Journal of Statistical Software
MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

RprobitB: Bayes Estimation of Discrete Choice Behavior Heterogeneity via Probit Models in R

Lennart Oelschläger
Bielefeld University

Dietmar Bauer
Bielefeld University

Abstract

RprobitB is an R package for Bayes estimation of probit models with a special focus on modeling choice behavior heterogeneity. In comparison to competing packages it places a focus on approximating the mixing distribution via a latent mixture of Gaussian distributions and thereby providing a classification of deciders. It provides tools for data management, model estimation via Markov Chain Monte Carlo Simulation, diagnostics tools for the Gibbs sampling and a prediction function. This paper demonstrates the functionalities of **RprobitB** on known choice datasets and compares estimation results across packages.

Keywords: discrete choice, probit models, heterogeneity, Bayes estimation, R.

1. Introduction

The multinomial probit model is one of the most widely-used statistical models to explain the choices that individuals make among a discrete set of alternatives, which is of central interest in many scientific areas, for example in transportation and marketing. In many such choice scenarios it is reasonable to assume, that the preferences of the decision makers are non-homogeneous. Based on personal characteristics, deciders generally weight attributes like time and cost differently. Heterogeneity in choice behavior can be modeled using mixing distributions for the coefficients. Recently, Oelschlaeger and Bauer proposed a new instrument for approximating the underlying mixing distribution that combines Bayes estimation and semi-parametric methods. This paper presents the implementation of the methodology in the R package **RprobitB**.

Traditionally, discrete choice models are interpreted as random utility models, including the multinomial logit (MNL) and the multinomial probit (MNP) model as the most prominent members. The MNL model affords straightforward analysis but suffers from the well-known independence of irrelevant alternatives assumption. In contrast, the MNP model avoids this assumption, which however comes at the price of more complex parameter estimation, cf. Train (2009). In their basic form, these models often fail to take into account heterogeneity of individual deciders, cf. Train (2009), Chapter 6, or Train (2016). A concrete example of heterogeneous preferences is constituted by the value of travel time, cf. Cirillo and Axhausen (2006). Modeling heterogeneity in preferences is indispensable in such cases and has been elaborated in both the MNL and the MNP model by imposing mixing distributions on the coefficients, cf. Train (2009) and Bhat (2011).

Specifying these mixing distributions is an important part of the model selection. In absence of alternatives, it has been common practice so far to try different types of standard parametric distributions (including the normal, log-normal, uniform and tent distribution) and to perform a likelihood value-based model selection, cf. Train (2009), Chapter 6. Aiming to capture correlation patterns across parameters, Fountas, Anastasopoulos, and Abdel-Aty (2018) and Fountas, Pantangi, Hulme, and Anastasopoulos (2019) apply multivariate normal mixing distributions in their probit models, which however comes at the price of imposing the rather strong normality assumption on their parameters.

In order to alleviate these restrictions Train (2016) proposes a non-parametric approach based on grid methods. Building on the ideas of Train (2016) and Bhat and Lavieri (2018) recently Bauer, Büscher, and Batram (2019) introduced procedures for non-parametrically estimating latent class mixed multinomial probit models where the number of classes is chosen iteratively in the algorithm. These procedures have been demonstrated to be useful in reasonable sized cross-sectional data sets. However, for large panel data sets with a significant number of choice occasions per person, the approach is numerically extremely demanding in particular due to its non-parametric nature and has to deal with the curse of dimensionality.

In the Bayesian framework Scaccia and Marcucci (2010) presents the idea to estimate latent class logit models with a fixed prespecified number of Gaussian components. This approach does not require the maximization of the likelihood while at the same time it allows for approximation of the underlying mixing distribution. The same idea has also been applied to probit models, cf. Xiong and Mannering (2013) for an analysis of adolescent driver-injury data. In both cases however, the specification of the number of latent classes is based only on a trial-and-error strategy.

Oelschlaeger and Bauer presents a more flexible approach that combines the ideas of a Bayesian framework, approximating the mixing distribution through a mixture of normal distributions and updates on the number of latent classes within the algorithm analogously to Bauer *et al.* (2019). As a consequence, the procedure unites the benefits of a reduced numerical complexity for the estimation compared to the non-parametric likelihood maximization approach and the ability to approximate any mixing distribution. Presenting simulation results on artificial test cases, it is shown that the approach is capable of approximating the underlying mixing distributions and thereby guiding the specification of mixing distributions for real-world applications.

This packages adds to the line of discrete choice software packages in R in the following way: Its focus is entirely on Bayesian estimation, thereby it differs from the packages Rchoice. Furthermore, it places a focus on modeling choice behaviour heterogeneity by approximating the underlying mixing distribution through a latent mixture of normal distributions. The method is explained in detail in Oelschlaeger and Bauer.

In this article we present the methodology, give an overview over the functionality of the package and apply the package to data sets. Some of them were already analysed and we aim to reconstruct their findings. In addition, we added two datasets that are especially appropriate for **RprobitB** in modeling choice behaviour heterogeneits. The first one is a dataset of contraception choice from the German family panel pairfam. It contains repeated observations of males and femals over several years having different social demographics and relationship status choosing different means of contraception. This choice a priori can be considered to be very hetereogenous and dependent on factors not directly observable by the researcher. The second application deals with the opening choice of chess players depending on their and their openents playing strenght measured in the popular measure system Elo, their gender and nationality. Like the contraception example, this choice a priori can be considers to depend on psychological factors that are not directly observable by the researcher. By applying the functionality of this package we demonstrate how we are able to classify the players into different categories of playing style.

1. With **RprobitB**, you can model the choices made by deciders among a discrete set of alternatives. For example, think of tourists that want to book a flight to their holiday destination. The knowledge why they prefer a certain route over another is of great value for airlines, especially the customer's willingness to pay for say a faster or more comfortable flight alternative.

2. Different deciders value different choice attributes differently. For example, it is imaginable that business people place a higher value on flight time and are willing to pay more for a faster route alternative than vacationers. Such choice behavior heterogeneity can be addressed by **RprobitB**. Furthermore, the package enables to identify groups of deciders that share similar preferences.

3. Finally, the package enables prediction of choice behavior when certain choice attributes change, for example the proportion of customers who will choose the competitor's product in the event of a price increase.

The functions of **RprobitB** can be grouped into ones for data management, model fitting, and model evaluation, see the flowchart below. The package can be used for two different purposes: (a) estimation of a model for given data and (b) estimation of a model for simulated data. Simulation typically serves to assess the properties of estimation algorithms either for research or in a bootstrap like fashion. **RprobitB** supports these functions.

2. The probit model

The probit model is a regression-type model where the dependent variable only takes a finite number of values and the error term is normally distributed (Agresti 2015). Its purpose is to estimate the probability that the dependent variable takes a certain, discrete value. The model's most popular application are discrete choice scenarios, in which the dependent variable is one of finitely many and mutually exclusive alternatives (that are not ordered), and explanatory variables typically are characteristics of the deciders or the alternatives. This section defines the model via latent utilities, outlines an extension for modeling heterogeneity, and discusses necessary normalization for parameter identification.

2.1. Latent utilities

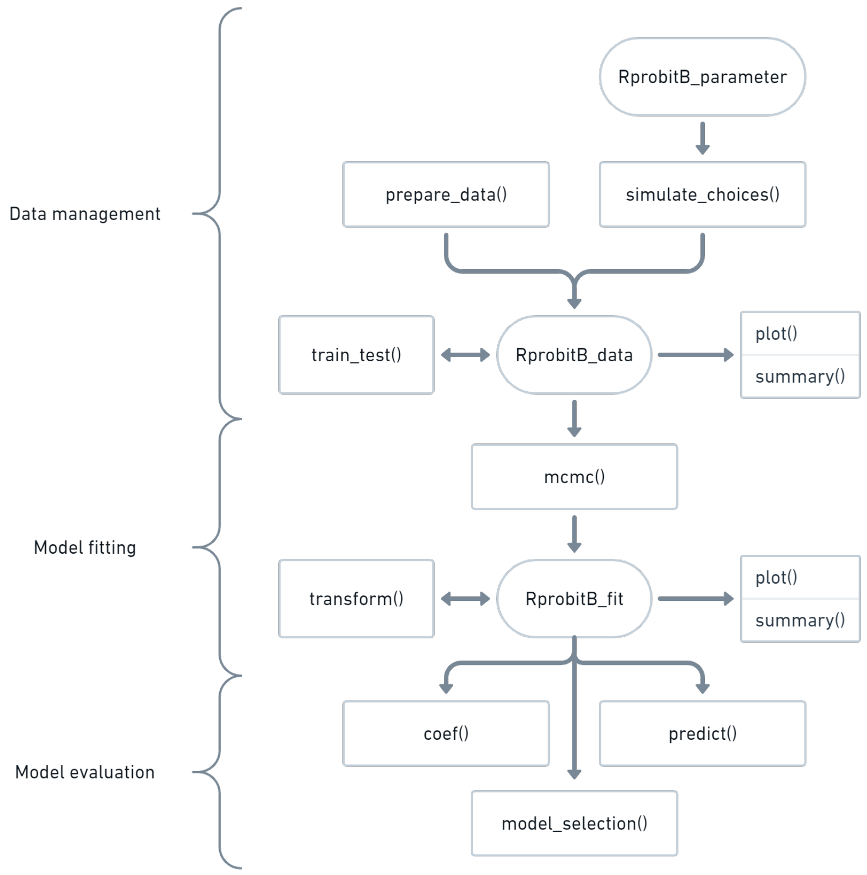


Figure 1: Flowchart of the package functionality.

Assume that we possess data of N decision makers which choose between $J \geq 2$ alternatives at each of T choice occasions.[1] Specific to each decision maker, alternative and choice occasion, we furthermore observe P choice attributes that we use to explain the choices. The choice attributes cannot be linked directly to the discrete choices but must take a detour over a latent variable. In the discrete choice setting, this variable can be interpreted as the decider's utility of a certain alternative. Decider n's utility U_{ntj} for alternative j at choice occasion t is modeled as

$$U_{ntj} = X'_{ntj}\beta + \epsilon_{ntj} \tag{1}$$

for $n = 1, \dots, N$, $t = 1, \dots, T$ and $j = 1, \dots, J$. Here, X_{ntj} is a (column) vector of P characteristics of j as faced by n at t, $\beta \in \mathbb{R}^P$ is a vector of coefficients, and $(\epsilon_{nt:}) = (\epsilon_{nt1}, \dots, \epsilon_{ntJ})' \sim \text{MVN}_J(0, \Sigma)$ is the model's error term vector for n at t, which in the probit model is assumed to be multivariate normally distributed with zero mean and covariance matrix Σ.[2]

[1]For notational simplicity, the number of choice occasions T is assumed to be the same for each decision maker here. However, we are not restricted to this case: **RprobitB** allows for unbalanced panels, i.e. varying T. Of course, the cross-sectional case $T = 1$ is possible.

[2]The assumption about the error-term distribution distinguishes the probit from the logit model. In the

Now let $y_{nt} = j$ denote the event that decision maker n chooses alternative j at choice occasion t. Assuming utility maximizing behavior of the decision makers, the decisions are linked to the utilities via

$$y_{nt} = \arg\max_{j=1,\dots,J} U_{ntj}. \tag{2}$$

2.2. Choice behavior heterogeneity

Note that the coefficient vector β in equation 1 is constant across decision makers. This assumption is too restrictive for many applications.[3] Relaxing this assumption, heterogeneity in choice behavior can be modeled by imposing a distribution on β such that each decider can have their own preferences.

Formally, we define $\beta = (\alpha, \beta_n)$, where α are P_f coefficients that are constant across deciders and β_n are P_r decider-specific coefficients. Consequently, $P = P_f + P_r$. Now if $P_r > 0$, β_n is distributed according to some P_r-variate distribution, the so-called mixing distribution.

Choosing an appropriate mixing distribution is a notoriously difficult task of the model specification. In many applications, different types of standard parametric distributions (including the normal, log-normal, uniform and tent distribution) are tried in conjunction with a likelihood value-based model selection, cf. Train (2009), Chapter 6. Instead, **RprobitB** implements the approach of Oelschläger and Bauer (2020) to approximate any underlying mixing distribution by a mixture of (multivariate) Gaussian densities. More precisely, the underlying mixing distribution for the random coefficients $(\beta_n)_n$ is approximated by a mixture of P_r-variate normal densities ϕ_{P_r} with mean vectors $b = (b_c)_c$ and covariance matrices $\Omega = (\Omega_c)_c$ using C components:

$$\beta_n \mid b, \Omega \sim \sum_{c=1}^{C} s_c \phi_{P_r}(\cdot \mid b_c, \Omega_c). \tag{3}$$

Here, $(s_c)_c$ are weights satisfying $0 < s_c \leq 1$ for $c = 1, \dots, C$ and $\sum_c s_c = 1$. One interpretation of the latent class model is obtained by introducing variables $z = (z_n)_n$, allocating each decision maker n to class c with probability s_c, i.e.

$$\text{Prob}(z_n = c) = s_c \wedge \beta_n \mid z, b, \Omega \sim \phi_{P_r}(\cdot \mid b_{z_n}, \Omega_{z_n}). \tag{4}$$

This interpretation enables a classification of the deciders in terms of their preferences, see Section ... for an example.

2.3. Model normalization

Any utility model is invariant towards the level and the scale of utility, as Train (2009) points out. Therefore, we consider the normalized model

$$\tilde{U}_{ntj} = \tilde{X}'_{ntj}\beta + \tilde{\epsilon}_{ntj}, \tag{5}$$

latter, each ϵ_{nti} is assumed to be independently extreme value distributed.

[3]For example, consider the case of modeling the choice of a means of transportation to work: It is easily imaginable that business people and pensioners do not share the same sensitivities towards cost and time.

$n = 1, \dots, N$, $t = 1, \dots, T$ and $j = 1, \dots, J-1$, where (choosing J as the reference alternative) $\tilde{U}_{ntj} = U_{ntj} - U_{ntJ}$, $\tilde{X}_{ntj} = X_{ntj} - X_{ntJ}$, and $\tilde{\epsilon}_{ntj} = \epsilon_{ntj} - \epsilon_{ntJ}$, where $(\tilde{\epsilon}_{nt:}) = (\tilde{\epsilon}_{nt1}, ..., \tilde{\epsilon}_{nt(J-1)})' \sim \text{MVN}_{J-1}(0, \tilde{\Sigma})$ and $\tilde{\Sigma}$ denotes a covariance matrix with the top-left element restricted to one.[4]

3. Choice data

This section formulates data requirements and introduces the formula framework of **RprobitB** for specifying a model equation. Subsequently, we demonstrate how to pass a formula to either `prepare_data()` (for preparing an empirical data set) or `simulate_choices()` (for simulating choices) with the help of examples.

3.1. Data requirements

Choice data sets typically consist of the observed choices in conjunction with choice characteristics.[5] The following formal requirements are requested by **RprobitB**: Data sets must be (a) of class '`data.frame`', (b) in wide format (that means each row provides the full information for one choice occasion), (c) contain a column with unique identifiers for each decision maker[6], (d) contain a column with the observed choices (required for model fitting but not for prediction), and (e) contain columns for the values of each alternative specific covariate for each alternative and for each decider specific covariate. The underlying set of choice alternatives is assumed to be mutually exclusive (one can choose one and only one alternative that are all different), exhaustive (the alternatives do not leave other options open), and finite (Train 2009).

3.2. Model formula

We have to inform **RprobitB** about the covariates we want to include in our model via specifying a `formula` object. Say we want to model the utility $U_{n,t,j}$ of decider n at choice occasion t for alternative j via the linear equation

$$U_{n,t,j} = A_{n,t,j}\beta_1 + B_{n,t}\beta_{2,j} + C_{n,t,j}\beta_{3,j} + \epsilon_{n,tj}.$$

Here, A and C are alternative and choice situation specific covariates, whereas B is choice situation specific. The coefficient β_1 is generic (i.e. the same for each alternative), whereas $\beta_{2,j}$ and $\beta_{3,j}$ are alternative specific.

To represent this structure, the `formula` object is of the form (analogously to mlogit) `choice ~ A | B | C`, where

- `choice` is the dependent variable (the discrete choice we aim to explain),

[4]**RprobitB** provides an alternative to fixing an error term variance in order to normalize with respect to scale by fixing an element of α, see Section ...

[5]In **RprobitB** only unordered alternatives (that is, alternatives cannot be ranked) are modeled. Every decider may take one or repeated choices (called choice occasions).

[6]Additionally, it can contain a column with identifier for each choice situation of each decider. If this information is missing, these identifier are generated automatically by the appearance of the choices in the data set.

- `A` are alternative and choice situation specific covariates with a generic coefficient (we call them covariates of type 1),

- `B` are choice situation specific covariates with alternative specific coefficients. Mind that not all alternative specific coefficients of type 2-covariates are identified. This is because the probit model is estimated on utility differences since the level of the utility is irrelevant, see the vignette on the model definition. Therefore, the coefficient of the last alternative of each type 2-covariate is set to 0. (we call them covariates of type 2),

- and `C` are alternative and choice situation specific covariates with alternative specific coefficients (we call them covariates of type 3).

Specifying a `formula` object for **RprobitB** must be consistent with the following rules:

- By default, alternative specific constants (ASCs) ASCs capture the average effect on utility of all factors that are not included in the model. Due to identifiability, we cannot estimate ASCs for all the alternatives. Therefore, they are added for all except for the last alternative. are added to the model. They can be removed by adding `+ 0` in the second spot, e.g. `choice ~ A | B + 0 | C`.

- To exclude covariates of the backmost categories, use either `0`, e.g. `choice ~ A | B | 0` or just leave this part out and write `choice ~ A | B`. However, to exclude covariates of front categories, we have to use `0`, e.g. `choice ~ 0 | B`.

- To include more than one covariate of the same category, use `+`, e.g. `choice ~ A1 + A2 | B`.

- If we don't want to include any covariates of the second category but we want to estimate alternative specific constants, add `1` in the second spot, e.g. `choice ~ A | 1`. The expression `choice ~ A | 0` is interpreted as no covariates of the second category and no alternative specific constants.

To impose random effects for specific variables, we need to define a character vector `re` with the corresponding variable names. To have random effects for the alternative specific constants, include `"ASC"` in `re`.

3.3. The prepare data function

Before model estimation with **RprobitB**, any empirical choice data set `choice_data` must pass the `prepare_data()` function:

```
> data <- prepare_data(form = form, choice_data = choice_data)
```

The function performs compatibility checks and data transformations and returns an object of class '`RprobitB_data`' that can be fed into the estimation routine `mcmc()`. The following arguments are optional:

- `re`: The character vector of variable names of `form` with random effects. `re = NULL` per default, i.e. no random effects.

- `alternatives`: We may not want to consider all alternatives in `choice_data`. In that case, we can specify a character vector `alternatives` with selected names of alternatives. If not specified, the choice set is defined by the observed choices.

- `id`: A character (single string), the name of the column in `choice_data` that contains a unique identifier for each decision maker. The default is `"id"`.

- `idc`: A character, the name of the column in `choice_data` that contains a unique identifier for each choice situation given the decision maker. Per default, these identifier are generated by the appearance of the choices in the data set.

- `standardize`: A character vector of variable names of `form` that get standardized. Covariates of type 1 or 3 have to be addressed by `<covariate>_<alternative>`. If `standardize = "all"`, all covariates get standardized. Per default, no covariate is standardized.

- `impute`: Specifies how to handle missing entries (`NA, NaN, -Inf, Inf`) in `choice_data`. The following options are available:

- `"complete_cases"`, which removes rows containing missing entries (the default),

- `"zero_out"`, which replaces missing entries by zero,

- `"mean"`, which imputes missing entries by the covariate mean.

3.4. Example 1: Train data

The Train data set contains 2929 stated choices by 235 Dutch individuals deciding between two virtual train trip options based on the price, the travel time, the level of comfort, and the number of changes. It fulfills the above requirements: Each row represents one choice occasion, the columns `id` and `choiceid` identify the deciders and the choice occasions, respectively. The column `choice` gives the observed choices. Four alternative-specific covariates are available, namely `price`, `time`, `change`, and `comfort`. There values are given for each alternative. For alternative specific variables, the alternative names must be added to the covariates via the `_` separator.

```
> data("Train", package = "mlogit")
> str(Train)
```

```
'data.frame':        2929 obs. of  11 variables:
 $ id       : int  1 1 1 1 1 1 1 1 1 1 ...
 $ choiceid : int  1 2 3 4 5 6 7 8 9 10 ...
 $ choice   : Factor w/ 2 levels "A","B": 1 1 1 2 2 2 2 2 1 1 ...
 $ price_A  : num  2400 2400 2400 4000 2400 4000 2400 2400 4000 2400 ...
 $ time_A   : num  150 150 115 130 150 115 150 115 115 150 ...
 $ change_A : num  0 0 0 0 0 0 0 0 0 0 ...
 $ comfort_A: num  1 1 1 1 1 0 1 1 0 1 ...
 $ price_B  : num  4000 3200 4000 3200 3200 2400 3200 3200 3200 4000 ...
 $ time_B   : num  150 130 115 150 150 130 115 150 130 115 ...
 $ change_B : num  0 0 0 0 0 0 0 0 0 0 ...
 $ comfort_B: num  1 1 0 0 0 0 1 0 1 0 ...
```

We specify a model formula for the Train data set. Say we want to include all the covariates `price`, `time`, `comfort`, and `change`, which are all alternative specific (that is, they contain a potentially different value for each alternative, such as different prices for A and B), so either of type 1 or type 3. The difference between type 1 and type 3 is that in the former case, we would estimate a generic coefficient (i.e. a coefficient that is constant across alternatives), whereas in the latter case, we would estimate alternative specific coefficients. Deciding between type 1 and type 3 for these covariates belongs into the topic of model selection, for which we provide a separate vignette. For now, we go with type 1 for all covariates and remove ASCs:

```
> form <- choice ~ price + time + comfort + change | 0
```

Additionally, we specify random effects for `price` and `time` (because we would typically expect heterogeneity here):

```
> re <- c("price","time")
```

RprobitB provides the function `check_form()` which can be used to check if `form` and `re` are correctly interpreted:

```
> check_form(form = form, re = re)
```

```
choice ~ price + time + comfort + change | 0
- dependent variable: choice
- type 1 covariate(s): price, time, comfort, change
- type 2 covariate(s):
- type 3 covariate(s):
- random effects: price, time
- ASC: FALSE
```

Let's prepare the Train data set for estimation with our previous specification of `form` and `re`:

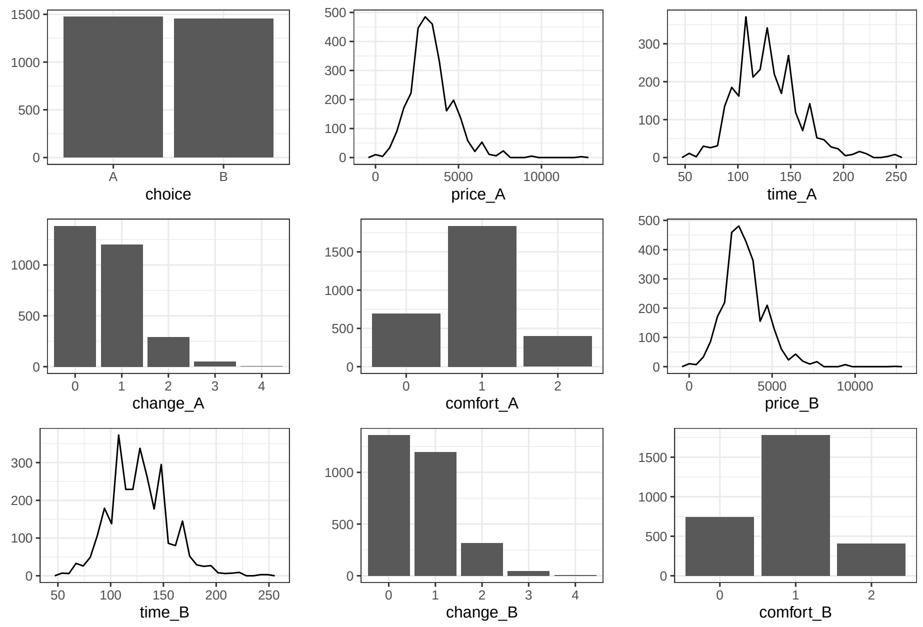
```
> data <- prepare_data(form = form, choice_data = Train, re = re, id = "id", idc = "choice
```

The `summary()` and `plot()` methods provide a quick data overview:

```
> summary(data)
```

```
Summary of empirical choice data
235 decision makers
5 to 19 choice occasions each
2929 choices in total

Alternatives
  frequency
A      1474
B      1455

Linear coefficients
     name    re
1 comfort FALSE
2  change FALSE
3   price  TRUE
4    time  TRUE
```

```
> plot(data)
```



3.5. Simulate choices

The `simulate_choices()` function simulates discrete choice data from a prespecified probit model. Say we want to simulate the choices of `N` deciders in `T` choice occasions (`T` can be either a positive number, representing a fixed number of choice occasions for each decision maker, or a vector of length `N` with decision maker specific numbers of choice occasions) among `J` alternatives from a model formulation `form`, we have to call

```
> data <- simulate_choices(form = form, N = N, T = T, J = J)
```

The function `simulate_choices()` has the following optional arguments:

- `re`: The character vector of variable names of `form` with random effects.

- `alternatives`: A character vector of length `J` with the names of the choice alternatives. If not specified, the alternatives are labeled by the first `J` upper-case letters of the Roman alphabet.

- `covariates`: A named list of covariate values. Each element must be a vector of length equal to the number of choice occasions and named according to a covariate, or follow the naming convention for alternative specific covariates, i.e. `<covariate>_<alternative>`. Covariates for which no values are specified are drawn from a standard normal distribution.

- `standardize`: A character vector of variable names of `form` that get standardized.

- `seed`: Set a seed for the simulation.

We can specify the true parameters by adding values for

- `alpha`, the fixed coefficient vector,

- `C`, the number (greater or equal 1) of latent classes of decision makers,

- `s`, the vector of class weights,
- `b`, the matrix of class means as columns,
- `Omega`, the matrix of class covariance matrices as columns,
- `Sigma`, the differenced error term covariance matrix, or `Sigma_full`, the full error term covariance matrix,
- `beta`, the matrix of the decision-maker specific coefficient vectors,
- `z`, the class allocation vector.

True parameters that are not specified will be set at random.

3.6. Example 2: Simulated choices

For illustration, we simulate the choices of `N = 100` deciders at `T = 10` choice occasions between the alternatives `A` and `B`:

```
> N <- 100
> T <- 10
> alternatives <- c("A", "B")
> form <- choice ~ var1 | var2 | var3
> re <- c("ASC", "var2")
```

RprobitB provides the function `overview_effects()` which can be used to get an overview of the effects for which parameters can be specified:

```
> overview_effects(form = form, re = re, alternatives = alternatives)
```

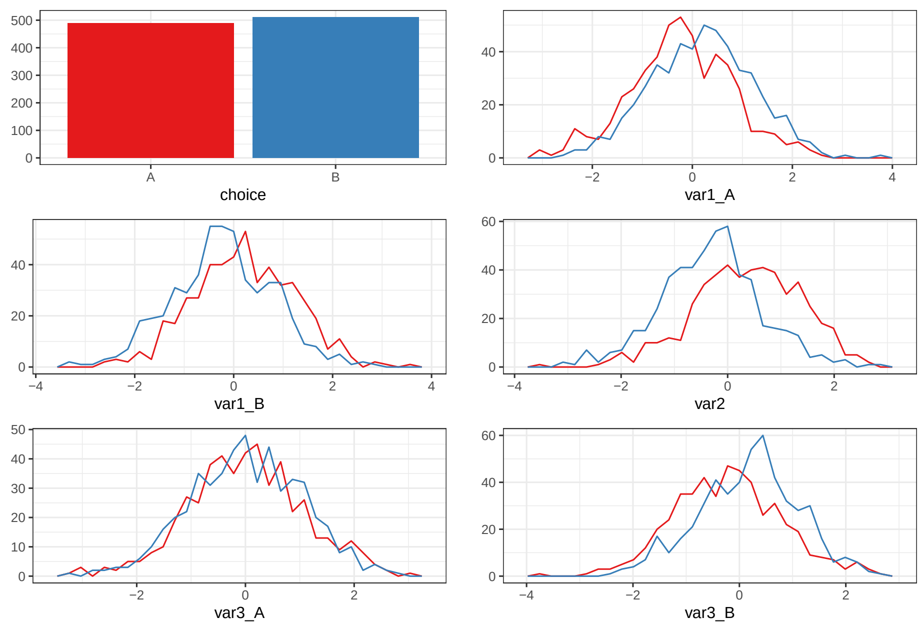
```
    name    re
1   var1 FALSE
2 var3_A FALSE
3 var3_B FALSE
4 var2_A  TRUE
5  ASC_A  TRUE
```

Hence, the coefficient vector `alpha` must be of length 3, where the elements 1 to 3 correspond to `var1`, `var3_A`, and `var3_B`, respectively. The matrix `b` must be of dimension `2 x C`, where (by default) `C = 1` and row 1 and 2 correspond to `var2_A` and `ASC_A`, respectively.

```
> data <- simulate_choices(
+    form = form,
+    N = N,
+    T = T,
+    J = 2,
+    re = re,
+    alternatives = alternatives,
+    seed = 1,
+    alpha = c(-1,0,1),
+     b = matrix(c(2,-0.5), ncol = 1)
+   )
> summary(data)

Summary of simulated choice data
100 decision makers
10 choice occasions each
1000 choices in total

Alternatives
  frequency
A        489
B        511

Linear coefficients
    name    re
1   var1 FALSE
2 var3_A FALSE
3 var3_B FALSE
4 var2_A  TRUE
5  ASC_A  TRUE
```

We can visualize the covariates grouped by the chosen alternatives:

```
> plot(data, by_choice = TRUE)
```



What we see is consistent with our specification: Higher values of `var1_A` for example correspond more frequently to choice `B` (upper-right panel), because the coefficient of `var1` (the first value of `alpha`) is negative.

3.7. Train and test data set

The function `train_test()` can be used to split the output of `prepare_data()` or `simulate_choices()` into a train and a test subset. This is useful when evaluating the prediction performance of a fitted model. For example, the following code puts 70% of deciders from our simulated `data` into the train subsample and 30% of deciders in the test subsample:

```
> train_test(data, test_proportion = 0.3, by = "N")

$train
Simulated data of 700 choices.

$test
Simulated data of 300 choices.
```

Alternatively, the following code puts 2 randomly chosen choice occasions per decider from `data` into the test subsample, the rest goes into the train subsample:

```
> train_test(data, test_number = 2, by = "T", random = TRUE, seed = 1)

$train
Simulated data of 800 choices.

$test
Simulated data of 200 choices.
```

4. Model fitting

Bayes estimation of the probit model builds upon the work of McCulloch and Rossi (1994), Nobile (1998), Allenby and Rossi (1998), and Imai and van Dyk (2005). A key ingredient is the concept of data augmentation, see Albert and Chib (1993): The idea is to treat the latent utilities U in the model equation $U = X\beta + \epsilon$ as additional parameters. Then, conditional on U, the probit model constitutes a standard Bayesian linear regression set-up. Its posterior distribution can be approximated by iteratively drawing and updating each model parameter conditional on the other parameters (the so-called Gibbs sampling approach).

A priori, we assume the following (conjugate) parameter distributions:

- $(s_1, \dots, s_C) \sim D_C(\delta)$, where $D_C(\delta)$ denotes the C-dimensional Dirichlet distribution with concentration parameter vector $\delta = (\delta_1, \dots, \delta_C)$,

- $\alpha \sim \text{MVN}_{P_f}(\psi, \Psi)$, where MVN_{P_f} denotes the P_f-dimensional normal distribution with mean ψ and covariance Ψ,

- $b_c \sim \text{MVN}_{P_r}(\xi, \Xi)$, independent for all c,

- $\Omega_c \sim W^{-1}_{P_r}(\nu, \Theta)$, independent for all c, where $W^{-1}_{P_r}(\nu, \Theta)$ denotes the P_r-dimensional inverse Wishart distribution with ν degrees of freedom and scale matrix Θ,

- and $\Sigma \sim W^{-1}_{J-1}(\kappa, \Lambda)$.

These prior distributions imply the following conditional posterior distributions:

- The class weights are drawn from the Dirichlet distribution

$$(s_1, \dots, s_C) \mid \delta, z \sim D_C(\delta_1 + m_1, \dots, \delta_C + m_C), \tag{6}$$

where for $c = 1, \dots, C$, $m_c = \#\{n : z_n = c\}$ denotes the current absolute class size. Mind that the model is invariant to permutations of the class labels $1, \dots, C$. For that reason, we accept an update only if the ordering $s_1 > \dots > s_C$ holds, thereby ensuring a unique labeling of the classes.

- Independently for all n, we update the allocation variables $(z_n)_n$ from their conditional distribution

$$\text{Prob}(z_n = c \mid s, \beta, b, \Omega) = \frac{s_c \phi_{P_r}(\beta_n \mid b_c, \Omega_c)}{\sum_c s_c \phi_{P_r}(\beta_n \mid b_c, \Omega_c)}. \tag{7}$$

- The class means $(b_c)_c$ are updated independently for all c via

$$b_c \mid \Xi, \Omega, \xi, z, \beta \sim \text{MVN}_{P_r}\left(\mu_{b_c}, \Sigma_{b_c}\right), \tag{8}$$

where $\mu_{b_c} = (\Xi^{-1} + m_c \Omega_c^{-1})^{-1}(\Xi^{-1}\xi + m_c \Omega_c^{-1}\bar{b}_c)$, $\Sigma_{b_c} = (\Xi^{-1} + m_c \Omega_c^{-1})^{-1}$, $\bar{b}_c = m_c^{-1} \sum_{n:z_n=c} \beta_n$.

- The class covariance matrices $(\Omega_c)_c$ are updated independently for all c via

$$\Omega_c \mid \nu, \Theta, z, \beta, b \sim W_{P_r}^{-1}(\mu_{\Omega_c}, \Sigma_{\Omega_c}), \tag{9}$$

where $\mu_{\Omega_c} = \nu + m_c$ and $\Sigma_{\Omega_c} = \Theta^{-1} + \sum_{n:z_n=c}(\beta_n - b_c)(\beta_n - b_c)'$.

- Independently for all n and t and conditionally on the other components, the utility vectors $(U_{nt:})$ follow a $J-1$-dimensional truncated multivariate normal distribution, where the truncation points are determined by the choices y_{nt}. To sample from a truncated multivariate normal distribution, we apply a sub-Gibbs sampler, following the approach of Geweke (1998):

$$U_{ntj} \mid U_{nt(-j)}, y_{nt}, \Sigma, W, \alpha, X, \beta \sim \mathcal{N}(\mu_{U_{ntj}}, \Sigma_{U_{ntj}}) \cdot \begin{cases} 1(U_{ntj} > \max(U_{nt(-j)}, 0)) & \text{if } y_{nt} = j \\ 1(U_{ntj} < \max(U_{nt(-j)}, 0)) & \text{if } y_{nt} \neq j \end{cases}, \tag{10}$$

where $U_{nt(-j)}$ denotes the vector $(U_{nt:})$ without the element U_{ntj}, $\mathcal{N}$ denotes the univariate normal distribution, $\Sigma_{U_{ntj}} = 1/(\Sigma^{-1})_{jj}$ and

$$\mu_{U_{ntj}} = W_{ntj}'\alpha + X_{ntj}'\beta_n - \Sigma_{U_{ntj}}(\Sigma^{-1})_{j(-j)}(U_{nt(-j)} - W_{nt(-j)}'\alpha - X_{nt(-j)}'\beta_n), \tag{11}$$

where $(\Sigma^{-1})_{jj}$ denotes the (j,j)th element of Σ^{-1}, $(\Sigma^{-1})_{j(-j)}$ the jth row without the jth entry, $W_{nt(-j)}$ and $X_{nt(-j)}$ the coefficient matrices W_{nt} and X_{nt}, respectively, without the jth column.

- Updating the fixed coefficient vector α is achieved by applying the formula for Bayesian linear regression of the regressors W_{nt} on the regressands $(U_{nt:}) - X_{nt}'\beta_n$, i.e.

$$\alpha \mid \Psi, \psi, W, \Sigma, U, X, \beta \sim \text{MVN}_{P_f}(\mu_\alpha, \Sigma_\alpha), \tag{12}$$

where $\mu_\alpha = \Sigma_\alpha(\Psi^{-1}\psi + \sum_{n=1,t=1}^{N,T} W_{nt}\Sigma^{-1}((U_{nt:}) - X_{nt}'\beta_n))$ and $\Sigma_\alpha = (\Psi^{-1} + \sum_{n=1,t=1}^{N,T} W_{nt}\Sigma^{-1}W_{nt}')^{-1}$.

- Analogously to α, the random coefficients $(\beta_n)_n$ are updated independently via

$$\beta_n \mid \Omega, b, X, \Sigma, U, W, \alpha \sim \text{MVN}_{P_r}(\mu_{\beta_n}, \Sigma_{\beta_n}), \tag{13}$$

where $\mu_{\beta_n} = \Sigma_{\beta_n}(\Omega_{z_n}^{-1} b_{z_n} + \sum_{t=1}^{T} X_{nt}\Sigma^{-1}(U_{nt} - W_{nt}'\alpha))$ and $\Sigma_{\beta_n} = (\Omega_{z_n}^{-1} + \sum_{t=1}^{T} X_{nt}\Sigma^{-1}X_{nt}')^{-1}$.

- The error term covariance matrix Σ is updated by means of

$$\Sigma \mid \kappa, \Lambda, U, W, \alpha, X, \beta \sim W_{J-1}^{-1}(\kappa + NT, \Lambda + S), \tag{14}$$

where $S = \sum_{n=1,t=1}^{N,T} \varepsilon_{nt}\varepsilon_{nt}'$ and $\varepsilon_{nt} = (U_{nt:}) - W_{nt}'\alpha - X_{nt}'\beta_n$.

4.1. Parameter normalization

Samples obtained from the updating scheme described above lack identification (except for s and z draws), compare to the vignette on the model definition. Therefore, subsequent to the sampling, the following normalization is required for the ith update in each iteration i:
- $\alpha^{(i)} \cdot \omega^{(i)}$,
- $b_c^{(i)} \cdot \omega^{(i)}$, $c = 1, \dots, C$,
- $U_{nt}^{(i)} \cdot \omega^{(i)}$, $n = 1, \dots, N$, $t = 1, \dots, T$,
- $\beta_n^{(i)} \cdot \omega^{(i)}$, $n = 1, \dots, N$,
- $\Omega_c^{(i)} \cdot (\omega^{(i)})^2$, $c = 1, \dots, C$, and
- $\Sigma^{(i)} \cdot (\omega^{(i)})^2$,

where either $\omega^{(i)} = \sqrt{\text{const}/(\Sigma^{(i)})_{jj}}$ with $(\Sigma^{(i)})_{jj}$ the jth diagonal element of $\Sigma^{(i)}$, $1 \leq j \leq J-1$, or alternatively $\omega^{(i)} = \text{const}/\alpha_p^{(i)}$ for some coordinate $1 \leq p \leq P_f$ of the ith draw for the coefficient vector α. Here, const is any positive constant (typically 1). The preferences will be flipped if $\omega^{(i)} < 0$, which only is the case if $\alpha_p^{(i)} < 0$.

4.2. Burn-in and thinning

The theory behind Gibbs sampling constitutes that the sequence of samples produced by the updating scheme is a Markov chain with stationary distribution equal to the desired joint posterior distribution. It takes a certain number of iterations for that stationary distribution to be approximated reasonably well. Therefore, it is common practice to discard the first B out of R samples (the so-called burn-in period). Furthermore, correlation between nearby samples should be expected. In order to obtain independent samples, we consider only every Qth sample when computing Gibbs sample statistics like expectation and standard deviation. The independence of the samples can be verified by computing the serial correlation and the convergence of the Gibbs sampler can be checked by considering trace plots, see below.

4.3. The mcmc function

The Gibbs sampling scheme described above can be executed by applying the function

```
> mcmc(data = data)
```

where `data` must be an ‘`RprobitB_data`’ object (see the vignette about choice data). The function has the following optional arguments:

- `scale`: A named list of three elements, determining the parameter normalization:

- `parameter`: Either `"a"` (for an element of `alpha`) or `"s"` (for a diagonal element of `Sigma`). Use the `overview_effects()` function to determine the parameter index. Note that you can set `"parameter" = "a"` only if the model has parameters with a fixed coefficient, i.e. only if `P_f>0`.

- `index`: The index of the parameter that gets fixed.

- `value`: The value for the fixed parameter (i.e. the value const introduced above.

Per default, the first error-term variance is fixed to 1, i.e. `scale = list("parameter" = "s", "index" = 1, "value" = 1)`.

- `R`: The number of iterations of the Gibbs sampler. The default is `R = 10000`.

- `B`: The length of the burn-in period, i.e. a non-negative number of samples to be discarded. The default is `B = R/2`.

- `Q`: The thinning factor for the Gibbs samples, i.e. only every `Q`th sample is kept. The default is `Q = 1`.

- `print_progress`: A boolean, determining whether to print the Gibbs sampler progress.

- `prior`: A named list of parameters for the prior distributions (their default values are documented in the `check_prior()` function):

- `eta`: The mean vector of length `P_f` of the normal prior for `alpha`.

- `Psi`: The covariance matrix of dimension `P_f x P_f` of the normal prior for `alpha`.

- `delta`: The concentration parameter of length 1 of the Dirichlet prior for `s`.

- `xi`: The mean vector of length `P_r` of the normal prior for each `b_c`.

- `D`: The covariance matrix of dimension `P_r` x `P_r` of the normal prior for each `b_c`.

- `nu`: The degrees of freedom (a natural number greater than `P_r`) of the Inverse Wishart prior for each `Omega_c`.

- `Theta`: The scale matrix of dimension `P_r` x `P_r` of the Inverse Wishart prior for each `Omega_c`.

- `kappa`: The degrees of freedom (a natural number greater than `J-1`) of the Inverse Wishart prior for `Sigma`.

- `E`: The scale matrix of dimension `J-1` x `J-1` of the Inverse Wishart prior for `Sigma`.

- `latent_classes`: A list of parameters specifying the number and the updating scheme of latent classes, see the vignette on modeling heterogeneity fitting.

4.4. Example 1: Train (cont.)

In the previous vignette on choice data we introduced the Train data set from the mlogit package (Croissant 2020) that contains 2922 choices between two fictional train route alternatives. First, we transform the travel `time` from minutes to hours and the travel `price` from guilders to euros:

```
> data("Train", package = "mlogit")
> Train$price_A <- Train$price_A / 100 * 2.20371
> Train$price_B <- Train$price_B / 100 * 2.20371
> Train$time_A <- Train$time_A / 60
> Train$time_B <- Train$time_B / 60
```

The following lines fit a probit model that explains the chosen trip alternatives (`choice`) by their `price`, `time`, number of `change` s, and level of `comfort` (the lower this value the higher the comfort). For normalization, the first linear coefficient, the `price`, was fixed to `-1`, which allows to interpret the other coefficients as monetary values:

```
> form <- choice ~ price + time + change + comfort | 0
> data <- prepare_data(form = form, choice_data = Train)
> model_train <- mcmc(
+    data = data,
+    scale = list("parameter" = "a", index = 1, value = -1)
+  )
```

The estimated model is saved in **RprobitB** and can be accessed via

```
> data(model_train, package = "RprobitB")
```

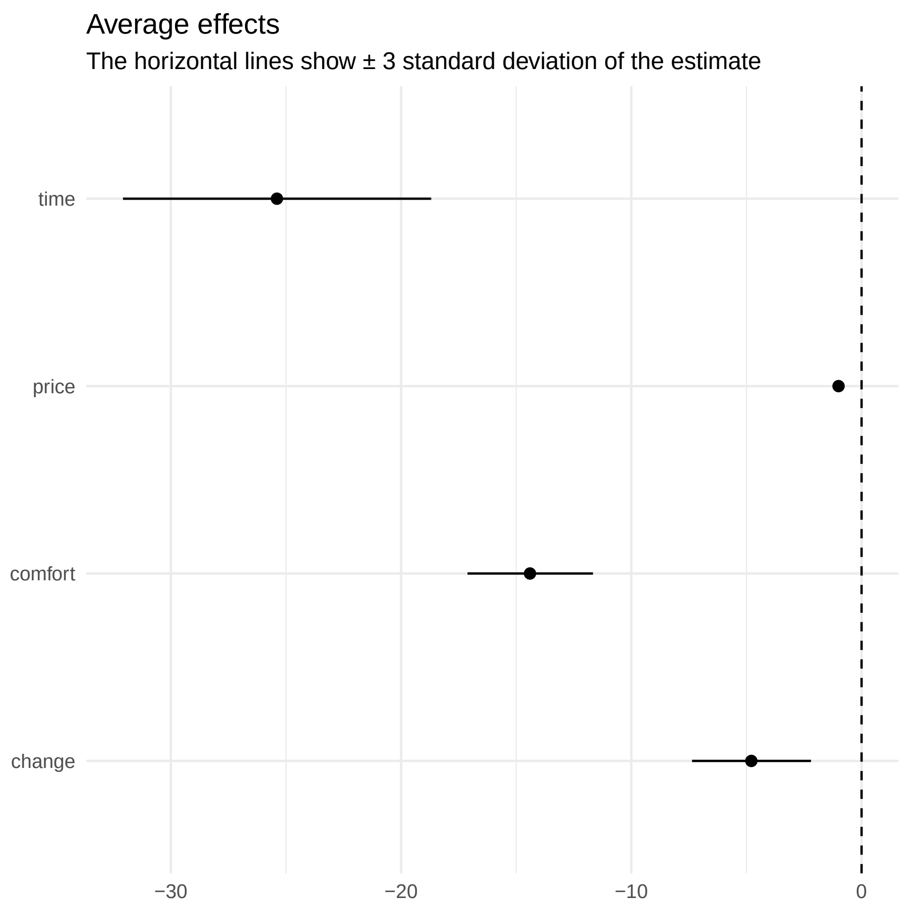
The estimated coefficients (using the mean of the Gibbs samples as a point estimate) can be printed via

```
> coef(model_train)
```

```
          Estimate   (sd)
1   price     -1.00 (0.00)
2    time    -25.39 (2.23)
3  change     -4.79 (0.86)
4 comfort    -14.40 (0.90)
```

and visualized via

```
> plot(coef(model_train), sd = 3)
```



The results indicate that the deciders value one hour travel time by about 25 euros, an additional change by 5 euros, and a more comfortable class by 14 euros. These results are consistent with the ones that are presented in a vignette of the mlogit package on the same data set but using the logit model.

4.5. Checking the Gibbs samples

The Gibbs samples are saved in list form in the '`RprobitB_fit`' object at the entry `"gibbs_samples"`, i.e.

```
> str(model_train$gibbs_samples, max.level = 2, give.attr = FALSE)
```

```
List of 2
 $ gibbs_samples_nbt:List of 2
  ..$ alpha: num [1:500, 1:4] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
  ..$ Sigma: num [1:500, 1] 607 581 627 615 850 ...
 $ gibbs_samples_raw:List of 2
  ..$ alpha: num [1:10000, 1:4] -0.000739 -0.017192 -0.027834 -0.034068 -0.035992 ...
  ..$ Sigma: num [1:10000, 1] 1.022 0.992 0.941 0.921 0.905 ...
```

This object contains 2 elements:

- `gibbs_samples_raw` is a list of the raw samples from the Gibbs sampler,

- and `gibbs_samples_nbt` are the Gibbs samples used for parameter estimates, i.e. the normalized and thinned Gibbs samples after the burn-in.

Calling the summary function on the estimated '`RprobitB_fit`' object yields additional information about the Gibbs samples `gibbs_samples_nbt`. You can specify a list `FUN` of functions that compute any point estimate of the Gibbs samples (Use the function `point_estimates()` to access the Gibbs sample statistics as an '`RprobitB_parameter`' object), for example

- `mean()` for the arithmetic mean,

- `stats::sd()` for the standard deviation,

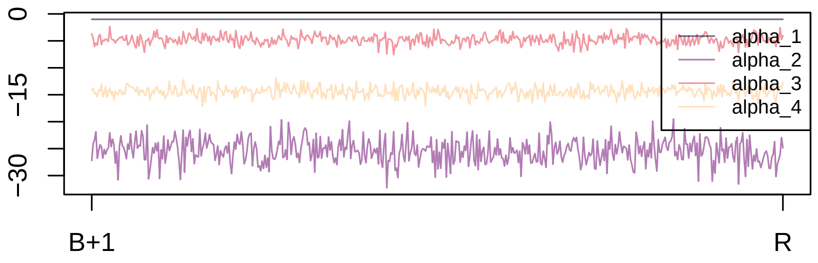
- `R_hat()` for the Gelman-Rubin statistic (Gelman and Rubin 1992) (A Gelman-Rubin statistic close to 1 indicates that the chain of Gibbs samples converged to the stationary distribution),

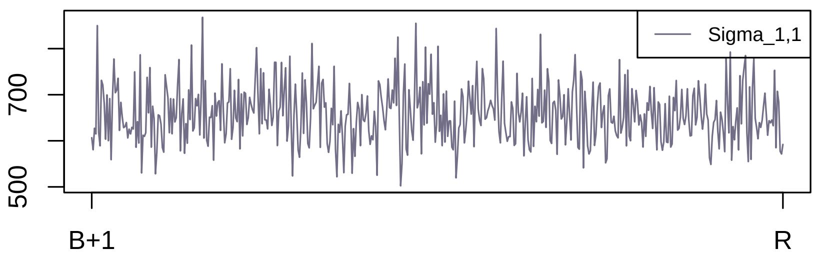
- or custom statistics like the absolute difference between the median and the mean.

```
> summary(model_train,
+         FUN = c("mean"        = mean,
+                 "sd"          = stats::sd,
+                 "R^"          = R_hat,
+                 "custom_stat" = function(x) abs(mean(x) - median(x))
+                 )
+        )


Probit model
choice ~ price + time + change + comfort | 0
R: 10000
B: 5000
Q: 10

Normalization
Level: Utility differences with respect to alternative 2.
Scale: Coefficient of alpha_1 fixed to -1.

Gibbs sample statistics
               mean           sd           R^  custom_stat
 alpha

     1        -1.00         0.00         1.00         0.00
     2       -25.39         2.23         1.01         0.00
     3        -4.79         0.86         1.00         0.02
     4       -14.40         0.90         1.00         0.01

 Sigma

   1,1       658.58        62.47         1.00         6.38
```

Calling the `plot()` method with the additional argument `type = "trace"` plots the trace of the Gibbs samples `gibbs_samples_nbt`:

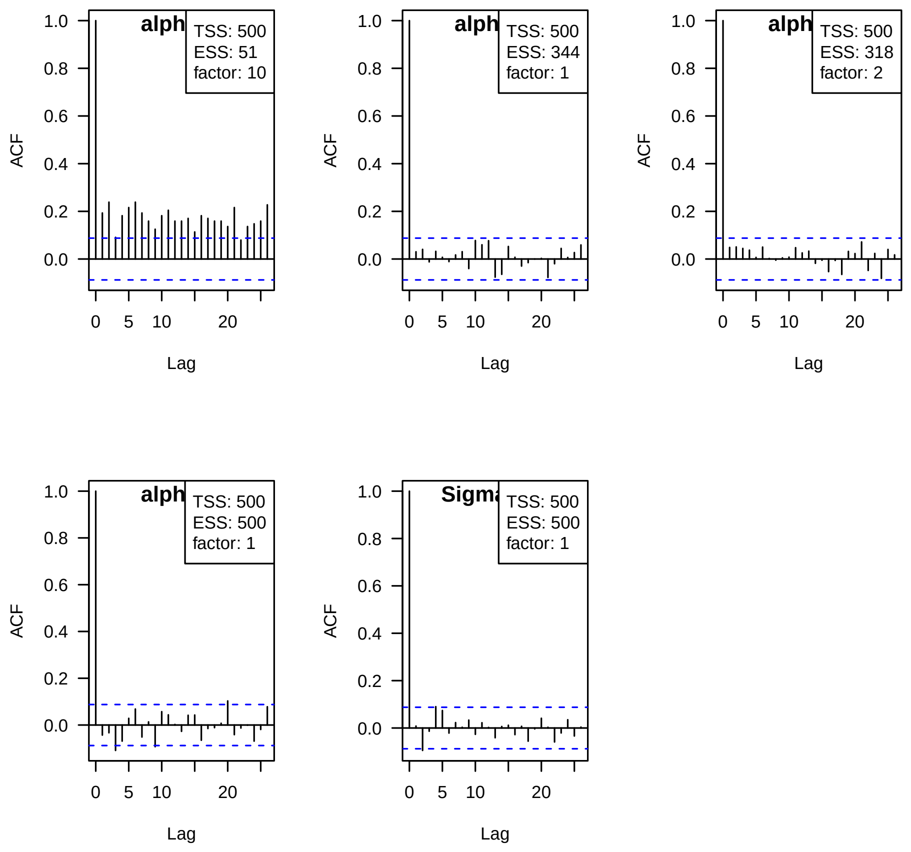
```
> par(mfrow = c(2,1))
> plot(model_train, type = "trace")
```





Additionally, we can visualize the serial correlation of the Gibbs samples via the argument `type = "acf"`. The boxes in the top-right corner state the total sample size TSS (here `R - B` = 10000 - 5000 = 5000), the effective sample size ESS, and the factor by which TSS is larger than ESS.

```
> par(mfrow = c(2,3))
> plot(model_train, type = "acf")
```



Here, the effective sample size is the value $\text{TSS}/(1 + \sum_{k\geq 1} \rho_k)$, where ρ_k is the auto correlation between the chain offset by k positions. The auto correlations are estimated via the `stats::acf()` function.

4.6. Model transformation after estimation

The `transform()` method can be used to transform an '`RprobitB_fit`' object in three ways:

1. change the length `B` of the burn-in period, for example

```
> model_train <- transform(model_train, B = 1)
```

2. change the thinning factor `Q` of the Gibbs samples, for example

```
> model_train <- transform(model_train, Q = 100)
```

3. or change the model normalization `scale`, for example

```
> model_train <- transform(model_train, scale = list(parameter = "s", index = 1, value = 1
```

4.7. Estimating a joint normal mixing distribution

The **mlogit** package (Croissant 2020) contains the data set `Electricity`, in which residential electricity customers were asked to decide between four hypothetical electricity suppliers. The suppliers differed in 6 characteristics:

1. their fixed price `pf` per kWh, 2. their contract length `cf`, 3. a boolean `loc`, indicating whether the supplier is a local company, 4. a boolean `wk`, indicating whether the supplier is a well known company, 5. a boolean `tod`, indicating whether the supplier offers a time-of-day electricity price (which is higher during the day and lower during the night), and 6. a boolean `seas`, indicating whether the supplier's price is seasonal dependent.

This constitutes a choice situation where choice behavior heterogeneity is expected: some customers might prefer a time-of-day electricity price (because they may be not at home during the day), while others can have the opposite preference. Ideally these differences in preferences should be modeled using characteristics of the deciders. In many cases (as in this data set) we do not have adequate information. Instead these differences in taste can be captured by means of a mixing distribution for the `tod` coefficient. This corresponds to the assumption of a random coefficient from the underlying mixing distribution to be drawn for each decider. We can use the estimated mixing distribution to determine for example the share of deciders that have a positive versus negative preference towards time-of-day electricity prices.

Additionally, we expect correlations between the random coefficients to certain covariates, for example a positive correlation between the influence of `loc` and `wk`: deciders that prefer local suppliers might also prefer well known companies due to recommendations and past experiences, although they might be more expensive than unknown suppliers. The fitted multivariate normal distribution will reveal these correlations.

The following lines prepare the `Electricity` data set for estimation. We use the convenience function `as_cov_names()` that relabels the data columns for alternative specific covariates into the required format `"<covariate>_<alternative>"`, compare to the vignette on choice data. Via the `re = c("cl","loc","wk","tod","seas")` argument, we specify that we want to model random effects for all but the price coefficient, which we will fix to `-1` to interpret the other estimates as monetary values.

```
> data("Electricity", package = "mlogit")
> Electricity <- as_cov_names(Electricity, c("pf","cl","loc","wk","tod","seas"), 1:4)
> data <- prepare_data(
+    form = choice ~ pf + cl + loc + wk + tod + seas | 0,
+    choice_data = Electricity,
+    re = c("cl","loc","wk","tod","seas")
+    )
> model_elec <- mcmc(data, R = 1000, scale = list(parameter = "a", index = 1, value = -1))
```

The estimated model is saved in **RprobitB** and can be accessed via:

```
> data(model_elec, package = "RprobitB")
```

Calling the `coef()` method on the estimated model also returns the estimated (marginal) variances of the mixing distribution besides the average mean effects:

```
> coef(model_elec)
```

```
        Estimate   (sd) Variance   (sd)
1   pf     -1.00 (0.00)       NA   (NA)
2   cl     -0.25 (0.03)     0.23 (0.04)
3  loc      2.75 (0.26)     6.79 (1.25)
4   wk      2.00 (0.19)     3.39 (0.67)
5  tod     -9.71 (0.25)    10.53 (2.18)
6 seas     -9.83 (0.22)     5.74 (1.14)
```

By the sign of the estimates we can for example deduce, that the existence of the time-of-day electricity price `tod` in the contract has a negative effect. However, the deciders are very heterogeneous here, because the estimated variance of this coefficient is large (`12.37`). The same holds for the contract length `cl`. In particular, the estimated share of the population that prefers to have a longer contract length equals:

```
> cl_mu <- coef(model_elec)["cl","mean"]
> cl_sd <- sqrt(coef(model_elec)["cl","var"])
> pnorm(cl_mu / cl_sd)
```

```
[1] 0.3026362
```

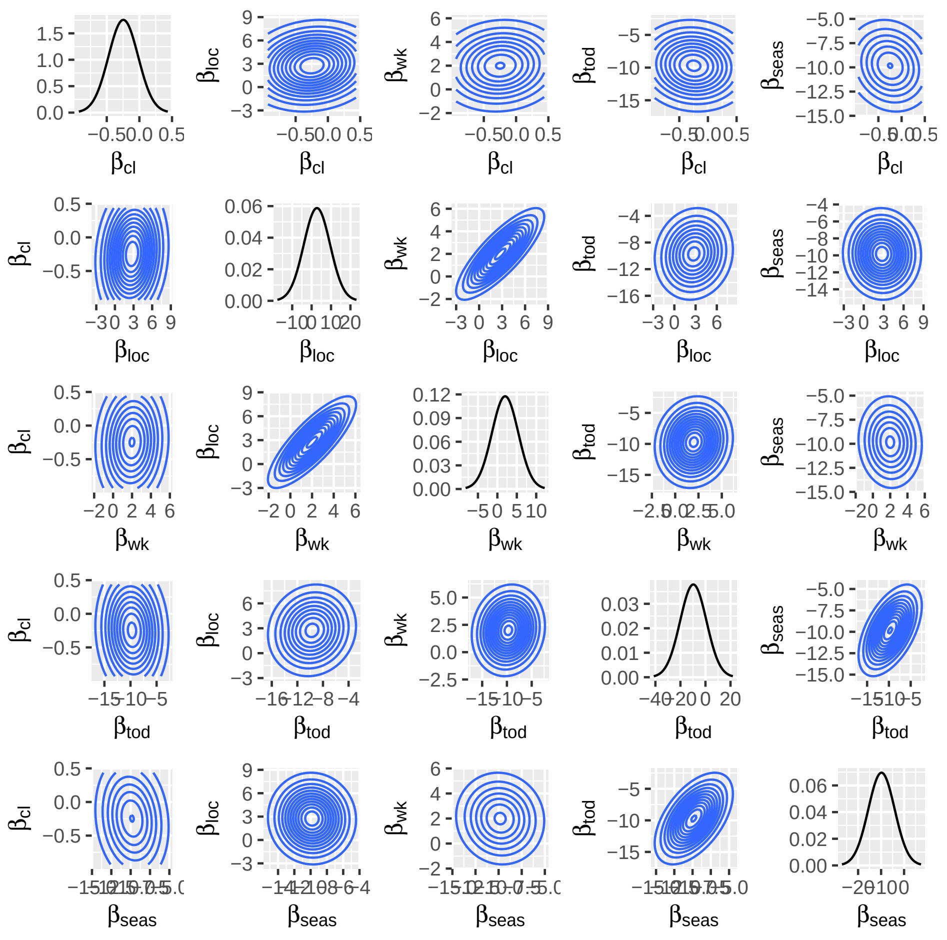
The correlation between the covariates can be accessed as follows (setting `cor = FALSE` instead returns the estimated covariance matrix):

```
> cov_mix(model_elec, cor = TRUE)
```

```
              cl         loc          wk         tod        seas
cl    1.00000000  0.11510129  0.06972935 -0.06581389 -0.13741649
loc   0.11510129  1.00000000  0.81363601  0.09436351 -0.02207951
wk    0.06972935  0.81363601  1.00000000  0.09713654 -0.05208165
tod  -0.06581389  0.09436351  0.09713654  1.00000000  0.53757657
seas -0.13741649 -0.02207951 -0.05208165  0.53757657  1.00000000
```

Here, we see the confirmation of our initial assumption about a high correlation between `loc` and `wk`. The pairwise mixing distributions can be visualized via calling the `plot()` method with the additional argument `type = mixture`:

```
> plot(model_elec, type = "mixture")
```



4.8. Estimating latent classes

More generally, **RprobitB** allows to specify a Gaussian mixture as the mixing distribution. In particular,

$$\beta \sim \sum_{c=1}^{C} \text{MVN}(b_c, \Omega_c).$$

This specification allows for a) a better approximation of the true underlying mixing distribution and b) a preference based classification of the deciders.

To estimate a latent mixture, specify a named list `latent_classes` with the following arguments and submit it to the estimation routine `mcmc()`:

- `C`, the fixed number (greater or equal 1) of latent classes, which is set to 1 per default,

- `weight_update`, a boolean, set to `TRUE` for a weight-based update of the latent classes, see below,

- `dp_update`, a boolean, set to `TRUE` for a Dirichlet process-based update of the latent classes, see below,

- `Cmax`, the maximum number of latent classes, set to `10` per default.

Weight-based update of the latent classes: The following weight-based updating scheme is analogue to Bauer *et al.* (2019) and executed within the burn-in period:

- We remove class c, if $s_c < \varepsilon_{\text{min}}$, i.e. if the class weight s_c drops below some threshold ε_{min}. This case indicates that class c has a negligible impact on the mixing distribution.

- We split class c into two classes c_1 and c_2, if $s_c > \varepsilon_{\text{max}}$. This case indicates that class c has a high influence on the mixing distribution whose approximation can potentially be improved by increasing the resolution in directions of high variance. Therefore, the class means b_{c_1} and b_{c_2} of the new classes c_1 and c_2 are shifted in opposite directions from the class mean b_c of the old class c in the direction of the highest variance.

- We join two classes c_1 and c_2 to one class c, if $\|b_{c_1} - b_{c_2}\| < \varepsilon_{\text{distmin}}$, i.e. if the euclidean distance between the class means b_{c_1} and b_{c_2} drops below some threshold $\varepsilon_{\text{distmin}}$. This case indicates location redundancy which should be repealed. The parameters of c are assigned by adding the values of s from c_1 and c_2 and averaging the values for b and Ω.

These rules contain choices on the values for ε_{min}, ε_{max} and $\varepsilon_{\text{distmin}}$. The adequate value for $\varepsilon_{\text{distmin}}$ depends on the scale of the parameters. Per default, **RprobitB** sets `epsmin = 0.01`, `epsmax = 0.99`, and `distmin = 0.1`. These values can be adapted through the `latent_class` list.

Dirichlet process-based update of the latent classes: As an alternative to the weight-based updating scheme to determine the correct number C of latent classes, **RprobitB** implemented the Dirichlet process. A similar approach in the context of discrete choice can be found in Burda, Harding, and Hausman (2008), where the Dirichlet process is applied to estimate a mixed logit-probit model. The Dirichlet Process is a Bayesian nonparametric method, where nonparametric means that the number of model parameters can theoretically grow to infinity. The method allows to add more mixture components to the mixing distribution if needed for a better approximation, see Neal (2000) for a documentation of the general case. The literature offers many representations of the method, including the Chinese Restaurant Process (Aldous 1985), the stick-braking metaphor (Sethuraman 1994), and the Polya Urn model (Blackwell and MacQueen 1973).

In our case, we face the situation to find a distribution g that explains the decider-specific coefficients $(\beta_n)_{n=1,\dots,N}$, where g is supposed to be a mixture of an unknown number C of Gaussian densities, i.e. $g = \sum_{c=1,\dots,C} s_c \text{MVN}(b_c, \Omega_c)$.

Let $z_n \in \{1, \dots, C\}$ denote the class membership of β_n. A priori, the mixture weights $(s_c)_c$ are given a Dirichlet prior with concentration parameter δ/C, i.e. $(s_c)_c \mid \delta \sim \text{D}_C(\delta/C, \dots, \delta/C)$. Rasmussen (2000) shows that

$$\Pr((z_n)_n \mid \delta) = \frac{\Gamma(\delta)}{\Gamma(N+\delta)} \prod_{c=1}^{C} \frac{\Gamma(m_c + \delta/C)}{\Gamma(\delta/C)},$$

where $\Gamma(\cdot)$ denotes the gamma function and $m_c = \#\{n : z_n = c\}$ the number of elements that are currently allocated to class c. Crucially, the last equation is independent of the class weights $(s_c)_c$, yet it still depends on the finite number C of latent classes. However, Li, Schofield, and Gönen (2019) shows that

$$\Pr(z_n = c \mid z_{-n}, \delta) = \frac{m_{c,-n} + \delta/C}{N - 1 + \delta},$$

where the notation $-n$ means excluding the nth element. We can let C approach infinity to derive:

$$\Pr(z_n = c \mid z_{-n}, \delta) \to \frac{m_{c,-n}}{N - 1 + \delta}.$$

Note that the allocation probabilities do not sum to 1, instead

$$\sum_{c=1}^{C} \frac{m_{c,-n}}{N-1+\delta} = \frac{N-1}{N-1+\delta}.$$

The difference to 1 equals

$$\Pr(z_n \neq z_m \ \forall\ m \neq n \mid z_{-n}, \delta) = \frac{\delta}{N-1+\delta}$$

and constitutes the probability that a new cluster for observation n is created. Neal (2000) points out that this probability is proportional to the prior parameter δ: A greater value for δ encourages the creation of new clusters, a smaller value for δ increases the probability of an allocation to an already existing class.

In summary, the Dirichlet process updates the allocation of each β coefficient vector one at a time, dependent on the other allocations. The number of clusters can theoretically rise to infinity, however, as we delete unoccupied clusters, C is bounded by N. As a final step after the allocation update, we update the class means b_c and covariance matrices Ω_c by means of their posterior predictive distribution. The mean and covariance matrix for a new generated cluster is drawn from the prior predictive distribution. The corresponding formulas are given in Li *et al.* (2019).

The Dirichlet process directly integrates into our existing Gibbs sampler. Given β values, it updated the class means b_c and class covariance matrices Ω_c. The Dirichlet process updating scheme is implemented in the function `update_classes_dp()`. In the following, we give a small example in the bivariate case `P_r = 2`. We sample true class means `b_true` and class covariance matrices `Omega_true` for `C_true = 3` true latent classes. The true (unbalanced) class sizes are given by the vector `N`, and `z_true` denotes the true allocations.

```
> set.seed(1)
> P_r <- 2
> C_true <- 3
> N <- c(100,70,30)
> (b_true <- matrix(replicate(C_true, rnorm(P_r)), nrow = P_r, ncol = C_true))
```

```
           [,1]       [,2]       [,3]
[1,] -0.6264538 -0.8356286  0.3295078
[2,]  0.1836433  1.5952808 -0.8204684
```

```
> (Omega_true <- matrix(replicate(C_true, rwishart(P_r + 1, 0.1*diag(P_r))$W, simplify = T
+                          nrow = P_r*P_r, ncol = C_true))
```

```
          [,1]        [,2]       [,3]
[1,] 0.3093652  0.14358543  0.2734617
[2,] 0.1012729 -0.07444148 -0.1474941
[3,] 0.1012729 -0.07444148 -0.1474941
[4,] 0.2648235  0.05751780  0.2184029
```

```
> beta <- c()
> for(c in 1:C_true) for(n in 1:N[c])
+    beta <- cbind(beta, rmvnorm(mu = b_true[,c,drop=F], Sigma = matrix(Omega_true[,c,drop
> z_true <- rep(1:3, times = N)
```

We specify the following prior parameters (for their definition see the vignette on model fitting):

```
> delta <- 0.1
> xi <- numeric(P_r)
> D <- diag(P_r)
> nu <- P_r + 2
> Theta <- diag(P_r)
```

Initially, we start with `C = 1` latent classes. The class mean `b` is set to zero, the covariance matrix `Omega` to the identity matrix:

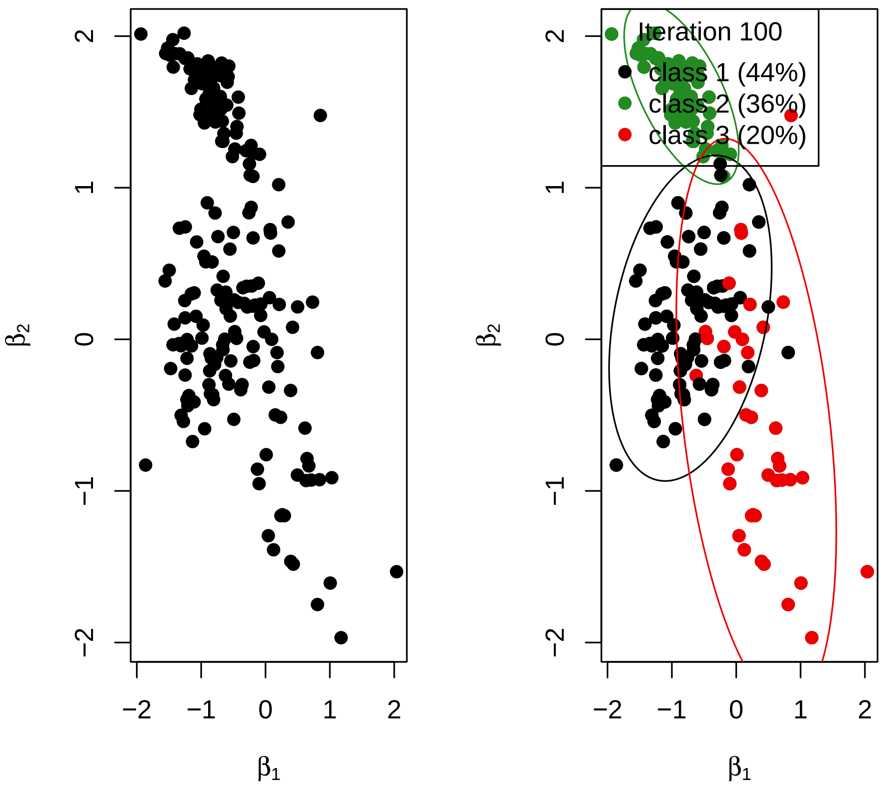
```
> z <- rep(1, ncol(beta))
> C <- 1
> b <- matrix(0, nrow = P_r, ncol = C)
> Omega <- matrix(rep(diag(P_r), C), nrow = P_r*P_r, ncol = C)
```

The following call to `update_classes_dp()` updates the latent classes in `100` iterations. Note that we specify the arguments `Cmax` and `s_desc`. The former denotes the maximum number of latent classes. This specification is not a requirement for the Dirichlet process per se, but rather for its implementation. Knowing the maximum possible class number, we can allocate the required memory space, which leads to a speed improvement. We later can verify that we won't exceed the number of `Cmax = 10` latent classes at any point of the Dirichlet process. Setting `s_desc = TRUE` ensures that the classes are ordered by their weights in a descending order to ensure identifiability.

```
> for(r in 1:100){
+    dp <- RprobitB::update_classes_dp(
+      Cmax = 10, beta, z, b, Omega, delta, xi, D, nu, Theta, s_desc = TRUE
+      )
+    z <- dp$z
+    b <- dp$b
+    Omega <- dp$Omega
+  }
```

The Dirichlet process was able to infer the true number `C_true = 3` of latent classes:

```
> par(mfrow = c(1,2))
> plot(t(beta), xlab = bquote(beta[1]), ylab = bquote(beta[2]), pch = 19)
> RprobitB:::plot_class_allocation(beta, z, b, Omega, r = 100, perc = 0.95)
```



5. Choice prediction

This section discusses in-sample and out-of-sample prediction within **RprobitB**. The former case refers to reproducing the observed choices on the basis of the covariates and the fitted model and subsequently using the deviations between prediction and reality as an indicator for the model performance. The latter means forecasting choice behavior for changes in the choice attributes. For illustration, we revisit the probit model of travelers deciding between two fictional train route alternatives:

```
> data("model_train", package = "RprobitB")
```

5.1. Reproducing the observed choices

RprobitB provides a `predict()` method for '`RprobitB_fit`' objects. Per default, the method returns a confusion matrix, which gives an overview of the in-sample prediction performance:

```
> predict(model_train)

    predicted
true    A    B
   A 1022  452
   B  440 1015
```

By setting the argument `overview = FALSE`, the method instead returns predictions on the level of individual choice occasions:

```
> pred <- predict(model_train, overview = FALSE)
> head(pred, n = 10)

   id choiceid         A          B true predicted correct
1   1        1 0.9152710 0.08472896    A         A    TRUE
2   1        2 0.6395050 0.36049499    A         A    TRUE
3   1        3 0.7918220 0.20817801    A         A    TRUE
4   1        4 0.1792245 0.82077546    B         B    TRUE
5   1        5 0.5500492 0.44995085    B         A   FALSE
6   1        6 0.1299623 0.87003770    B         B    TRUE
7   1        7 0.5436992 0.45630079    B         A   FALSE
8   1        8 0.7589727 0.24102733    B         A   FALSE
9   1        9 0.5483893 0.45161074    A         A    TRUE
10  1       10 0.5931064 0.40689358    A         A    TRUE
```

Among the three incorrect predictions shown here, the one for decider `id = 1` in choice occasion `idc = 8` is the most outstanding. Alternative `B` was chosen although the model predicts a probability of 75% for alternative `A`. We can use the convenience function `get_cov()` to extract the characteristics of this particular choice situation:

```
> get_cov(model_train, id = 1, idc = 8)

  id choiceid choice  price_A   time_A change_A comfort_A  price_B
8  1        8      B 52.88904 1.916667        0         1 70.51872
  time_B change_B comfort_B
8    2.5        0         0
```

The trip option `A` was 20 euros cheaper and 30 minutes faster, which by our model outweighs the better comfort class for alternative `B`, recall the estimated effects:

```
> coef(model_train)

          Estimate   (sd)
1   price    -1.00 (0.00)
2    time   -25.39 (2.23)
3  change    -4.79 (0.86)
4 comfort   -14.40 (0.90)
```

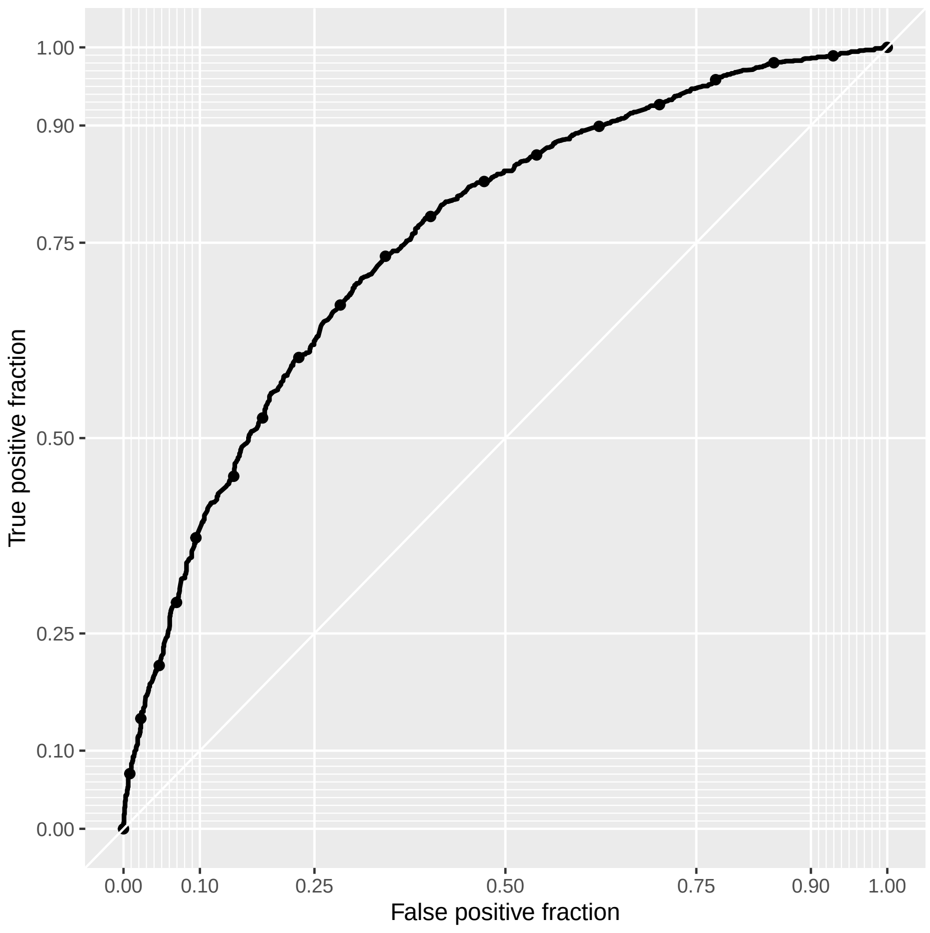
The misclassification can be explained by preferences that differ from the average decider (choice behavior heterogeneity), or by unobserved factors that influenced the choice. Indeed, the variance of the error term was estimated high:

```
> point_estimates(model_train)$Sigma
```

```
         [,1]
[1,] 658.5798
attr(,"names")
[1] "1,1"
```

Apart from the prediction accuracy, the model performance can be evaluated more nuanced in terms of sensitivity and specificity. The following snippet exemplary shows how to visualize these measures by means of a receiver operating characteristic (ROC) curve (Fawcett 2006), using the **plotROC** package (Sachs 2017). The curve is constructed by plotting the true positive fraction against the false positive fraction at various cutoffs (here `n.cuts = 20`). The closer the curve is to the top-left corner, the better the binary classification.

```
> library(plotROC)
> ggplot(data = pred, aes(m = A, d = ifelse(true == "A", 1, 0))) +
+    geom_roc(n.cuts = 20, labels = FALSE) +
+    style_roc(theme = theme_grey)
```



5.2. Forecasting choice behavior

The `predict()` method has an additional `data` argument. Per default, `data = NULL`, which results into the in-sample case outlined above. Alternatively, `data` can be either an '`RprobitB_data`' object (for example the test subsample derived from the `train_test()` function, or a data frame of custom choice characteristics.

We demonstrate the second case in the following. Assume that a train company wants to anticipate the effect of a price increase on their market share. By our model, increasing the ticket price from 100 euros to 110 euros (ceteris paribus) draws 15% of the customers to the competitor who does not increase their prices.

```
> predict(
+    model_train,
+    data = data.frame("price_A" = c(100,110),
+                      "price_B" = c(100,100)),
+    overview = FALSE)

  id choiceid         A         B prediction
1  1        1 0.5000000 0.5000000          A
2  2        1 0.3483907 0.6516093          B
```

However, offering a better comfort class compensates for the higher price and even results in a gain of 7% market share:

```
> predict(
+   model_train,
+   data = data.frame("price_A"   = c(100,110),
+                     "comfort_A" = c(1,0),
+                     "price_B"   = c(100,100),
+                     "comfort_B" = c(1,1)),
+   overview = FALSE)

  id choiceid         A         B prediction
1  1        1 0.5000000 0.5000000          A
2  2        1 0.5680923 0.4319077          A
```

6. Model selection

The task of model selection targets the question: If there are several competing models, how do I choose the most appropriate one? This section outlines the model selection tools implemented in **RprobitB**. For illustration, we revisit the probit model of travelers deciding between two fictional train route alternatives:

```
> data("model_train", package = "RprobitB")
> model_train

Probit model 'choice ~ price + time + change + comfort | 0'.
```

As a competing model, we consider explaining the choices only by the alternative's price, i.e. the probit model with the formula `choice ~ price | 0‘`. (This model is also saved in **RprobitB** and can be accessed via `data("model_train_sparse", package = "RprobitB")`.)

```
> model_train_sparse <- nested_model(model_train, form = choice ~ price | 0)
```

The `nested_model()` function helps to estimate a new version of `model_train` with new specifications. Here, only `form` has been changed.

The model selection function: **RprobitB** provides the convenience function `model_selection()`, which takes an arbitrary number of ‘`RprobitB_fit`’ objects and returns a matrix of model selection criteria:

```
> model_selection(model_train, model_train_sparse,
+                 criteria = c("npar", "LL", "AIC", "BIC", "WAIC", "MMLL", "BF", "pred_ac
                         model_train model_train_sparse
npar                               4                  1
LL                          -1727.74           -1865.86
AIC                          3463.48            3733.72
BIC                          3487.41            3739.70
WAIC                         3463.76            3733.91
se(WAIC)                        0.18               0.07
pWAIC                           4.32               1.15
MMLL                        -1732.14           -1867.48
BF(*,model_train)                  1             < 0.01
BF(*,model_train_sparse)       > 100                  1
pred_acc                      69.55%             63.37%
```

Specifying `criteria` is optional. Per default, `criteria = c("npar", "LL", "AIC", "BIC")`. The available model selection criteria are explained in the following.

npar: `"npar"` yields the number of model parameters, which is computed by the `npar()` method:

```
> npar(model_train, model_train_sparse)

[1] 4 1
```

Here, `model_train` has 4 parameters (a coefficient for price, time, change, and comfort, respectively), while `model_train_sparse` has only a single price coefficient.

LL: If `"LL"` is included in `criteria`, `model_selection()` returns the model's log-likelihood values. They can also be directly accessed via the `logLik()` method. The log-likelihood values are per default computed at the point estimates derived from the Gibbs sample means. `logLik()` has the argument `par_set`, where alternative statistics for the point estimate can be specified.

```
> logLik(model_train)

[1] -1727.742

> logLik(model_train_sparse)

[1] -1865.861
```

AIC: Including `"AIC"` yields the Akaike's Information Criterion (Akaike 1974), which is computed as

$$-2 \cdot \mathrm{LL} + k \cdot \mathrm{npar},$$

where LL is the model's log-likelihood value, k is the penalty per parameter with $k = 2$ per default for the classical AIC, and npar is the number of parameters in the fitted model.

Alternatively, the `AIC()` method also returns the AIC values:

```
> AIC(model_train, model_train_sparse, k = 2)

[1] 3463.485 3733.723
```

The AIC quantifies the trade-off between over- and under-fitting, where smaller values are preferred. Here, the increase in goodness of fit justifies the additional 3 parameters of `model_train`.

BIC: Similar to the AIC, `"BIC"` yields the Bayesian Information Criterion (Schwarz 1978), which is defined as

$$-2 \cdot \text{LL} + \log(\text{nobs}) \cdot \text{npar},$$

where LL is the model's log-likelihood value, nobs is the number of data points (which can be accessed via the `nobs()` method), and npar is the number of parameters in the fitted model. The interpretation is analogue to AIC.

RprobitB also provided a method for the BIC value:

```
> BIC(model_train, model_train_sparse)

[1] 3487.414 3739.705
```

WAIC: WAIC is short for Widely Applicable (or Watanabe-Akaike) Information Criterion (Watanabe and Opper 2010). As for AIC and BIC, the smaller the WAIC value the better the model. Including `"WAIC"` in `criteria` yields the WAIC value, its standard error `se(WAIC)`, and the effective number of parameters `pWAIC`, see below.

The WAIC is defined as

$$-2 \cdot \text{lppd} + 2 \cdot p_{\text{WAIC}},$$

where lppd stands for log pointwise predictive density and p_{WAIC} is a penalty term proportional to the variance in the posterior distribution that is sometimes called effective number of parameters, see McElreath (2020) p. 220 for a reference.

The lppd is approximated as follows. Let

$$p_{si} = \Pr(y_i \mid \theta_s)$$

be the probability of observation y_i (here the single choices) given the s-th set θ_s of parameter samples from the posterior. Then

$$\text{lppd} = \sum_i \log \left(S^{-1} \sum_s p_{si} \right).$$

The penalty term is computed as the sum over the variances in log-probability for each observation:

$$p_{\text{WAIC}} = \sum_i \mathbb{V}_\theta \log(p_{si}).$$

The WAIC has a standard error of

$$\sqrt{n \cdot \mathbb{V}_i \left[-2\left(\text{lppd} - \mathbb{V}_\theta \log(p_{si})\right)\right]},$$

where n is the number of choices.

Before computing the WAIC of an `RprobitB_fit` object, the probabilities p_{si} must be computed via the `compute_p_si()` function:

```
> model_train <- compute_p_si(model_train)
```

Afterwards, the WAIC can be accessed as follows, where the number in brackets is the standard error:

```
> WAIC(model_train)
```

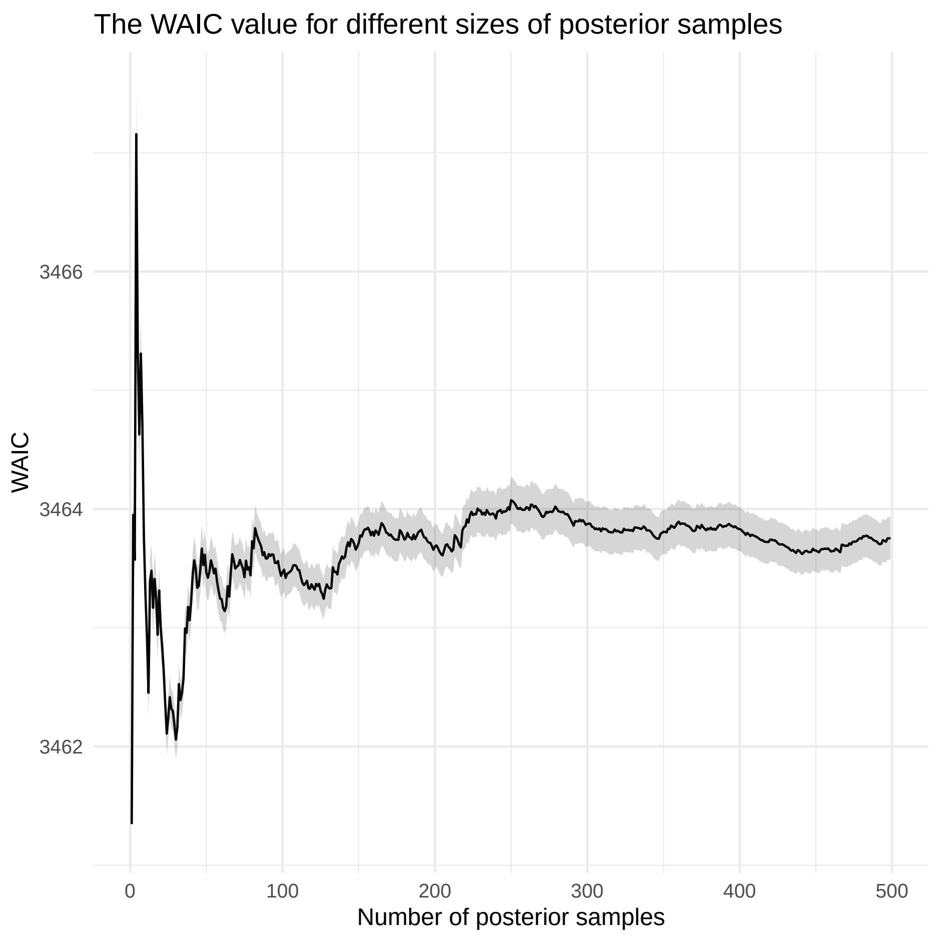
```
3463.76 (0.18)
```

```
> WAIC(model_train_sparse)
```

```
3733.91 (0.07)
```

You can visualize the convergence of the WAIC as follows:

```
> plot(WAIC(model_train))
> plot(WAIC(model_train_sparse))
```



Here, both approximations look satisfactory. If the WAIC value does not seem to have converged, use more Gibbs samples by increasing `R` in `mcmc()` or decreasing `B` or `Q` via `transform()`, see the vignette on model fitting.

MMLL: `"MMLL"` in `criteria` stands for marginal model log-likelihood. The model's marginal likelihood $\Pr(y \mid M)$ for a model M and data y is required for the computation of Bayes factors, see below. In general, the term has no closed form and must be approximated numerically.

RprobitB uses the posterior Gibbs samples derived from the `mcmc()` function to approximate the model's marginal likelihood via the posterior harmonic mean estimator (Newton and Raftery 1994): Let S denote the number of available posterior samples $\theta_1, \dots, \theta_S$. Then,

$$\Pr(y \mid M) = (\mathbb{E}_{\text{posterior}} 1/\Pr(y \mid \theta, M))^{-1} \approx \left(\frac{1}{S} \sum_s 1/\Pr(y \mid \theta_s, M) \right)^{-1} = \tilde{\Pr}(y \mid M).$$

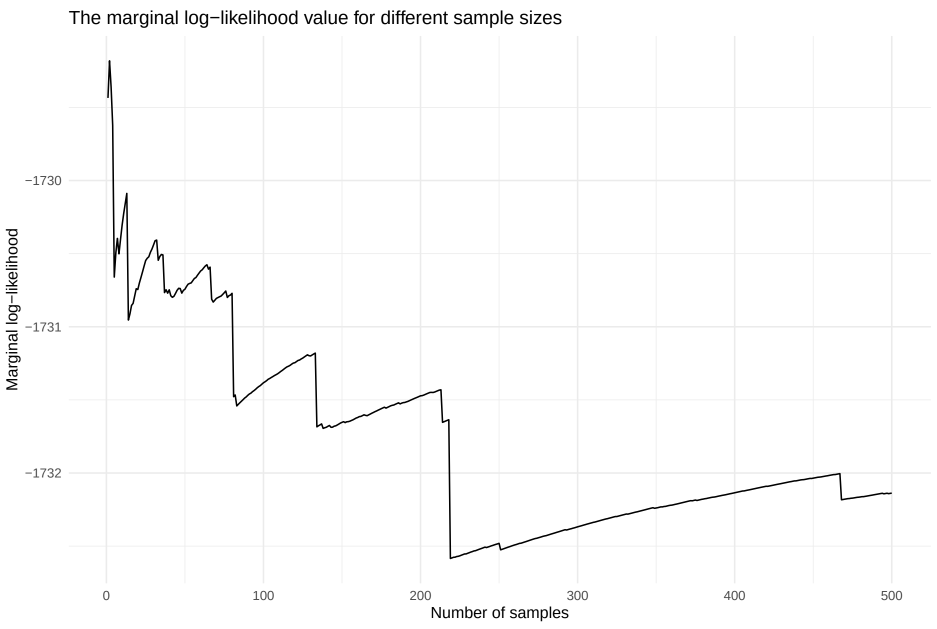
By the law of large numbers, $\tilde{\Pr}(y \mid M) \to \Pr(y \mid M)$ almost surely as $S \to \infty$.

As for the WAIC, computing the MMLL relies on the probabilities $p_{si} = \Pr(y_i \mid \theta_s)$, which must first be computed via the `compute_p_si()` function. Afterwards, the `mml()` function can be called with an '`RprobitB_fit`' object as input. The function returns the '`RprobitB_fit`' object, where the marginal likelihood value is saved as the entry "mml" and the marginal log-likelihood value as the attribute "mmll". Note that the marginal likelihood value is very small. The given representation is required so that the value is not rounded to 0 by the computer.

```
> model_train <- mml(model_train)
> model_train$mml

3.54e-117 * exp(-1464)

> attr(model_train$mml, "mmll")

[1] -1732.138
```

Analogue to the WAIC value, the computation of the MMLL is an approximation that improves with rising (posterior) sample size. The convergence can again be verified visually via the `plot()` method:

```
> plot(model_train$mml, log = TRUE)
```



There are two options for improving the approximation: As for the WAIC, you can use more posterior samples. Alternatively, you can combine the posterior harmonic mean estimate with the prior arithmetic mean estimator (Hammersley and Handscomb 1964): For this approach, S samples $\theta_1, \dots, \theta_S$ are drawn from the model's prior distribution. Then,

$$\Pr(y \mid M) = \mathbb{E}_{\text{prior}} \Pr(y \mid \theta, M) \approx \frac{1}{S} \sum_s \Pr(y \mid \theta_s, M) = \tilde{\Pr}(y \mid M).$$

Again, it hols by the law of large numbers, that $\tilde{\Pr}(y \mid M) \to \Pr(y \mid M)$ almost surely as $S \to \infty$. The final approximation of the model's marginal likelihood than is a weighted sum of the posterior harmonic mean estimate and the prior arithmetic mean estimate, where the weights are determined by the sample sizes.

To use the prior arithmetic mean estimator, call the `mml()` function with a specification of the number of prior draws `S` and set `recompute = TRUE`:

```
> model_train <- mml(model_train, S = 1000, recompute = TRUE)
```

Note that the prior arithmetic mean estimator works well if the prior and posterior distribution have a similar shape and strong overlap, as Gronau, Sarafoglou, Matzke, Ly, Boehm, Marsman, Leslie, Forster, Wagenmakers, and Steingroever (2017) points out. Otherwise, most of the sampled prior values result in a likelihood value close to zero, thereby contributing only marginally to the approximation. In this case, a very large number `S` of prior samples is required.

Bayes factor: The Bayes factor is an index of relative posterior model plausibility of one model over another (Marin and Robert 2014). Given data `y` and two models `mod0` and `mod1`, it is defined as

$$BF(\texttt{mod0},\texttt{mod1}) = \frac{\Pr(\texttt{mod0} \mid \texttt{y})}{\Pr(\texttt{mod1} \mid \texttt{y})} = \frac{\Pr(\texttt{y} \mid \texttt{mod0})}{\Pr(\texttt{y} \mid \texttt{mod1})} / \frac{\Pr(\texttt{mod0})}{\Pr(\texttt{mod1})}.$$

The ratio $\Pr(\texttt{mod0}) / \Pr(\texttt{mod1})$ expresses the factor by which `mod0` a priori is assumed to be the correct model. Per default, **RprobitB** sets $\Pr(\texttt{mod0}) = \Pr(\texttt{mod1}) = 0.5$. The front part $\Pr(\texttt{y} \mid \texttt{mod0}) / \Pr(\texttt{y} \mid \texttt{mod1})$ is the ratio of the marginal model likelihoods. A value of $BF(\texttt{mod0},\texttt{mod1}) > 1$ means that the model `mod0` is more strongly supported by the data under consideration than `mod1`.

Adding `"BF"` to the `criteria` argument of `model_selection()` yields the Bayes factors. We see a decisive evidence (Jeffreys 1998) in favor of '`model_train`'.

```
> model_selection(model_train, model_train_sparse, criteria = c("BF"))
```

```
                           model_train model_train_sparse
BF(*,model_train)                    1             < 0.01
BF(*,model_train_sparse)         > 100                  1
```

pred acc: Finally, adding `"pred_acc"` to the `criteria` argument for the `model_selection()` function returns the share of correctly predicted choices. From the output of `model_selection()` above (or alternatively the one in the following) we deduce that '`model_train`' correctly predicts about 6% of the choices more than '`model_train_sparse`':

```
> pred_acc(model_train, model_train_sparse)
```

```
[1] 0.6954592 0.6336634
```

7. Applications

7.1. Chess opening choice

7.2. Berserking choice in online chess

7.3. Child wish

7.4. Contraception choice

8. Conclusion

This paper addressed the problem of specifying mixing distributions in the multinomial probit model with a panel data setting, constituting an important part of the model selection for which the literature does not provide much guidance so far. In the absence of alternatives, many applications of the mixed multinomial probit model rely on different types of standard parametric distributions for modelling heterogeneity in preferences in combination with a likelihood-value based model selection. This course of action is very restrictive and imposes strong assumptions on the distribution of the model parameters that could potentially lead to misspecification and biased estimates.

We proposed a new approach that improves the current specification strategies in several ways: First, our approach does not require any distributional assumption, since the latent class setting is flexible enough to approximate practically any distribution shape and allowing for any correlation pattern. Second, the weight-based updating scheme ensures that the number of latent classes does not need to be prespecified. Third, the imposed Bayesian estimation framework avoids many numerical problems that occur in the frequentist approach. Most notably, no likelihood function has to be evaluated nor approximated. Comparing the numerical estimation speed to the non-parametric frequentist approach of Bauer *et al.* (2019), we found that our implementation of the Bayesian approach is at least 10 times faster. The improvement becomes more distinct for panel data settings with a high number of choice occasions. This is due to the fact that for given total sample size NT a large T is beneficial for the Bayesian approach as then the number of vectors $\beta_n,\ n = 1, ..., N$ is comparatively small, while in the frequentist approach calculating the likelihood becomes more challenging for increasing the number T of choice situations faced by each of the N individuals. On the other hand, the grid based frequentist approach of Bauer *et al.* (2019) can potentially achieve a better approximation (especially of point masses) due to the relatively high number of latent classes. However, this approach requires that a suitable grid is set prior to the estimation with a specification of upper bounds for the coefficients. Additionally, the curse of dimensionality plays a crucial role, which is less of a burden in the Bayesian approach. Note that for a fully specified parametric structure these concerns do not play such a big role also for the frequentist approach.

Our simulation results verified that the proposed approach achieves good approximations of the mixing distribution in common choice modelling situations, where the underlying heterogeneity cannot be captured by standard parametric distributions. It would be interesting to apply the approach also to empirical data in the future. Additionally, further research on how to properly address sign-restricted coefficients is required.

Computational details

The results in this paper were obtained using R 4.1.0 with the **RprobitB** 1.0.0.9000 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`.

Acknowledgments

This work has been financed partly by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Projektnummer 356500581 which is gratefully acknowledged.

References

Agresti A (2015). *Foundations of Linear and Generalized Linear Models.* Wiley. ISBN 978-1-118-73003-4.

Akaike H (1974). "A New Look at the Statistical Model Identification." **19**.

Albert JH, Chib S (1993). "Bayesian Analysis of Binary and Polychotomous Response Data." *Journal of the American Statistical Association*, **88**.

Aldous DJ (1985). "Exchangeability and related topics." **1117**, 1–198.

Allenby GM, Rossi P (1998). "Marketing models of consumer heterogeneity." *Journal of Econometrics*, **89**.

Bauer D, Büscher S, Batram M (2019). "Non-parameteric estiation of mixed discrete choice models." *Second International Choice Modelling Conference in Kobe.*

Bhat C (2011). "The Maximum Approximate Composite Marginal Likelihood (MACML) Estimation of Multinomial Probit-Based Unordered Response Choice Models." *Transportation Research Part B: Methodological*, **45**.

Bhat C, Lavieri P (2018). "A new mixed MNP model accommodating a variety of dependent non-normal coefficient distributions." *Theory and Decision*, **84**.

Blackwell D, MacQueen J (1973). "Ferguson distributions via Polya urn schemes." *The Annals of Statistics*, **1**, 353–355.

Burda M, Harding M, Hausman J (2008). "A Bayesian mixed logit–probit model for multinomial choice." *Journal of Econometrics*, **147**(2), 232–246. `doi:10.1016/j.jeconom.2008.09.029`. URL `https://www.sciencedirect.com/science/article/pii/S0304407608001395`.

Cirillo C, Axhausen K (2006). "Evidence on the distribution of values of travel time savings from a six-week diary." *Transportation Research Part A: Policy and Practice*, **40**.

Croissant Y (2020). "Estimation of Random Utility Models in R: The mlogit Package." *Journal of Statistical Software*, **95**(11), 1–41. doi:10.18637/jss.v095.i11.

Fawcett T (2006). "An introduction to ROC analysis." *Pattern Recognition Letters*, **27**(8), 861–874. doi:10.1016/j.patrec.2005.10.010. URL https://www.sciencedirect.com/science/article/pii/S016786550500303X.

Fountas G, Anastasopoulos Ph, Abdel-Aty M (2018). "Analysis of accident injury-severities using a correlated random parameters ordered probit approach with time variant covariates." *Analytic Methods in Accident Research*, **18**.

Fountas G, Pantangi SS, Hulme KF, Anastasopoulos PC (2019). "The effects of driver fatigue, gender, and distracted driving on perceived and observed aggressive driving behavior: A correlated grouped random parameters bivariate probit approach." *Analytic Methods in Accident Research*, **22**.

Gelman A, Rubin DB (1992). "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science*, **7**(4), 457 – 472. doi:10.1214/ss/1177011136.

Geweke J (1998). "Efficient Simulation from the Multivariate Normal and Student-t Distributions Subject to Linear Constraints and the Evaluation of Constraint Probabilities."

Gronau QF, Sarafoglou A, Matzke D, Ly A, Boehm U, Marsman M, Leslie DS, Forster JJ, Wagenmakers E, Steingroever H (2017). "A tutorial on bridge sampling." *Journal of mathematical psychology*, **81**, 80–97.

Hammersley JM, Handscomb DC (1964). "General principles of the monte carlo method." pp. 50–75.

Imai K, van Dyk DA (2005). "A Bayesian analysis of the multinomial probit model using marginal data augmentation." *Journal of Econometrics*, **124**.

Jeffreys H (1998). *The theory of probability*. OUP Oxford.

Li Y, Schofield E, Gönen M (2019). "A tutorial on Dirichlet process mixture modeling." *Journal of Mathematical Psychology*, **91**, 128–144. ISSN 0022-2496. doi:10.1016/j.jmp.2019.04.004. URL https://www.sciencedirect.com/science/article/pii/S0022249618301068.

Marin J, Robert C (2014). *Bayesian essentials with R*. Springer Textbooks in Statistics. Springer Verlag, New York.

McCulloch R, Rossi P (1994). "An exact likelihood analysis of the multinomial probit model." *Journal of Econometrics*, **64**.

McElreath R (2020). *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. 2. ed. edition. Chapman and Hall/CRC.

Neal RM (2000). "Markov Chain Sampling Methods for Dirichlet Process Mixture Models." *Journal of Computational and Graphical Statistics*, **9**(2), 249–265. ISSN 10618600. URL http://www.jstor.org/stable/1390653.

Newton MA, Raftery AE (1994). "Approximate Bayesian inference with the weighted likelihood bootstrap." *Journal of the Royal Statistical Society: Series B (Methodological)*, **56**(1), 3–26.

Nobile A (1998). "A hybrid Markov chain for the Bayesian analysis of the multinomial probit model." *Statistics and Computing*, **8**.

Oelschläger L, Bauer D (2020). "Bayes Estimation of Latent Class Mixed Multinomial Probit Models." *TRB Annual Meeting 2021*.

Rasmussen CE (2000). *The Infinite Gaussian Mixture Model*, volume 12. MIT Press.

Sachs MC (2017). "plotROC: A Tool for Plotting ROC Curves." *Journal of Statistical Software, Code Snippets*, **79**(2), 1–19. doi:10.18637/jss.v079.c02.

Scaccia L, Marcucci E (2010). "Bayesian Flexible Modelling of Mixed Logit Models."

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**.

Sethuraman J (1994). "A constructive definition of dirichlet priors." *Statistica Sinica*, **4**(2), 639–650. URL http://www.jstor.org/stable/24305538.

Train K (2009). *Discrete choice methods with simulation*. 2. ed. edition. Cambridge Univ. Press.

Train K (2016). "Mixed logit with a flexible mixing distribution." *Journal of choice modelling*, **19**.

Watanabe S, Opper M (2010). "Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory." *Journal of machine learning research*, **11**(12).

Xiong Y, Mannering FL (2013). "The heterogeneous effects of guardian supervision on adolescent driver-injury severities: A finite-mixture random-parameters approach." *Transportation Research Part B: Methodological*, **49**.

Affiliation:

Lennart Oelschläger
Department of Business Administration and Economics
Bielefeld University
Postfach 10 01 31
E-mail: `lennart.oelschlaeger@uni-bielefeld.de`

Journal of Statistical Software `http://www.jstatsoft.org/`
published by the Foundation for Open Access Statistics `http://www.foastat.org/`
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
`doi:10.18637/jss.v000.i00` *Accepted:* yyyy-mm-dd