `results` contains the results of the proposed method with imputation and `complete_case_results` contains the results of the complete case analysis.

The object `sim_example$results` is a data frame containing (root mean squared error (RMSE), bias, 95% credible interval width (ci_width), and coverage of the 95% credible interval (cov) for estimated exposure (beta) and covariate (gamma) regression coefficients, as well as precision and recall for the 6 outcome categories.

```
round(sim_example$results,2)
```

```
##   RMSE_beta bias_beta ci_width_beta cov_beta RMSE_gamma bias_gamma
## 1      0.66      0.05          1.79     0.93       0.66      -0.02
##   ci_width_gamma cov_gamma precision1 precision2 precision3 precision4
## 1           1.66      0.93       0.91       0.69       0.41       0.23
##   precision5 precision6 recall1 recall2 recall3 recall4 recall5 recall6
## 1        0.5       0.18     0.9    0.69    0.41    0.14    0.59    0.27
```

The object `sim_example$complete_case_results` is a data frame containing (root mean squared error (RMSE), bias, 95% credible interval width (ci_width), and coverage of the 95% credible interval (cov) for estimated exposure (beta) and covariate (gamma) regression coefficients in the complete case analysis.

```
round(sim_example$complete_case_results,2)
```

```
##   RMSE_beta bias_beta ci_width_beta cov_beta RMSE_gamma bias_gamma
## 1      0.73      0.12          2.01        1       0.72       0.03
##   ci_width_gamma cov_gamma
## 1           1.89      0.97
```

As you can see, we obtain estimation gains from our proposed method over the complete case analysis as evidenced by smaller RMSE and credible interval width for exposure and covariate regression coefficients.