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

Many applied research areas seek to understand decision maker's choices among a discrete set of alternatives, for example between different brands (marketing) or options to commute (transportation). Of central interest is heterogeneity in choice behavior: do deciders weight choice attributes like product price or travel time differently? If yes, to what extend? And what groups of deciders share similar preferences? Answering questions of this type is the motivation behind the presented statistical software **RprobitB** (Oelschläger and Bauer 2021). The package name is a portmanteau of R (the programming language), the probit model class, and the Bayesian estimation framework.

The probit model is one of the most widely-used statistical models to explain discrete choices. Heterogeneity in choice behavior can be modeled using mixing distributions for the coefficients. Recently, Oelschläger and Bauer (2020) 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.

	Probit	Logit	Bayes	ML	Ord.	Mix.	LC	upd. LC
Rchoice	✓	✓		✓	✓	✓		
mlogit	✓	✓		✓	✓	✓		
Biogeme	✓	✓		✓	✓	✓	✓	
apollo	✓	✓	✓	✓	✓	✓	✓	
bayesm	✓	✓	✓		✓	✓		
MNP	✓		✓		✓			
RprobitB	✓		✓			✓	✓	✓

Table 1: Overview of packages for discrete choice modeling.

Example	Illustrated package functionalities
1: Train	`prepare_data()`, `fit_model()`, `predict()`, `model_selection()`
2: Simulated choices	`simulate_choices()`, estimation and weight-based update of latent classes
3: Electricity	estimation and interpretation of random effects
4: Online chess strategy	Dirichlet process-based update of latent classes, preference classification

Table 2: Overview of examples.

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

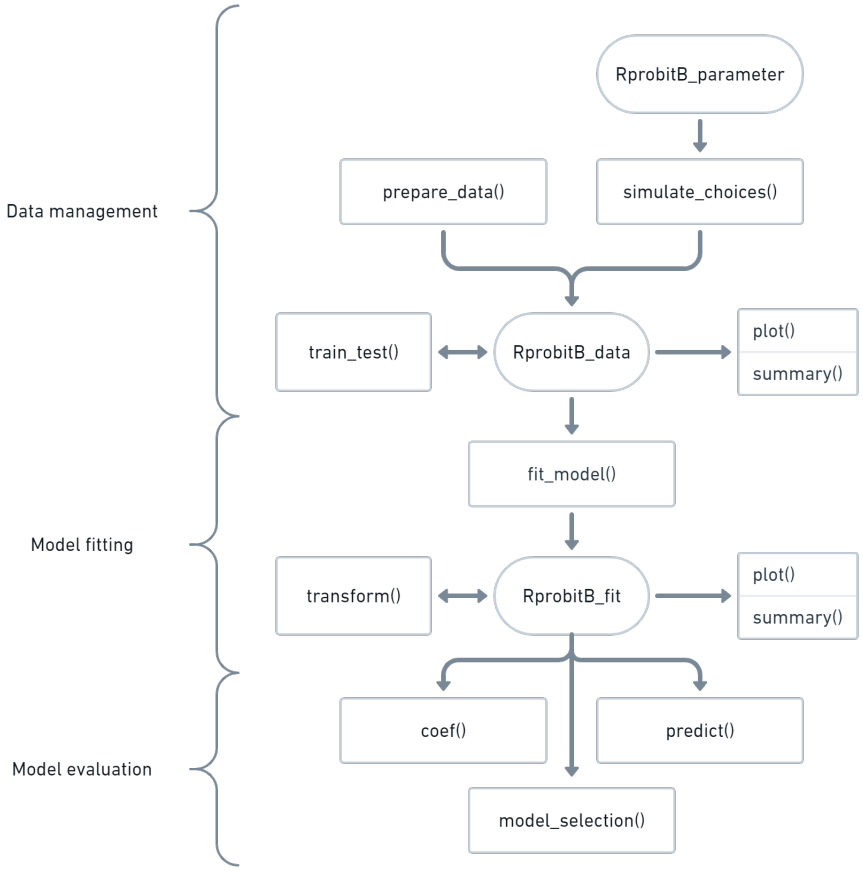


Figure 1: Package flowchart with main functions as rectangles and objects as ovals.

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 takes a finite number of values and the error term is normally distributed (Agresti 2015). Its most popular

application are discrete choice scenarios. The dependent variable in this case is one of finitely many and mutually exclusive alternatives, and explanatory variables typically are characteristics of the deciders or the alternatives. This section defines the probit model based on the concept of latent utilities, outlines a model extension for incorporating choice behavior heterogeneity, and discusses required normalization for parameter identification.

2.1. Latent utilities

Assume that we know the choices of N deciders choosing 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 covariates. We seek to explain the choices by a linear combination of those. However, the linear combination is continuous and cannot be linked directly to the discrete choices. Hence, we extend the model by latent variables U_{ntj} for each decider n, choice occasion t, and alternative j. In the discrete choice setting, these variables can be interpreted as the decider's utility of a certain alternative at a certain choice occasion. Formally:

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

for $n = 1, \ldots, N$, $t = 1, \ldots, T$ and $j = 1, \ldots, J$. 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 $(\epsilon_{nt:}) = (\epsilon_{nt1}, \ldots, \epsilon_{ntJ})' \sim \text{MVN}_J(0, \Sigma)$ 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]

Assuming utility maximizing behavior of the decision makers[3], we link the latent utilities to the choices via

$$y_{nt} = \underset{j=1,\ldots,J}{\text{argmax}}\, U_{ntj}.$$

Here, $y_{nt} = j$ denotes the event that decider n chooses j at occasion t.

2.2. Choice behavior heterogeneity

Note that the coefficient vector β in equation (1) is constant across decision makers (a so-called fixed effect). This assumption is too restrictive for many applications[4] and can be relaxed by imposing a distribution on β, a so-called mixing distribution (Train 2009). Now, each decider n can have their own sensitivities β_n as a realization from the underlying mixing distribution. To allow for a combination of such random effects next to fixed effects, we

[1] For notational simplicity, the number of choice occasions T is assumed to be the same for each decision maker here. However, **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 latter, each ϵ_{ntj} is assumed to be independently extreme value distributed.

[3] We note that utility maximizing behavior is a common assumption in econometric models. However, many studies have shown that humans do not decide in this rational sense in general, see for example Hewig, Kretschmer, Trippe, Hecht, Coles, Holroyd, and Miltner (2011).

[4] As an example, consider the choice of a means of transportation: It is easily imaginable that business people and pensioners do not share the same sensitivities towards cost and time.

replace β in in equation (1) by $\beta = (\alpha, \beta_n)$, where α are P_f coefficients that are constant across deciders and β_n are P_r decider specific coefficients, $P_f + P_r = P$. Now if $P_r > 0$, β_n is distributed according to some P_r-variate 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 (Train 2009). 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:

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

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.

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

This interpretation enables a classification of the deciders in terms of their preferences, see Section 4.6 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. For identification of the model parameters, we therefore normalize the model by considering only utility differences and fixing one error term variance. Formally, equation (1) is transformed to

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

$n = 1, \dots, N$, $t = 1, \dots, T$, and $j = 1, \dots, J-1$. Here (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 one diagonal element fixed to a positive number.[5]

3. Choice data

[5] Restricting an element of $\tilde{\Sigma}$ determines the utility scale. Fixing one fixed effect α serves the same purpose. Both alternatives are implemented in **RprobitB**, see Section 4.2.

Choice data typically consist of the observed choices in conjunction with choice characteristics.[6] **RprobitB** requests that data sets are (a) of class '`data.frame`' and (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[7], (d) contain a column with the observed choices (required for model fitting but not for prediction), and (e) contain columns for the values of alternative and/or decider specific covariates. 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).

This section introduces the formula framework of **RprobitB** for specifying the set of covariates entering a model. The framework is adapted from **mlogit**, which is flexible enough to allow for different types of covariates: covariates that are constant across alternatives (e.g. the decider's age), covariates that are alternative specific (e.g. the alternative's price), covariates with a generic coefficient (e.g. paying the same amount of money for train company A versus B should make no difference), and covariates that have alternative specific coefficients (e.g. spending time in a crowded train versus a private jet makes a difference). Subsequently, we demonstrate how to pass such a formula to the functions `prepare_data()` for preparing empirical data for estimation and `simulate_choices()` for simulating choice data.

3.1. Formula framework

We generalize equation (1) to allow for different types of covariates:

$$U_{ntj} = A_{ntj}\beta_1 + B_{nt}\beta_{2j} + C_{ntj}\beta_{3j} + \epsilon_{ntj}. \tag{2}$$

Here, the covariates A and C depend on the alternative, whereas B is only choice occasion specific. The coefficient β_1 is generic (i.e. the same for each alternative), whereas β_{2j} and β_{3j} are alternative specific. Note that the full collection $(\beta_{2j})_{j=1,\dots,J}$ is not identified. This is because we took utility differences for model normalization, see Section 2.3. We therefore fix β_{2k} to 0 for one alternative k (the so-called base alternative). The coefficients $(\beta_{2j})_{j\neq k}$ have to be interpreted with respect to alternative k.

Equation (2) can be entered into R via specifying the '`formula`' object `choice ~ A | B | C`, where `choice` is the name of the dependent variable (the discrete choice we aim to explain). By default, alternative specific constants (ASCs)[8] 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 want to estimate ASCs, add `1` in the second spot, e.g. `choice ~ A`

[6] We only consider unordered alternatives (that is, alternatives cannot be ranked). Every decider may take one or repeated choices (called choice occasions).

[7] Additionally, it can contain a column with identifier for each choice occasion of each decider.

[8] ASCs capture the average effect on utility of all factors that are not included in the model. We cannot estimate ASCs for all the alternatives due to identifiability. Therefore, they are added for all except for the base alternative.

`| 1 | C`. The expression `choice ~ A | 0 | C` is interpreted as no covariates of the second category and no alternative specific constants.

3.2. Preparing data for estimation

Before model estimation, any choice data set `choice_data` must pass the `prepare_data()` function together with a formula object `form` introduced above:

```
> 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 `fit_model()` (introduced in Section 4). The following arguments of `prepare_data()` are optional:

- `re`: A character vector of covariate names in `form` with random effects (see Section 2.2). Per default `re = NULL`, i.e. no random effects. To have random effects for the ASCs, include `"ASC"` in `re`.
- `alternatives`: A character vector of alternative names, defining the choice set. If not specified, all alternatives chosen in the data set are considered.
- `base_alternative`: One element of `alternatives` specifying the base alternative (see Section 3.1). Per default, `base_alternative` is the last element of `alternatives`.
- `id` and `idc`: The names of the columns in `choice_data` that contain unique identifier for each decision maker and for each choice occasion, respectively. Per default, `id = "id"` and `idc = NULL`, in which case the choice occasion 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 (i.e. rescaled to have a mean of 0 and a standard deviation of 1). Per default, no covariate is standardized.
- `impute`: A character, specifying how to handle missing data entries. Options are `"complete_cases"` (removing rows that contain missing entries, which is the default behavior), `"zero_out"` (replacing missing entries by 0), and `"mean"` (imputing missing entries by the covariate mean).

Example 1: Train. The **mlogit** package contains the choice data set `Train`, which we use to demonstrate the `prepare_data()` function. The example is continued in Section 4.2 for a demonstration of the estimation routine. The data set contains 2929 stated choices of 235 deciders between two fictional train trip alternatives `A` and `B`. The trip alternatives are characterized by their `price`, the travel `time`, the level of `comfort` (the lower the value the higher the comfort), and the number of `change`s. The data is in wide format; the columns `id` and `choiceid` identify the deciders and the choice occasions, respectively; the column `choice` provides the choices. For convenience, we transform `time` from minutes to hours and `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
> 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  52.9 52.9 52.9 88.1 52.9 ...
 $ time_A   : num  2.5 2.5 1.92 2.17 2.5 ...
 $ 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  88.1 70.5 88.1 70.5 70.5 ...
 $ time_B   : num  2.5 2.17 1.92 2.5 2.5 ...
 $ 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 ...
```

The naming convention `<covariate>_<alternative>` in the `Train` data set for alternative specific covariates is the required format for **RprobitB**. Say we want to include all of them in our probit model, connect them to generic coefficients, and exclude ASCs. Then we would specify the formula

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

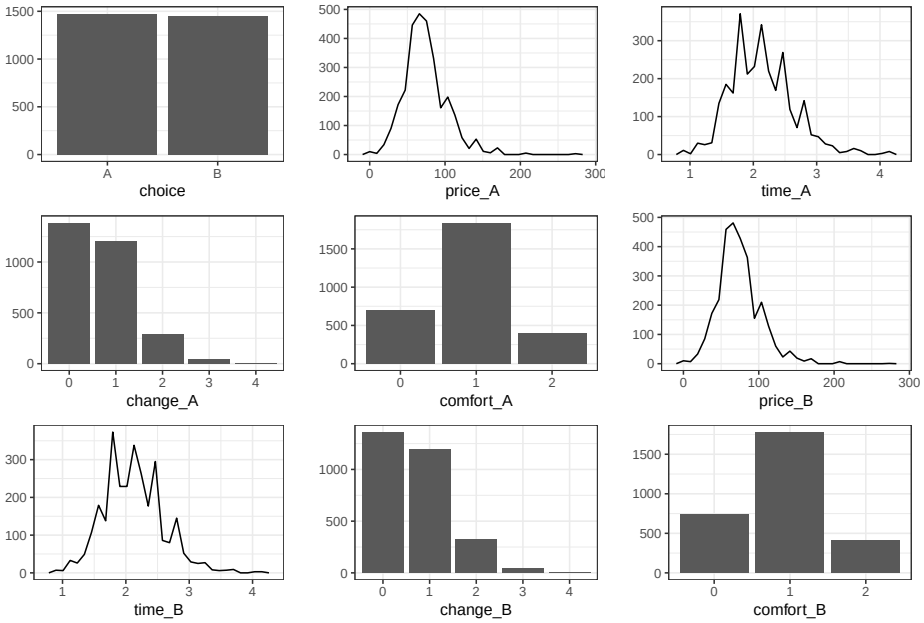
Passing `form` to `prepare_data()` returns an '`RprobitB_data`' object. The object can be inspected via its `summary()` and `plot()` methods:

```
> data_train <- prepare_data(
+    form = form,
+    choice_data = Train,
+    id = "id",
+    idc = "choiceid"
+    )
> summary(data_train)

  number deciders choice occasions choices total
1             235     5 to 19 each          2929

  alternative frequency
1           A      1474
2           B      1455

   effect as_value as_coef random
```

```
1   price     TRUE   FALSE  FALSE
2    time     TRUE   FALSE  FALSE
3 comfort     TRUE   FALSE  FALSE
4  change     TRUE   FALSE  FALSE
```

Above, `as_value`, `as_coef`, and `random` are booleans for each effect, indicating whether the covariate is alternative specific, the coefficient is alternative specific, and the effect is random, respectively.

```
> plot(data_train)
```



3.3. Simulating choice data

The `simulate_choices()` function simulates discrete choice data from a probit model. Say we want to simulate the choices of `N` deciders in `T` choice occasions[9] among `J` alternatives. Together with a model formula `form`, we would have to call

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

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

- `re`: Analogue to `prepare_data()`.
- `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.

[9] `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

- `base_alternative`: Analogue to `prepare_data()`.
- `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`: Analogue to `prepare_data()`.
- `seed`: Optionally set a seed for the simulation.

True model parameters are set at random per default or can be specified via the function's ellipsis argument:

- a numeric vector `alpha` with the fixed effects,
- an integer `C`, the number (greater or equal 1) of latent classes of decision makers (`C = 1` per default),
- a numeric vector `s` of length `C` with the class weights,
- a matrix `b` with the class means as columns,
- a matrix `Omega` with the class covariance matrices as columns,
- a matrix `Sigma`, the differenced error term covariance matrix, or `Sigma_full`, the full error term covariance matrix,
- a matrix `beta` with the decision-maker specific coefficient vectors as columns,
- a numeric vector `z` of length `N` with elements in `1:C`, representing the class allocations.

Example 2: Simulated choices. For illustration, we simulate the choices of `N = 100` deciders at `T = 30` choice occasions between the fictitious alternatives `alt1` and `alt2`. The choices are explained by the alternative specific covariates `var1` and `var3`. Covariate `var2` is only choice occasion specific but connected to a random effect, as well as the ASCs:

```
> N <- 100
> T <- 30
> alternatives <- c("alt1", "alt2")
> base_alternative <- "alt2"
> 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,
+    base_alternative = base_alternative
+  )
```

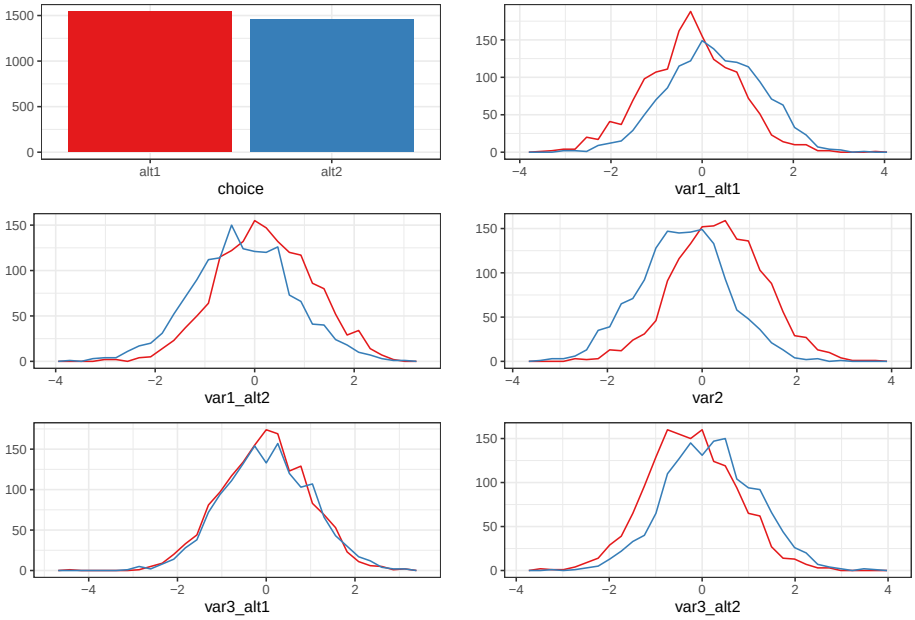
```
     effect as_value as_coef random
1      var1     TRUE   FALSE  FALSE
2 var3_alt1     TRUE    TRUE  FALSE
3 var3_alt2     TRUE    TRUE  FALSE
4 var2_alt1    FALSE    TRUE   TRUE
5  ASC_alt1    FALSE    TRUE   TRUE
```

The model has three fixed effects (`random = FALSE`), hence the vector `alpha` must be of length 3, where the elements 1 to 3 correspond to `var1`, `var3_alt1`, and `var3_alt2`, respectively. Additionally, the model has two fixed effects (`random = TRUE`), hence the matrix `b` must be of dimension `2 x C`, where row 1 and 2 correspond to `var2_alt1` and `ASC_alt1`, respectively. We specify `C = 2` latent classes in the data generating process with the intention, that this example will serve as an illustration of the estimation of latent classes in Section 4.5.

```
> data_sim <- simulate_choices(
+    form = form,
+    N = N,
+    T = T,
+    J = 2,
+    re = re,
+    alternatives = alternatives,
+    base_alternative = base_alternative,
+    seed = 1,
+    alpha = c(-1,0,1),
+    C = 2,
+    s = c(0.7,0.3),
+    b = matrix(c(2,-0.5,1,1), ncol = 2),
+    Sigma = 1
+  )
```

The `plot()` method of '`RprobitB_data`' object has the optional argument `by_choice`. Setting `by_choice = TRUE` visualizes the (randomly drawn) covariates grouped by the chosen alternatives:

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



The graphic is consistent with our model specification: for example, covariate `var1` was specified to have a negative effect on `alt1`, because the coefficient of `var1` (the first value of `alpha`) is negative (−1). Hence higher values of `var1_alt1` correspond more frequently to choice `alt2` (upper-right panel).

3.4. Train and test data set

RprobitB provides the function `train_test()` that can split an '`RprobitB_data`' object (i.e. the output of `prepare_data()` or `simulate_choices()`) in two parts: a train subset and a test subset. The model then is typically estimated on the train subset and applied to explain the choices in the test subset. This procedure can be useful for evaluating how the model generalizes to unseen data. For example, the following call puts 70% of deciders from `data_sim` into the train subsample and 30% of deciders into the test subsample:

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

$train
Simulated data of 2100 choices.

$test
Simulated data of 900 choices.
```

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

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

$train
Simulated data of 2800 choices.
```

```
$test
Simulated data of 200 choices.
```

4. Model fitting

RprobitB estimates the probit model in a Bayesian framework. The framework 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 (Albert and Chib 1993), which treats the latent utilities in model equation (1) as additional parameters. Then, conditional on these parameters, 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).

This section lists the prior distributions for the model parameters and formulates the conditional posterior distributions. We subsequently introduce the estimation routine `fit_model()` and apply it to the two examples from the previous section. The remainder of this section presents how **RprobitB** estimates latent classes and updates their number either weight-based or via a Dirichlet process.

4.1. Prior and posterior distributions

This section follows closely Oelschläger and Bauer (2020). We a priori 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,
- $\Omega_c \sim W_{P_r}^{-1}(\nu, \Theta)$, independent for all c, where $W_{P_r}^{-1}(\nu, \Theta)$ denotes the P_r-dimensional inverse Wishart distribution with ν degrees of freedom and scale matrix Θ,
- and $\Sigma \sim W_{J-1}^{-1}(\kappa, \Lambda)$.

These priors imply the following conditional posterior distributions:

- The class weights are drawn from the Dirichlet distribution

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

  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

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

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

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

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

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

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

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):

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

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

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

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

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

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

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

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

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

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

Samples obtained from the updating scheme described above lack identification (except for s and z draws), compare Section 2.3. Therefore, subsequent to the sampling, we apply the following normalization for the i-th 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 j-th 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 i-th 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. The estimation routine

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

```
> fit_model(data = data)
```

where `data` is an '`RprobitB_data`' object (see Section 3). The function has the following optional arguments:

- `scale`: A formula object, which determines the utility scale (cf. Section 2.3). It is of the form `<parameter> ~ <value>`, where `<parameter>` is either the name of a fixed effect or `Sigma_<j>` for the `<j>`-th diagonal element of `Sigma`, and `<value>` is the value of the fixed parameter (i.e. const introduced above). Per default `scale = Sigma_1 ~ 1`, i.e. the first error-term variance is fixed to 1.
- `R`: The number of iterations of the Gibbs sampler. The default is `R = 10000`.
- `B`: The length of the burn-in period[10], i.e. a non-negative number of samples to be discarded. The default is `B = R/2`.

[10] 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).

- `Q`: The thinning factor for the Gibbs samples[11], 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, see `help(check_prior, package = "RprobitB")`):
  - `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 Section 4.3.

Example 1: Train (cont.). Recall the `Train` data set of stated train trip alternatives, characterized by their `price`, `time`, number of `change`s, and level of `comfort`. From this data, we previously build the '`RprobitB_data`' object `data_train`, which we now pass to the estimation routine `fit_model()`. For model normalization, we fix the coefficient of the `price` effect to `-1`, which has the advantage that we can interpret the other coefficients as monetary values:

```
> model_train <- fit_model(data = data_train, scale = price ~ -1)
```

The 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

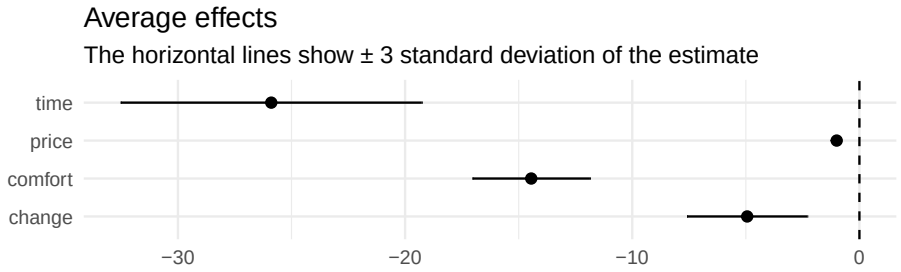
```
> coef(model_train)
```

[11] In order to obtain independent Gibbs samples, we consider only every `Q`-th sample when computing parameter statistics.

```
           Estimate   (sd)
1   price     -1.00 (0.00)
2    time    -25.89 (2.21)
3  change     -4.94 (0.88)
4 comfort    -14.45 (0.86)
```

and visualized via

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



The results indicate that the deciders value one hour travel time by about 26 euros, an additional change by 5 euros, and a more comfortable class by 14 euros.[12] 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_raw:List of 2
  ..$ alpha: num [1:10000, 1:4] -0.00239 -0.02179 -0.02986 -0.03343 -0.03584 ...
  ..$ Sigma: num [1:10000, 1] 0.978 0.949 0.862 0.864 0.911 ...
 $ 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] 649 541 662 691 623 ...
```

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 transformed Gibbs samples used for parameter estimates (i.e. the normalized and thinned Gibbs samples after the burn-in).

Calling the `summary()` method on the estimated '`RprobitB_fit`' object yields additional information about the (transformed) Gibbs samples. The method receives a list `FUN` of functions that compute any point estimate of the Gibbs samples, for example

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

[12] We note that 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.

- `R_hat()` for the Gelman-Rubin statistic (Gelman and Rubin 1992)[13],

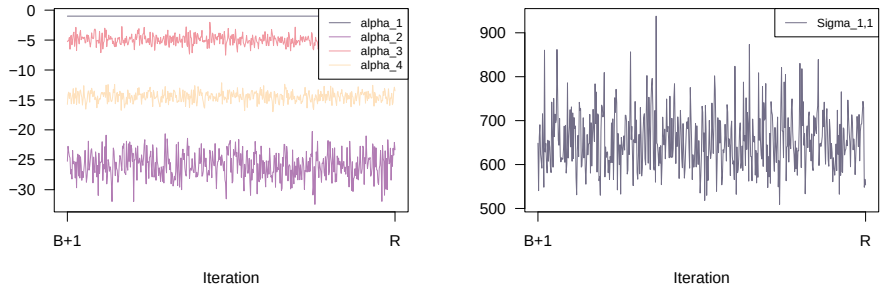
- or custom statistics like the absolute difference between the median and the mean as an indicator for the symmetry of the posterior distribution.

```
> summary(model_train,
+          FUN = c("mean" = mean, "sd" = stats::sd, "R^" = RprobitB::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

Utility normalization
Level: Utility differences with respect to alternative 'B'.
Scale: Coefficient of effect 'price' (alpha_1) fixed to -1.

Gibbs sample statistics
               mean           sd           R^  custom_stat
 alpha

     1        -1.00         0.00         1.00         0.00
     2       -25.89         2.21         1.00         0.02
     3        -4.94         0.88         1.01         0.01
     4       -14.45         0.86         1.00         0.02

 Sigma

   1,1       655.56        65.79         1.00         7.28
```

Calling the `plot()` method with the additional argument `type = "trace"` plots the trace of the transformed and thinned Gibbs samples after the burn-in:

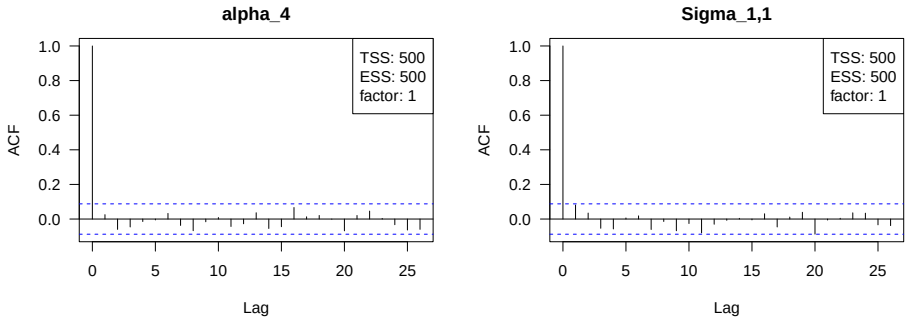
```
> par(mfrow = c(1,2))
> plot(model_train, type = "trace")
```

[13] A Gelman-Rubin statistic close to 1 indicates that the chain of Gibbs samples converged to the stationary distribution.



Additionally, we can visualize the serial correlation of the Gibbs samples via the argument `type = "acf"` (here exemplary for `alpha_4` and `Sigma_1,1`):

```
> par(mfrow = c(1,2))
> plot(model_train, type = "acf", ignore = c("alpha_1", "alpha_2", "alpha_3"))
```



The boxes in the top-right corner state the total sample size TSS, given by (`R - B`) / `Q`, the effective sample size ESS, and the factor by which TSS is larger than ESS. The effective sample size is the value $\text{TSS}/(1 + 2\sum_{k\geq 1}\rho_k)$, where ρ_k is the k-th order auto correlation of the Gibbs samples (Marin and Robert 2014). The auto correlations are estimated via the `stats::acf()` function.

In our example, the presented diagnosing tools (Gelman-Rubin statistic, trace plot, autocorrelation plot) indicate a satisfying quality of the Gibbs samples. If that would not the case, the `transform()` method can be used to increase the length of the burn-in (via `transform(model_train, B = B_new)`) or to increase the thinning factor (via `transform(model_train, Q = Q_new)`).[14]

4.3. Estimating a joint normal mixing distribution

This section demonstrates how to estimate a joint normal mixing distribution in **RprobitB** using another real-data example.

[14] The `transform()` function can also be used to change the model normalization `scale`, for example `transform(model_train, scale = Sigma_1  1)`.

Example 3: Electricity. The **mlogit** package (Croissant 2020) contains the data set `Electricity`, in which residential electricity customers were asked to decide between four contract offers of hypothetical electricity suppliers. Heterogeneity in choice behavior is expected here, because customers might value contract characteristics differently based on their living conditions. In particular, the contracts differed in 6 characteristics:

1. their fixed price `pf` per kilowatt hour,
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.

The following lines prepare the `Electricity` data set for estimation. We first use the convenience function `as_cov_names()` that relabels the data columns for alternative specific covariates into the required format `"<covariate>_<alternative>"`:

```
> data("Electricity", package = "mlogit")
> Electricity <- as_cov_names(
+    choice_data = Electricity,
+    cov = c("pf","cl","loc","wk","tod","seas"),
+    alternatives = 1:4
+  )
```

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_elec <- prepare_data(
+    form = choice ~ pf + cl + loc + wk + tod + seas | 0,
+    choice_data = Electricity,
+    re = c("cl","loc","wk","tod","seas")
+  )
> model_elec <- fit_model(data_elec, R = 5000, scale = pf ~ -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 returns a table of the average effects and the estimated (marginal) variances of the mixing distribution:

```
> 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      2.77 (0.24)     6.74 (1.19)
4   wk      2.02 (0.19)     3.48 (0.69)
5  tod     -9.70 (0.23)    10.88 (1.80)
6 seas     -9.87 (0.19)     5.90 (1.06)
```

We can for example deduce, that a longer contract length has a negative effect on average (`-0.25`). However, our model shows that 30% of the customers still prefer to have a longer contract length. This share is estimated by computing the proportion under the mixing distribution that yields a positive coefficient for `cl`:

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

```
[1] 0.2996708
```

Furthermore, the estimated joint mixing distribution allows to infer correlations between effects. They can be extracted via the `cov_mix()` function (setting `cor = FALSE` would return the covariances). For example, we see a high correlation between `loc` and `wk` (deciders that prefer local suppliers also prefer well known companies):

```
> round(cov_mix(model_elec, cor = TRUE), 2)
```

```
        cl  loc    wk   tod  seas
cl    1.00 0.12  0.09 -0.06 -0.14
loc   0.12 1.00  0.81  0.12  0.02
wk    0.09 0.81  1.00  0.11 -0.02
tod  -0.06 0.12  0.11  1.00  0.54
seas -0.14 0.02 -0.02  0.54  1.00
```

4.4. Estimating latent classes

RprobitB allows to specify a Gaussian mixture $\sum_{c=1}^{C} \text{MVN}(b_c, \Omega_c)$ as the mixing distribution (cf. Section 2.2). 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 such a latent mixture, pass the list `latent_classes = list("C" = C)` to `fit_model()`, with `C` being the number (greater or equal 1) of latent classes (set to 1 per default).[15]

[15] We here assume that C is known and fixed. The following Sections 4.5 and 4.6 present two updating schemes in which C does not need to be pre-specified.

Example 2: Simulation (cont.). We previously simulated the 'RprobitB_data' object data_sim from a probit model with two latent classes. We now aim to reproduce the model parameters from the data generating process:

```
> model_sim <- fit_model(
+    data = data_sim,
+    R = 1000,
+    latent_classes = list("C" = 2),
+    seed = 1
+  )
> summary(model_sim)

Probit model
choice ~ var1 | var2 | var3
R: 1000
B: 500
Q: 1

Utility normalization
Level: Utility differences with respect to alternative 'alt2'.
Scale: Coefficient of the 1. error term variance fixed to 1.

Latent classes
C = 2

Gibbs sample statistics
          true    mean      sd      R^
 alpha

     1   -1.00   -0.99    0.09    1.20
     2    0.00   -0.03    0.04    1.03
     3    1.00    0.93    0.09    1.07

 s

     1    0.70    0.70    0.09    1.01
     2    0.30    0.30    0.09    1.01

 b

   1.1    2.00    2.04    0.21    1.06
   1.2   -0.50   -0.51    0.28    1.00
   2.1    1.00    0.74    0.41    1.05
   2.2    1.00    1.20    0.32    1.00

 Omega
```

```
 1,1,1    0.31    0.23    0.14    1.71
 1,1,2    0.71    0.37    0.25    1.52
 1,2,2    4.67    4.33    1.16    1.04
 2,1,1    1.67    1.18    0.51    1.11
 2,1,2   -1.20   -0.71    0.35    1.03
 2,2,2    0.87    0.66    0.31    1.01

 Sigma

   1,1    1.00    1.00    0.00    1.00
```

4.5. Weight-based update of the latent classes

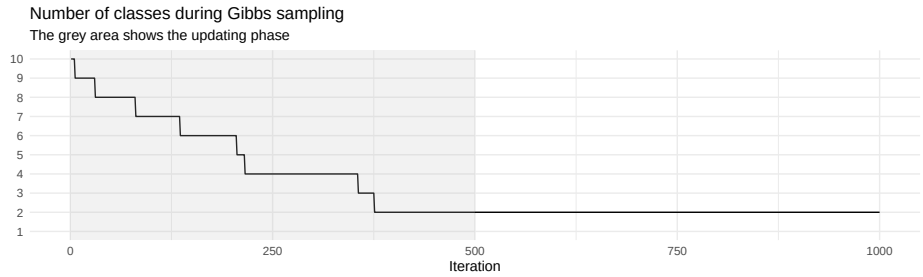
Adding `"wp_update" = TRUE` to the list `latent_classes` above executes the following weight-based updating scheme (analogue to Bauer *et al.* (2019)):

- Class c is removed, 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.
- Class c is splitted 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.
- Classes c_1 and c_2 are joined 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.

The values for ε_{min}, ε_{max} and $\varepsilon_{\text{distmin}}$ can be specified. Per default, **RprobitB** sets `epsmin = 0.01`, `epsmax = 0.99`, and `distmin = 0.1`.

```
> model_sim <- fit_model(
+    data = data_sim,
+    R = 1000,
+    latent_classes = list(
+      "C" = 10,
+      "weight_update" = TRUE,
+      "buffer" = 5
+      ),
+    seed = 1
+  )
```

```
> plot(model_sim, type = "class_seq")
```



4.6. Dirichlet process-based update of the latent classes

As an alternative to the weight-based updating scheme, **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 n-th 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:

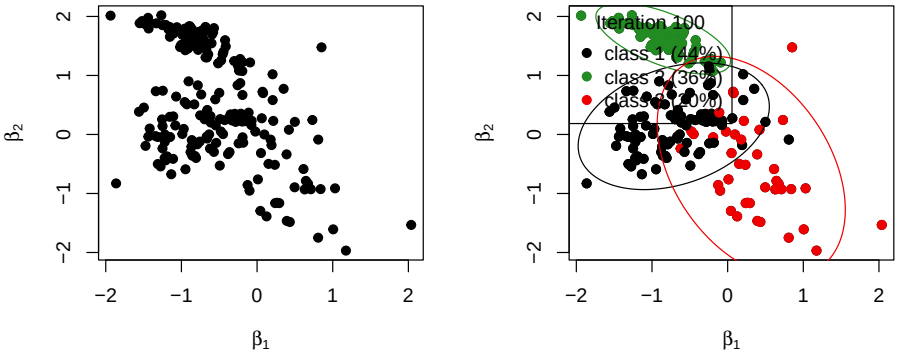
```
> z <- rep(1, ncol(beta))
> C <- 1
> b <- matrix(0, nrow = P_r, ncol = C)
> Omega <- matrix(rep(diag(P_r), C), nrow = P_r*P_r, ncol = C)
```

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)
```

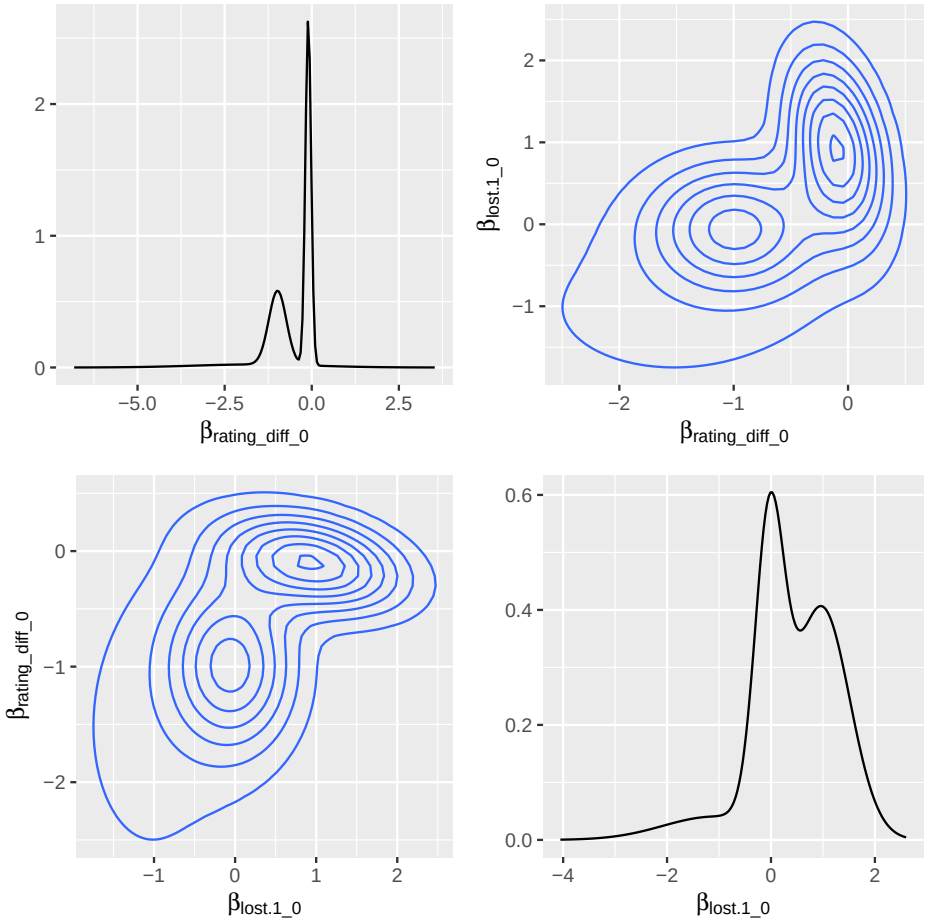


Example 4: Online chess strategy.

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

The estimated pairwise mixing distributions can be visualized via calling the `plot()` method with the additional argument `type = mixture`:

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



5. Choice prediction

RprobitB provides a `predict()` method for in-sample and out-of-sample prediction. 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 our probit model of travelers deciding between two fictional train route alternatives.

Example 1: Train (cont.). Per default, the `predict()` method returns a confusion matrix, which gives an overview of the in-sample prediction performance:

```
> predict(model_train)
```

```
    predicted
true    A    B
   A 1024  450
   B  438 1017
```

By setting the argument `overview = FALSE`, the method instead returns predictions on the level of individual choice occasions:[16]

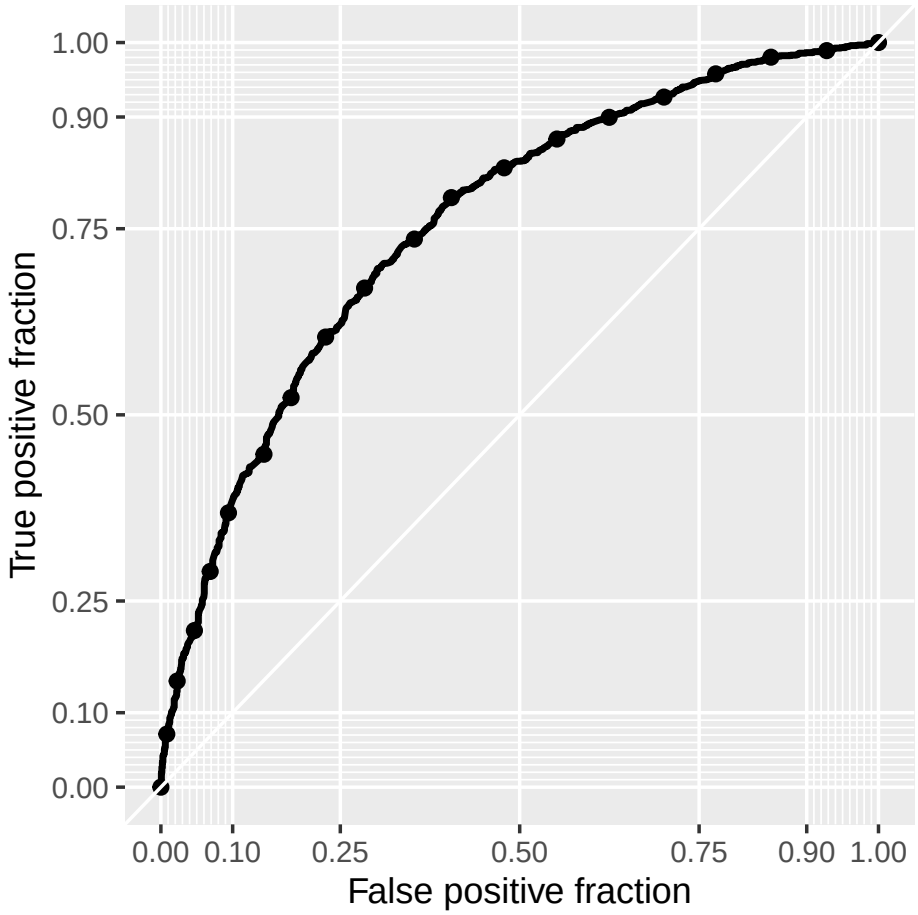
```
> pred <- predict(model_train, overview = FALSE)
> head(pred, n = 5)
```

```
  id choiceid         A          B true predicted correct
1  1        1 0.9157601 0.08423988    A         A    TRUE
2  1        2 0.6373833 0.36261675    A         A    TRUE
3  1        3 0.7918504 0.20814963    A         A    TRUE
4  1        4 0.1799070 0.82009305    B         B    TRUE
5  1        5 0.5494634 0.45053656    B         A   FALSE
```

Apart from the prediction accuracy, the model performance can be evaluated more nuanced in terms of sensitivity and specificity, for example via a receiver operating characteristic (ROC) curve (Fawcett 2006), using the **plotROC** package (Sachs 2017):

```
> 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)
```

[16] Incorrect predictions can be analyzed via the convenience function `get_cov()`, which extracts the characteristics of a particular choice situation.



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 0.3480596 0.6519404          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 0.5689424 0.4310576          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                        -1727.70          -1865.86
AIC                        3463.41           3733.73
```

```
BIC                          3487.34          3739.71
WAIC                         3463.74          3733.87
se(WAIC)                        0.19             0.07
pWAIC                           4.37             1.13
MMLL                        -1731.13         -1867.26
BF(*,model_train)                  1           < 0.01
BF(*,model_train_sparse)       > 100                1
pred_acc                      69.68%           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] -1727.704

> logLik(model_train_sparse)

[1] -1865.863
```

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] 3463.408 3733.725
```

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

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

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] 3487.338 3739.708
```

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

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

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

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

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

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

The WAIC has a standard error of

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

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)
> WAIC(model_train_sparse)
```

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 `fit_model()` 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 `fit_model()` 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
```

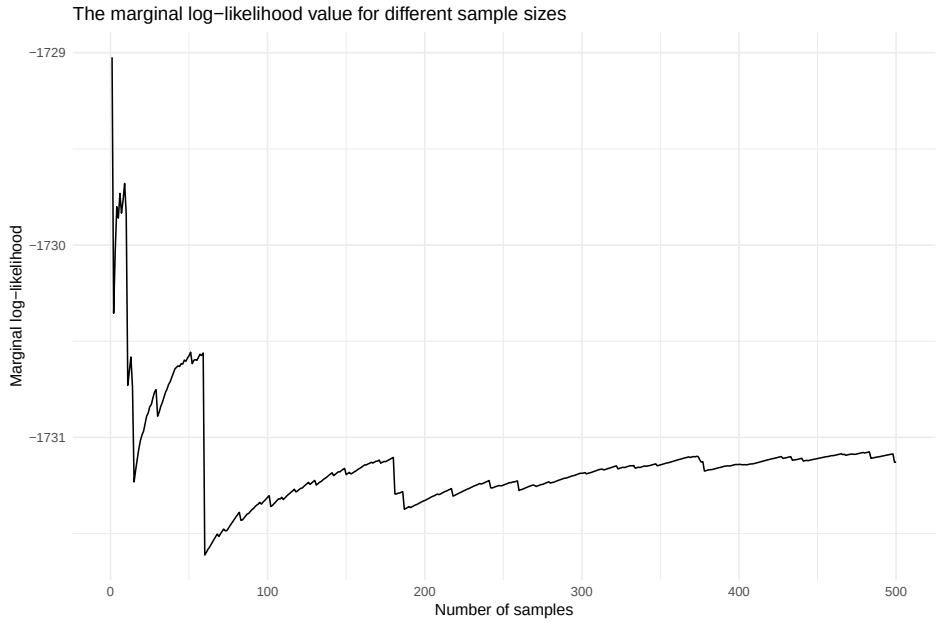
```
9.72e-117 * exp(-1464)
```

```
> attr(model_train$mml, "mmll")
```

```
[1] -1731.128
```

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

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

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"))
```

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] 0.6968249 0.6336634
```

7. 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.

Hewig J, Kretschmer N, Trippe RH, Hecht H, Coles MGH, Holroyd CB, Miltner WHR (2011). "Why humans deviate from rational choice." *Psychophysiology*, **48**(4), 507–514. doi:10.1111/j.1469-8986.2010.01081.x.

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*.

Oelschläger L, Bauer D (2021). *RprobitB: Bayes Estimation of Mixed Multinomial Probit Models*. R package version 1.1.0, URL https://CRAN.R-project.org/package=RprobitB.

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, Dietmar Bauer
Department of Business Administration and Economics
Bielefeld University
Postfach 10 01 31
E-mail: lennart.oelschlaeger@uni-bielefeld.de, dietmar.bauer@uni-bielefeld.de