Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

RprobitB: Bayes Estimation of Choice Behavior Heterogeneity in R

Lennart Oelschläger
Bielefeld University

Dietmar Bauer
Bielefeld University

Abstract

RprobitB is an R package for Bayes estimation of probit choice models for cross-sectional and panel data. It enables the analysis of binary, multivariate, ordered, and ranked choices, and places a special focus on modeling choice behavior heterogeneity among deciders. The main functionality includes choice data management, choice data simulation, model estimation via Markov Chain Monte Carlo methods, tools for convergence diagnostic, model selection via information criteria and Bayes factors, in-sample and out-of-sample choice prediction, and preference-based decider classification via the latent class model extension. The class number can be inferred via the implemented Dirichlet process or a weight-based updating scheme. This article illustrates the package functionality on the basis of four empirical choice data sets.

Keywords: discrete choice, probit model, latent classes, choice behavior heterogeneity, Bayes estimation, preference classification, R.

1. Introduction

Discrete choice models aim to explain past and ultimately predict future choice behavior. They do so by connecting observed choices to observed covariates that influence the decision, for example attributes of the choice alternatives or decider's socio-demographic characteristics. While some influencing characteristics are easily ascertainable through surveys, others are not. For example, car buyers might base their purchase decision between a low-emission car versus an SUV on their green life propensity. Such a political attitude is hard to quantify, so it is typically not queried, in contrast to characteristics like income and household size. But not accounting for such unobserved choice behavior heterogeneity generally leads to inferior model fit and prediction quality. The presented R package **RprobitB**[1] (Oelschläger and Bauer 2021)

[1]The name **RprobitB** is a portmanteau of the language R, the probit model, and the Bayes framework.

provides state-of-the-art tools for modeling unobserved taste heterogeneity in the context of discrete choices.

We interpret discrete choice models as so-called random utility models. In this framework, the decider's latent utility for the available alternatives is decomposed into a function of the covariates, in our case a linear combination of those, and a random error term. The error term distribution determines the model type, where **RprobitB** implements the probit model with a joint normal distribution across alternatives (as opposed to the logit model which assumes independent extreme value distributions). The coefficients in the linear combination represent the ceteris paribus effect of the covariates on the utility, and ratios of coefficients quantify substitution patters, for example the willingness to pay more money for a lower CO2 emission rate.

Constant coefficients across deciders lead to substitution patterns that again are constant, which is unreasonable in many scenarios. Instead, we can assume that the coefficients are drawn from an underlying distribution, a so-called mixing distribution. This specification allows for decider-specific coefficients and hence flexible taste heterogeneity, cf. Train (2009) and Bhat (2011). **RprobitB** implements the recently proposed approach of Oelschläger and Bauer (2020) to approximate the underlying mixing distribution by a mixture of Gaussian densities, leading to the latent class mixed probit model.

The latent class model extension also enables preference-based decider classification, i.e. identifying groups of deciders that share the same preferences. However, class interpretation demands a proper specification of the total class number. While a trial-and-error strategy in conjunction with likelihood-based model selection is theoretically possible, **RprobitB** offers two approaches that avoid pre-specifying the class number: weight-based class updates within the estimation routine (Oelschläger and Bauer 2020) and class updates based on the Dirichlet process, similar to Burda *et al.* (2008).

Model fitting in **RprobitB** takes places in a Bayesian framework via Markov chain Monte Carlo simulation. This approach has several benefits compared to frequentist inference: it does not need to compute the probit likelihood (which is not in closed form and hence would require approximation), it avoids numerical challenges associated with finding the likelihood optimum, it enables to impose prior believes on the model parameters, and it provides posterior parameter distributions instead of point estimates only. Additionally, the Bayesian approach was shown to be computationally faster with increasing number of random effects and normal mixing distributions than the maximum likelihood approach, cf. Train (2001) for a simulation study in the logit case.

RprobitB extends the collection of already available software for estimating probit choice models. **Rchoice** (Sarrias 2016) and **mlogit** (Croissant 2020) are two R packages for maximum likelihood estimation (MLE) of the (mixed) probit model. They also implemented the logit model. The Python library **Biogeme** (Bierlaire 2020) can estimate latent class (LC) models. The **apollo** package (Hess and Palma 2019) allows for flexible logit and probit model specifications, with both maximum likelihood and Bayes estimation. **bayesm** (Rossi 2019) and **MNP** (Imai and van Dyk 2022) are two alternatives for Bayes estimating of the probit model, both implementing Markov chain Monte Carlo simulation methods similar to **RprobitB**. **RprobitB** complements the collection by providing estimation of latent class mixed probit models without the need of pre-specifying the class number, as outlined above. The software comparison is summarized in Table 1.

		Probit	Logit	Bayes	MLE	Mixed	LC	LC update
Rchoice	R	✓	✓		✓	✓		
mlogit	R	✓	✓		✓	✓		
Biogeme	Python	✓	✓		✓	✓	✓	
apollo	R	✓	✓	✓	✓	✓	✓	
bayesm	R	✓	✓	✓		✓		
MNP	R	✓		✓				
RprobitB	R	✓		✓		✓	✓	✓

Table 1: Overview of publicly available software for estimating discrete choice models.

This article provides a general description of **RprobitB** and is structured as follows. To fix our notation, Section 3 defines the probit model and formalizes the concepts of mixing distributions and latent classes. Sections 4 - 6 describe the package functionality, including choice data preparation, simulation, model fitting, choice prediction, preference classification, and model selection (the main functions for these tasks are visualized in Figure 1). For illustration, we use four different empirical data sets: the two stated-choice data sets `Train` and `Electricity` from the **mlogit** package, a revealed-choice data set about online chess strategy that is contained in **RprobiB**, and ordered choice data from the German family panel pairfam (Brüderl *et al.* 2020). Section 7 concludes and gives an outlook of anticipated package extensions.

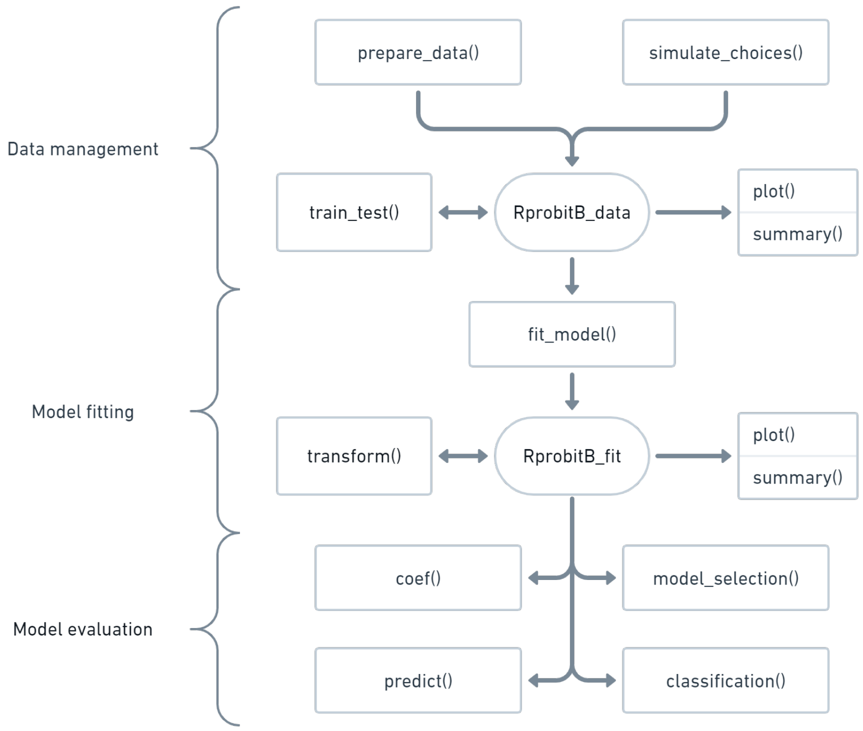


Figure 1: Flowchart of the main **RprobitB** functionalities (rectangles) and objects (ovals).

2. The probit model

This section briefly defines the probit model. For the model's data input, assume that we know the choices of N deciders choosing between $J \geq 2$ alternatives at each of T choice occasions.[2] Specific to each decider, alternative and choice occasion, we observe P covariates, a linear combination of which explains the choices:

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

for $n = 1, \dots, N$, $t = 1, \dots, T$ and $j = 1, \dots, J$. Here, X_{ntj} is a (column) vector of P characteristics specific to alternative j as faced by decider n at choice occasion t, $\beta_n \in \mathbb{R}^P$ is the coefficient vector of n, 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.

The value U_{ntj} on the left-hand side of equation (1) can be interpreted as the decider's utility. It is unobserved by the researcher, but we assume that the deciders know their utilities for each alternative and make a choice which is consistent with utility maximization.[3] Therefore, we link

$$y_{nt} = \underset{j=1,\dots,J}{\operatorname{argmax}}\, U_{ntj}, \tag{2}$$

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

Equation (1) has a decider-specific coefficient vector β_n. Some entries of β_n can be fixed across deciders, in which case the coefficient vector is of the form (α, β_n), where α are P_f coefficients that are constant across deciders and β_n are P_r decider-specific coefficients, $P_f + P_r = P$. The decider-specific coefficients are assumed to be realizations of an underlying distribution, a so-called mixing distribution (Train 2009, Ch. 6). This distribution characterizes heterogeneity among the deciders and allows for individual sensitivities as motivated in the introduction.

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, pp. 136 ff.). Instead, **RprobitB** implements the approach of Oelschläger and Bauer (2020) to approximate any underlying mixing distribution by a mixture of P_r-variate Gaussian 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 allows for decider classifications, see Section 4.6 for an example.

[2]The number T of choice occasions is the same for each decider here for notational simplicity. However, **RprobitB** allows for unbalanced panels, i.e. varying T. Of course, the cross-sectional case $T = 1$ is possible.

[3]Utility maximizing behavior is a common assumption in econometric models. However, studies have shown that humans do not decide in this rational sense in general (Hewig *et al.* 2011).

Finally, the probit model requires normalization. This is because any utility model is invariant towards the level and the scale of utility (Train 2009, Ch. 2). We therefore normalize the model by taking utility differences and fixing one error term variance. Formally, equation (1) is transformed to

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

$j = 1, \dots, J-1$, where (choosing J as the reference alternative), $\tilde{U}_{ntj} = U_{ntj} - U_{ntJ}$, $\tilde{X}_{ntj} = X_{ntj} - X_{ntJ}$, and $\tilde{\epsilon}_{ntj} = \epsilon_{ntj} - \epsilon_{ntJ}$. The error term differences $(\tilde{\epsilon}_{nt:}) = (\tilde{\epsilon}_{nt1}, ..., \tilde{\epsilon}_{nt(J-1)})'$ again are multivariate normally distributed with mean 0 but transformed covariance matrix $\tilde{\Sigma}$, in which one diagonal element is fixed to a positive number.[4]

3. Choice data

RprobitB requests that choice data sets are (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 (and optionally each choice occasion), (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, Ch. 2). Alternatives can be considered as ordered (e.g. the level of agreement on a Likert-type rating scale, cf. Section 3.4), and additionally full rankings of the alternatives can be provided (e.g. when asking the respondent to rank all available alternatives from best to worst, cf. Section 3.5).

3.1. Different types of covariates

Different covariate types can be considered: covariates that are constant across alternatives (e.g. a car buyer's income), covariates that are alternative specific (e.g. the car's price), covariates with a generic coefficient (e.g. paying the same amount of money for car company A versus B should make no difference), and covariates that have alternative specific coefficients (e.g. the range of an electric car might be of more importance than for other types of propulsion). To allow for these different types, we generalize equation (1) to

$$U_{ntj} = \beta_{0j} + A_{ntj}\beta_1 + B_{nt}\beta_{2j} + C_{ntj}\beta_{3j} + \epsilon_{ntj}. \tag{3}$$

Here, the covariates A and C depend on the alternative j, while B is only decider and choice occasion specific. The coefficient β_1 for A is generic (i.e. the same for each alternative), whereas β_{2j} and β_{3j} for B and C are alternative specific. The intercept $\beta_0 j$ is called alternative specific constant (ASC). ASCs capture the average effect on the alternative's utility of all factors that are not included in the model.

The full collections $(\beta_{0j})_{j=1,...,J}$ of ASCs and $(\beta_{2j})_{j=1,...,J}$ of coefficients for covariate type B are not identified. This is because we took utility differences for model normalization (cf. Section 2), and hence one coefficient is a linear combination of the others, respectively. We

[4]Fixing one element of $\tilde{\Sigma}$ determines the utility scale. Fixing one fixed effect (i.e. one entry of α) serves the same purpose. Both alternatives are implemented in **RprobitB**, see Section 4.2.

therefore fix β_{0k} and β_{2k} to 0 for one base alternative k. The coefficients $(\beta_{0j})_{j\neq k}$ and $(\beta_{2j})_{j\neq k}$ then have to be interpreted with respect to k.

3.2. Formula framework

Specifying equation (3) in R requires a flexible formula framework. **RprobitB** can interpret a '`formula`' object of the form `choice ~ A | B | C`, where `choice` is the name of the dependent variable (the discrete choice we aim to explain), and `A`, `B`, and `C` are the different covariate types of Section 3.1.[5]

The framework has the following rules. ASCs are added to the model per default. 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 | 1 | C`. The expression `choice ~ A | 0 | C` is interpreted as no covariates of the second category and no alternative specific constants.

3.3. 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 in Section 3.2:

```
> 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()` (that we introduce in Section 4). The following arguments of `prepare_data()` are optional:

- `re`: A character vector of covariate names in `form` with random effects. Per default `re = NULL`, i.e. no random effects.
- `alternatives`: A character vector of the alternative names, defining the choice set. If not specified, all alternatives chosen in `choice_data` are considered.
- `ordered` and `ranked`: Two booleans, that are set to `FALSE` per default. If set to `TRUE`, the alternatives are interpreted as ordered, and the choices are interpreted as ranked, respectively. See Sections 3.4 and 3.5 for details.
- `base`: One element of `alternatives` specifying the base alternative (cf. Section 3.1). Per default, `base` 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 `choice_data`.

[5] This formula framework is adapted from **mlogit**.

- `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 (none per default).
- `impute`: A character, specifying how to handle missing covariates in `choice_data`. Options are `"complete_cases"` (removing rows that contain missing entries, which is the default behavior), `"zero"` (replacing missing entries by 0), and `"mean"` (imputing missing entries by the covariate mean).

Example 1: Train trips. The **mlogit** package contains the data set `Train` with 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 `changes`. 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 ...
```

For demonstration, we include all choice characteristics into our probit model, connect them to generic and fixed coefficients, and exclude ASCs:

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

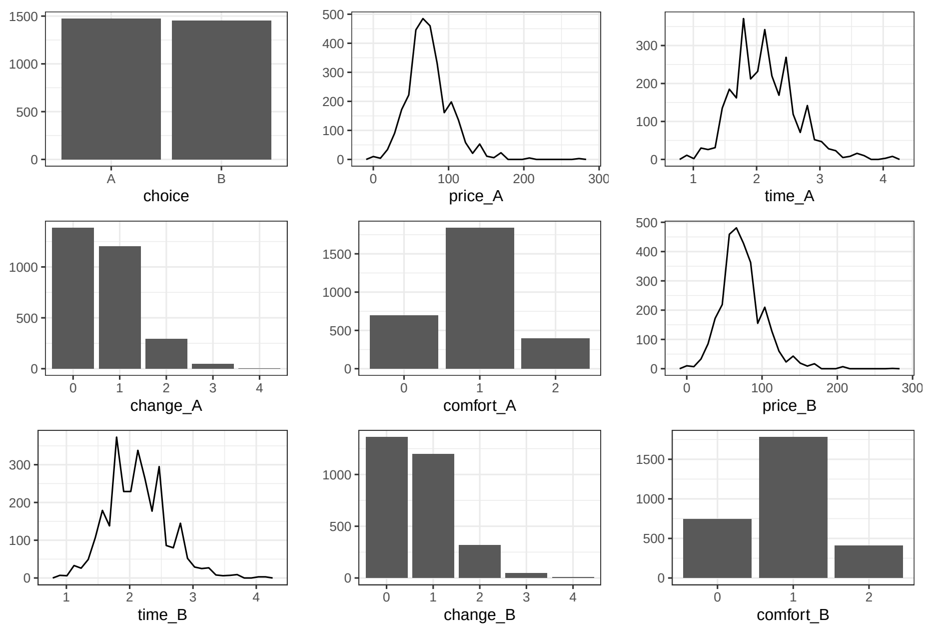
Passing `form` to `prepare_data()` returns an 'RprobitB_data' object, which in turn can be fed into the estimation routine `fit_model()` (cf. Section 4):

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

The data object can be inspected via its `summary()` and `plot()` methods:

```
> summary(data_train)

                  count
deciders            235
choice occasions   5-19
total choices      2929
alternatives          2
- 'A'              1474
- 'B'              1455
```

```
> plot(data_train)
```



3.4. Ordered alternatives

The two choice alternatives in Example 1 are unordered. If we had asked "rate your train trip from 1 (horrible) to 7 (great)", than the respondents would choose their answer from a set of ordered alternatives. Ordered alternatives can by analyzed in **RprobitB** by setting `ordered = TRUE` in `prepare_data()`. In this case, `alternatives` becomes a mandatory argument, with the alternative names ordered from worst to best.

The concept of decider's having separate utilities for each alternative is no longer natural for the ordered case (Train 2009, Ch. 7.4). Instead, we model only one utility value

$$U_{nt} = X_{nt}'\beta_n + \epsilon_{nt}$$

per decider n and choice occasion t, which we interpret as the "level of association" that n has with the choice question. The utility value falls into discrete categories, which in turn are linked to the ordered alternatives $j = 1, \dots, J$. Formally, the link in equation (2) gets replaced by

$$y_{nt} = \sum_{j=1,\dots,J} j \cdot I(\gamma_{j-1} < U_{nt} \leq \gamma_j),$$

with end points $\gamma_0 = -\infty$ and $\gamma_J = +\infty$, and new model coefficients $(\gamma_j)_{j=1,\dots,J-1}$.

Normalizing the ordered probit model with respect to the utility level differs from the unordered case: since we model only one utility value, we can no longer take utility differences, but instead fix $\gamma_1 = 0$. For scale normalization however, we again can either fix the variance of the (now univariate) normal distribution of the error term ϵ_{nt}, or fix a fixed effect. Both options are available in **RprobitB** through the `scale` argument for the estimation routine `fit_model()` (cf. Section 4.2).[6]

Example 2: Child wish. Missing

3.5. Ranked choices

Missing

3.6. Simulating choice data

The `simulate_choices()` function simulates choice data from a pre-specified probit model. Say we want to simulate the choices of `N` deciders in `T` choice occasions[7] 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`, `base`, `standardize`: Analogue to `prepare_data()` (cf. Section 3.3).
- `alternatives`: A character vector of length `J` with the names of the choice alternatives (per default 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. Unspecified covariates are drawn from a standard normal distribution.
- `seed`: Optionally set a seed for the simulation.

The true model parameters are set at random per default. Alternatively, they can be specified via a named list that is passed to the function's `true_parameter` argument and contains:

- a numeric vector `alpha` with the fixed effects,

[6]Note that there is even a third option for scale normalization: restricting the value space for the thresholds to a bounded interval also serves the purpose, but is not implemented.

[7]`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

- the number C of latent classes (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_full (Sigma), the (differenced) 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")
> form <- choice ~ var1 | var2 | var3
> re <- c("ASC","var2")
```

The overview_effects() function provides an overview of the effect types:

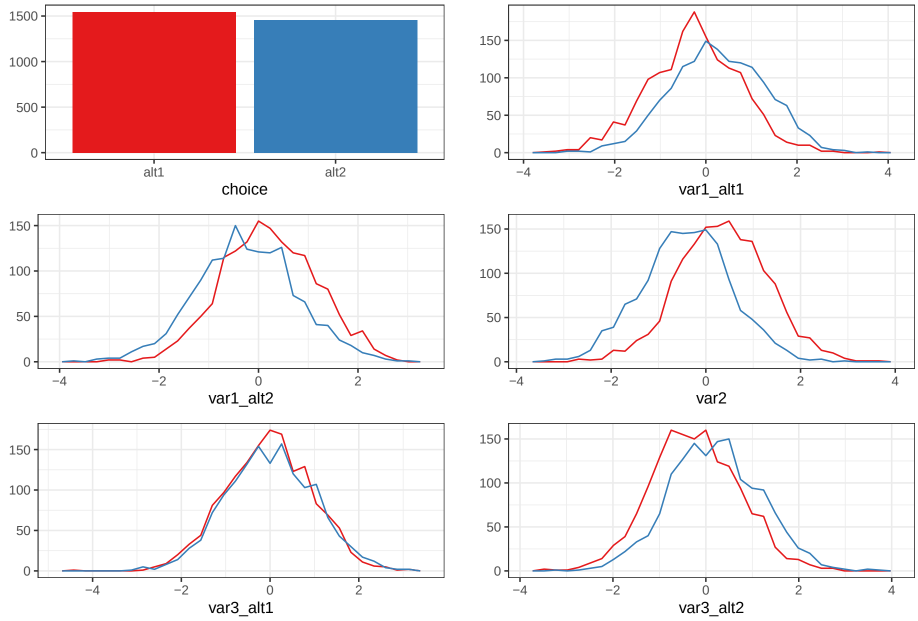
```
> overview_effects(form = form, re = re, alternatives = alternatives)

     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), consequently 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 random 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, which we will reproduce in Sections 4.4 and 4.5:

```
> data_sim <- simulate_choices(
+    form = form, N = N, T = T, J = 2,
+    re = re, alternatives = alternatives, seed = 1,
+    true_parameter = list(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`' objects 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).

4. Model fitting

RprobitB estimates the probit model in a Bayesian framework that 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 via Gibbs sampling.

In the following, we list the prior distributions for the model parameters, formulate the conditional posterior distributions, introduce the estimation routine `fit_model()`, and apply it to the two examples from the previous section. The remainder of this section is devoted to the estimation of latent class models and the implemented class updating schemes.

4.1. Prior and posterior distributions

We a priori assume the following (conjugate) parameter distributions:

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

- $\alpha \sim \text{MVN}_{P_f}(\psi, \Psi)$, 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 $\tilde{\Sigma} \sim W_{J-1}^{-1}(\kappa, \Lambda)$.

These priors imply the following conditional posterior distributions (we are closely following Oelschläger and Bauer (2020)):

- 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 $m_c = \#\{n : z_n = c\}$ denotes the current absolute size of class c. The model is invariant to permutations of the class labels $1, \dots, C$. We therefore accept an update only if the ordering $s_1 > \dots > s_C$ still holds (thereby ensuring a unique class labeling).

- The allocation variables $(z_n)_n$ are updated independently for all n via

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

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

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

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

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

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

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

- Independently for all n, t and conditionally on the other components, the differenced utility vectors $(\tilde{U}_{nt:})$ follow a $(J-1)$-variate truncated 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 (analogue to Geweke (1998)):

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

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

$$\mu_{\tilde{U}_{ntj}} = \tilde{W}'_{ntj}\alpha + \tilde{X}'_{ntj}\beta_n - \Sigma_{\tilde{U}_{ntj}}(\tilde{\Sigma}^{-1})_{j(-j)}(\tilde{U}_{nt(-j)} - \tilde{W}'_{nt(-j)}\alpha - \tilde{X}'_{nt(-j)}\beta_n),$$

  where $(\tilde{\Sigma}^{-1})_{jj}$ denotes the (j,j)-th element of $\tilde{\Sigma}^{-1}$, $(\tilde{\Sigma}^{-1})_{j(-j)}$ the j-th row without the j-th entry, $\tilde{W}_{nt(-j)}$ and $\tilde{X}_{nt(-j)}$ the differenced covariate matrices connected to fixed and random effects, respectively, with the j-th column removed.

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

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

$$\mu_\alpha = \Sigma_\alpha(\Psi^{-1}\psi + \textstyle\sum_{n=1,t=1}^{N,T} \tilde{W}_{nt}\tilde{\Sigma}^{-1}((\tilde{U}_{nt:}) - \tilde{X}'_{nt}\beta_n)),\ \Sigma_\alpha = (\Psi^{-1} + \textstyle\sum_{n=1,t=1}^{N,T} \tilde{W}_{nt}\tilde{\Sigma}^{-1}\tilde{W}'_{nt})^{-1}.$$

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

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

$$\mu_{\beta_n} = \Sigma_{\beta_n}(\Omega_{z_n}^{-1} b_{z_n} + \textstyle\sum_{t=1}^{T} \tilde{X}_{nt}\tilde{\Sigma}^{-1}(\tilde{U}_{nt} - \tilde{W}'_{nt}\alpha)),\ \Sigma_{\beta_n} = (\Omega_{z_n}^{-1} + \textstyle\sum_{t=1}^{T} \tilde{X}_{nt}\tilde{\Sigma}^{-1}\tilde{X}_{nt}{}')^{-1}.$$

- The covariance matrix $\tilde{\Sigma}$ of the error term differences is updated by means of

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

where $S = \sum_{n=1,t=1}^{N,T} \tilde{\varepsilon}_{nt}\tilde{\varepsilon}'_{nt}$ and $\tilde{\varepsilon}_{nt} = (\tilde{U}_{nt:}) - \tilde{W}'_{nt}\alpha - \tilde{X}'_{nt}\beta_n$.

The Gibbs samples obtained from this updating scheme (except for s and z draws) lack identification w.r.t. scale (cf. Section 2). Subsequent to the sampling and for the i-th updates in each iteration i, we therefore apply the normalization $\alpha^{(i)} \cdot \omega^{(i)}$, $b_c^{(i)} \cdot \omega^{(i)}$, $\tilde{U}_{nt}^{(i)} \cdot \omega^{(i)}$, $\beta_n^{(i)} \cdot \omega^{(i)}$, $\Omega_c^{(i)} \cdot (\omega^{(i)})^2$, and $\tilde{\Sigma}^{(i)} \cdot (\omega^{(i)})^2$, where either $\omega^{(i)} = \sqrt{\text{const}/(\tilde{\Sigma}^{(i)})_{jj}}$ with $(\tilde{\Sigma}^{(i)})_{jj}$ the j-th diagonal element of $\tilde{\Sigma}^{(i)}$, $1 \leq j \leq J-1$, 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 a constant to specify custom utility scales.

4.2. The estimation routine

The Gibbs sampling scheme can be executed via the function call

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

where `data` is an '`RprobitB_data`' object (cf. Section 3). Optional arguments are:

- `scale`: A formula object, which determines the utility scale (cf. Section 2). 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 in Section 4.1). 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 (`B = R/2` per default).[8]

[8] 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).

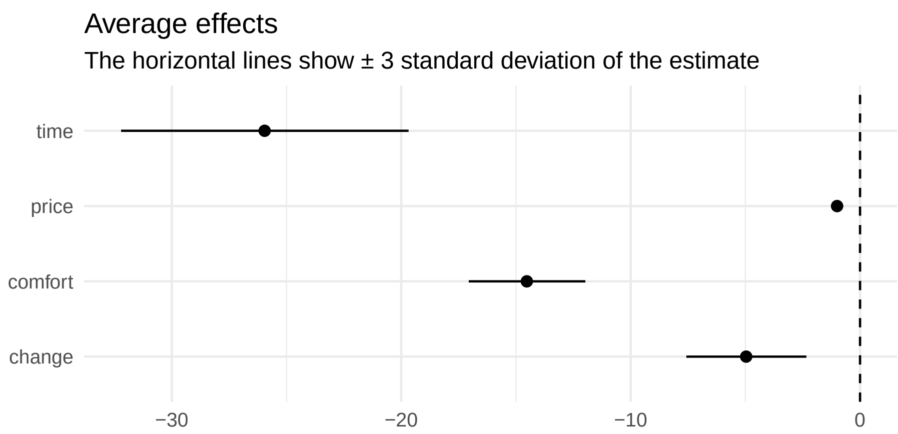
- `Q`: The thinning factor for the Gibbs samples (`Q = 1` per default).
- `print_progress`: A boolean, determining whether to print the Gibbs sampler progress.
- `prior`: A named list of parameters for the prior distributions. Default values are documented in the `check_prior()` function, see `help(check_prior, package = "RprobitB")`.

Example 1: Train trips (cont.). Recall the `Train` data set of stated train trip alternatives, characterized by their `price`, `time`, number of `changes`, 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 `price` coefficient to `-1`, which has the advantage that we can interpret the other coefficients as monetary values:

```
> model_train <- fit_model(data = data_train, R = 1000, scale = price ~ -1)
```

The estimated coefficients (using the mean of the Gibbs samples as a point estimate) can be 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 15 euros.[9] 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 arbitrary functions that can compute point estimates of the Gibbs samples, per default `mean()` for the arithmetic mean, `stats::sd()` for the standard deviation, and `R_hat()` for the Gelman-Rubin statistic (Gelman and Rubin 1992)[10]:

```
> FUN <- c("mean" = mean, "sd" = stats::sd, "R^" = RprobitB::R_hat)
> summary(model_train, FUN = FUN)
```

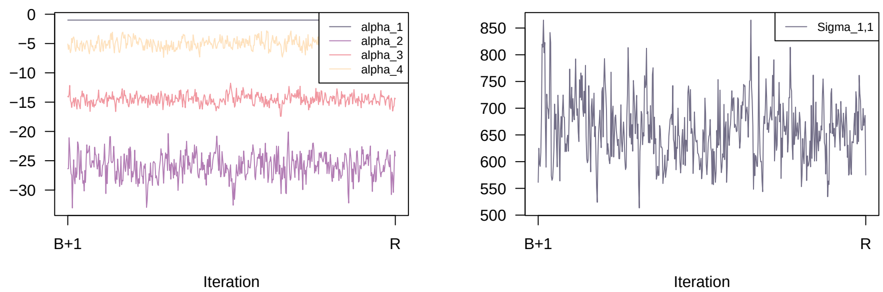
[9]We note that these results are consistent with the ones that are presented in a vignette of **mlogit** entitled "The random parameters (or mixed) logit model" on the same data set but using the logit model.

[10]A Gelman-Rubin statistic (a lot) greater than 1 indicates convergence issues of the Gibbs sampler.

```
Probit model
Formula: choice ~ price + time + comfort + change | 0
R: 1000, B: 500, Q: 1

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^
 alpha

     1   -1.00    0.00    1.00
     2  -25.95    2.08    1.00
     3  -14.52    0.84    1.00
     4   -4.96    0.87    1.03

 Sigma

   1,1  660.03   58.54    1.00
```

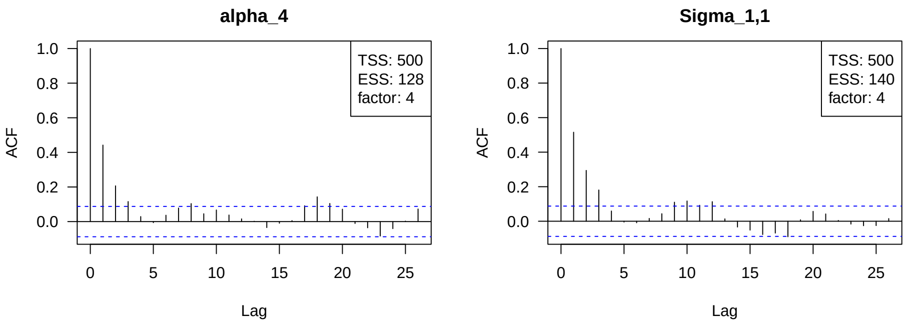
Calling the `plot()` method with the additional argument `type = "trace"` plots the trace of the transformed and thinned Gibbs samples after the burn-in:

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



Additionally, we can visualize the autocorrelation of the Gibbs samples via the argument `type = "acf"`, below exemplary for `alpha_4` and `Sigma_1,1`). The boxes in the plot's 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 autocorrelation of the Gibbs samples (Marin and Robert 2014). The autocorrelations are estimated via the `stats::acf()` function.

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



To obtain more independent samples, the `transform()` method can be used to increase the thinning factor:[11]

```
> model_train <- transform(model_train, Q = 5)
```

4.3. Estimating a joint normal mixing distribution

We demonstrate how to estimate a joint normal mixing distribution in **RprobitB** on the basis of another real-data example. To enable comparison across methods and implementations, we use another data set from **mlogit**. Their results (using the logit model) are documented in the package vignette entitled "Exercise 3: Mixed logit model".

Example 3: Electricity suppliers. The `Electricity` data set from **mlogit** contains choices of residential electricity customers that were asked to decide between four contract offers of hypothetical electricity suppliers. Heterogeneity in choice behavior is expected here, because customers might value certain contract characteristics differently based on their living conditions. In particular, the contract offers differed in 6 characteristics: their fixed price `pf` per kilowatt hour, their contract length `cf`, whether the supplier is a local company (boolean `loc`), whether the supplier is a well known company (boolean `wk`), whether the supplier offers a time-of-day electricity price which is higher during the day and lower during the night (boolean `tod`), and whether the supplier's price is seasonal dependent (boolean `seas`).

The following lines prepare the 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
+  )
```

[11]The function can also be used to increase the length of the burn-in period (via `transform(model_train, B = B_new)`) or to change the utility scale, for example `transform(model_train, scale = Sigma_1    1)`.

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 again will fix to `-1` to interpret the other estimates as monetary values (cf. Example 1):

```
> 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 = 1000, scale = pf ~ -1)
```

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)     0.31 (0.03)
3  loc       2.79 (0.25)     7.43 (1.25)
4   wk       2.07 (0.21)     3.84 (0.67)
5  tod      -9.70 (0.21)    10.72 (1.32)
6 seas      -9.89 (0.18)     6.25 (1.03)
```

We can for example deduce, that a longer contract length has a negative effect on average (-0.25). However, our model shows that 32% 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] 0.3249726
```

The estimated joint mixing distribution additionally 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 correlation of 0.79 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 0.09 0.07 -0.04 -0.10
loc   0.09 1.00 0.79  0.13  0.04
wk    0.07 0.79 1.00  0.14  0.03
tod  -0.04 0.13 0.14  1.00  0.55
seas -0.10 0.04 0.03  0.55  1.00
```

4.4. Estimating a latent class model

RprobitB allows to specify a Gaussian mixture as the mixing distribution, which allows for (a) a flexible 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). 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 prespecified.

Example 2: Simulated choices (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
Formula: 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
```

Comparing the columns of true parameters (`true`) and Gibbs sample means (`mean`), we deduce that the model parameters (especially those characterizing the latent classes) can be estimated consistently.

4.5. Weight-based update of the latent classes

Adding `"weight_update" = TRUE` to the list for the `latent_classes` argument of `fit_model()` executes the following weight-based updating scheme of the latent classes (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.

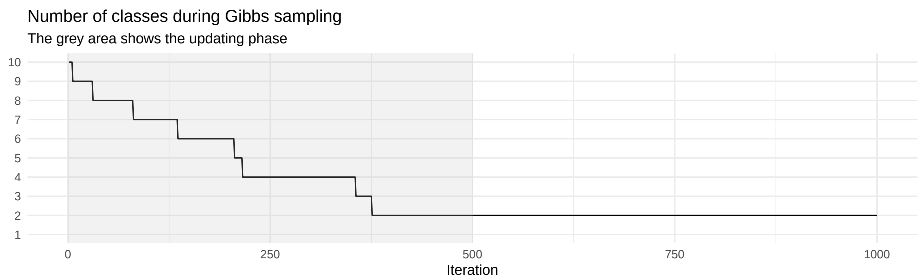
The values for ε_{min}, ε_{max} and $\varepsilon_{\text{distmin}}$ can be specified via the `latent_classes` argument (per default `epsmin = 0.01`, `epsmax = 0.99`, and `distmin = 0.1`).

Example 2: Simulated choices (cont.). For our simulation example, we additionally specify `"C" = 10` (the initial number of latent classes) and `"buffer" = 5` (to execute the updating scheme only in every $buffer$-th iteration):

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

The updating behavior of the class numbers can be visualized as follows:

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



4.6. Dirichlet process-based update of the latent classes

The Dirichlet process is a Bayesian nonparametric method that adds as many mixture components to the mixing distribution as needed for a good approximation. We briefly formulate the theory and refer to Neal (2000) for more details.

A priori, the mixture weights $(s_c)_c$ are given a Dirichlet prior with concentration parameter δ/C. For the class allocation variables z, Rasmussen (2000) shows that

$$\Pr(z \mid \delta) = \frac{\Gamma(\delta)}{\Gamma(N+\delta)} \prod_{c=1}^{C} \frac{\Gamma(m_c + \delta/C)}{\Gamma(\delta/C)}, \tag{4}$$

where $\Gamma(\cdot)$ denotes the gamma function and m_c the size of class c. Crucially, equation (4) is independent of the class weights $(s_c)_c$ (in contrast to the conditional posterior distribution stated in Section 4.1). From this equation, Li *et al.* (2019) shows that

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

where the limit is taken as C approaches infinity, and z_{-n} denotes the vector z without the n-th element. Now,

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

equals the probability that a new cluster for observation n is created. This probability is directly proportional to the prior parameter δ (Neal 2000): a greater value for δ encourages the creation of new clusters, smaller values increase the probability of an allocation to an already existing class. The number of clusters can theoretically rise to infinity, however, as we delete unoccupied clusters, C is bounded by N.

The Dirichlet process directly integrates into our existing Gibbs sampler: given $(\beta_n)_n$, we update the class means b_c and covariance matrices Ω_c by means of their posterior predictive distribution. The mean vector and covariance matrix for new generated clusters are drawn from their prior predictive distribution (Li *et al.* (2019) provides the formulas). The full updating scheme is implemented in the function `update_classes_dp()` and can be executed within the estimation routine `fit_model()` by adding `dp_update = TRUE` to the list argument for `latent_classes`.

Example 4: Online chess strategy. We demonstrate the Dirichlet process updating scheme via an example from online chess. **RprobitB** containes revealed gambling preference data of chess players in the yearly bullet arena 2022 on the online chess platform https://lichess.org: at the beginning of each game, both players can choose to trade half of their clock time[12] against the option to win an extra tournament point in case they win the game. The tournament lasted 4 hours, participants were paired again immediately after they finished a game, and the player with the most tournament points in the end won the event. The platform calls the trade clock time against a potential extra tournament "berserking". Several questions regarding the trade "clock time against a potential extra tournament" (which the platform calls "berserking") immediately arise: Do higher-rated chess players prefer to gamble? Does the remaining tournament time have an influence on the berserking choice? Can players be classified based on their revealed preferences to berserk?

The `choice_berserk` data set provides the following information: whether a player berserked (`berserk = 1` if yes), wether they had the `white` pieces, their `rating` (a value provided by the platform indicating the playing strength), the rating difference `rating_diff` to the opponent, whether they lost the game (`lost = 1` if yes), the remaining tournament time `min_rem` in minutes, and whether they are currently on a winning `streak` (which gives extra points). We additionally consider the lagged covariates `berserk.1` (the berserking choice in the previous game) and `lost.1` (the result of a player's previous game), which can be created via the convenience function `choice_berserk()`. We specify random effects for the rating difference and the result of the previous game, aiming to classify the chess players based on their berserking choice to these circumstances.

```
> choice_berserk <- create_lagged_cov(
+    choice_data = RprobitB::choice_berserk,
+    column = c("berserk","lost"), k = 1, id = "player_id"
+  )
> data <- prepare_data(
+    form = berserk ~ 0 | white + rating + rating_diff + min_rem + streak +
+      berserk.1 + lost.1 + 1,
+    re = c("rating_diff","lost.1"), choice_data = choice_berserk,
+    id = "player_id", idc = "game_id",
+    standardize = c("rating","rating_diff","min_rem"), impute = "zero"
+  )
> model_berserk <- fit_model(
```

[12]Both players start a game with a time credit of one minute, which is consumend when it's their turn to make a move. A player whos time runs up looses the game automatically.

```
+     data, latent_classes = list("dp_update" = TRUE, "C" = 10), R = 5000
+   )
```

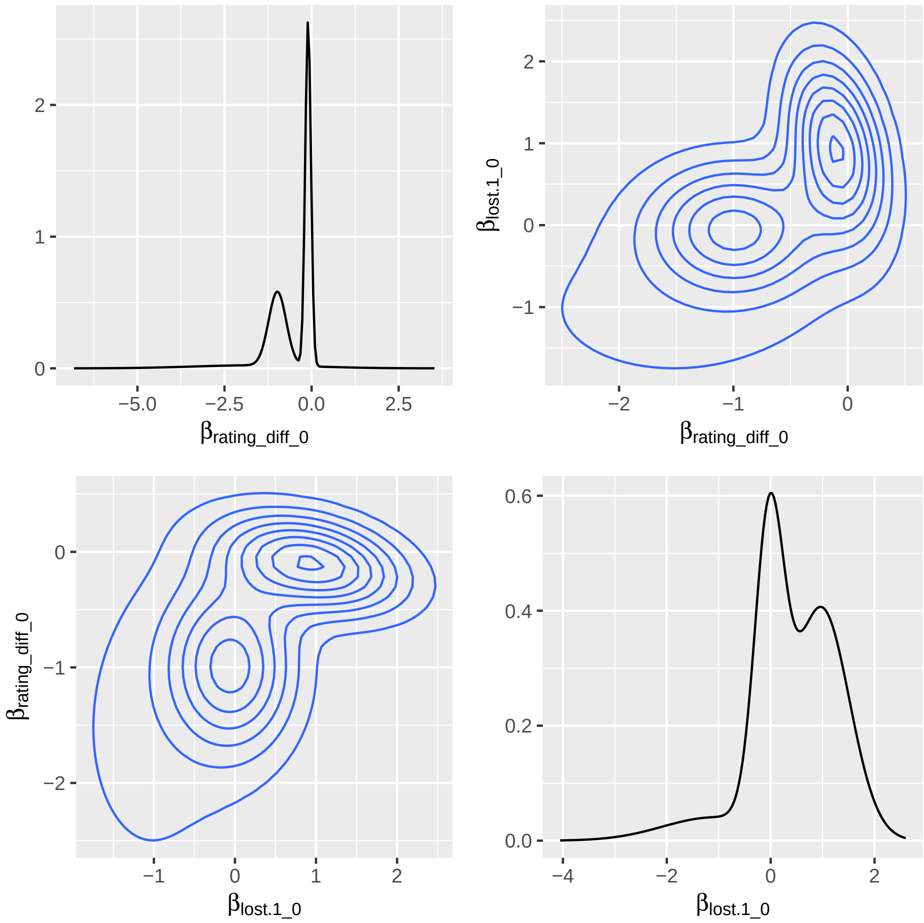
Estimating this model with $N = 6174$ deciders, $T = 1$ to 177 choice occasions and 126902 choices in total took about 4 hours computation time. For convenience, we pre-computed the model and saved the resulting `model_berserk` object in the package:

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

```
                    Estimate   (sd) Variance   (sd)
1           white_0     0.04 (0.02)       NA   (NA)
2          rating_0     0.11 (0.01)       NA   (NA)
3         min_rem_0    -0.04 (0.01)       NA   (NA)
4          streak_0     0.27 (0.03)       NA   (NA)
5       berserk.1_0    -1.21 (0.02)       NA   (NA)
6             ASC_0     2.05 (0.03)       NA   (NA)
7  rating_diff_0 [1]   -0.10 (0.02)     0.08 (0.01)
8  rating_diff_0 [2]   -0.98 (0.06)     0.25 (0.05)
9  rating_diff_0 [3]   -1.65 (0.21)     1.72 (0.32)
10      lost.1_0 [1]    0.98 (0.09)     0.54 (0.10)
11      lost.1_0 [2]   -0.03 (0.08)     0.28 (0.06)
12      lost.1_0 [3]   -1.09 (0.18)     0.99 (0.21)
```

The classes can be visualized via calling the `plot()` method with the additional argument `type = mixture`:

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




```
> head(preference_classification(model_berserk), n = 5)
```

```
                        1     2     3 est
a_chess_player_123  0.556 0.376 0.068   1
a_nizamoff          0.276 0.648 0.076   2
a_salikhov          0.828 0.172 0.000   1
a137p314            0.616 0.368 0.016   1
a1bharadwaj2019_64k 0.684 0.296 0.020   1
```

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 trips (cont.). Per default, the `predict()` method returns a confusion matrix, which gives an overview of the in-sample prediction performance: Warning Train p. 69.

```
> predict(model_train)
```

```
    predicted
true    A    B
   A 1034  440
   B  452 1003
```

By setting the argument `overview = FALSE`, the method instead returns predictions on the level of individual choice occasions:[13]

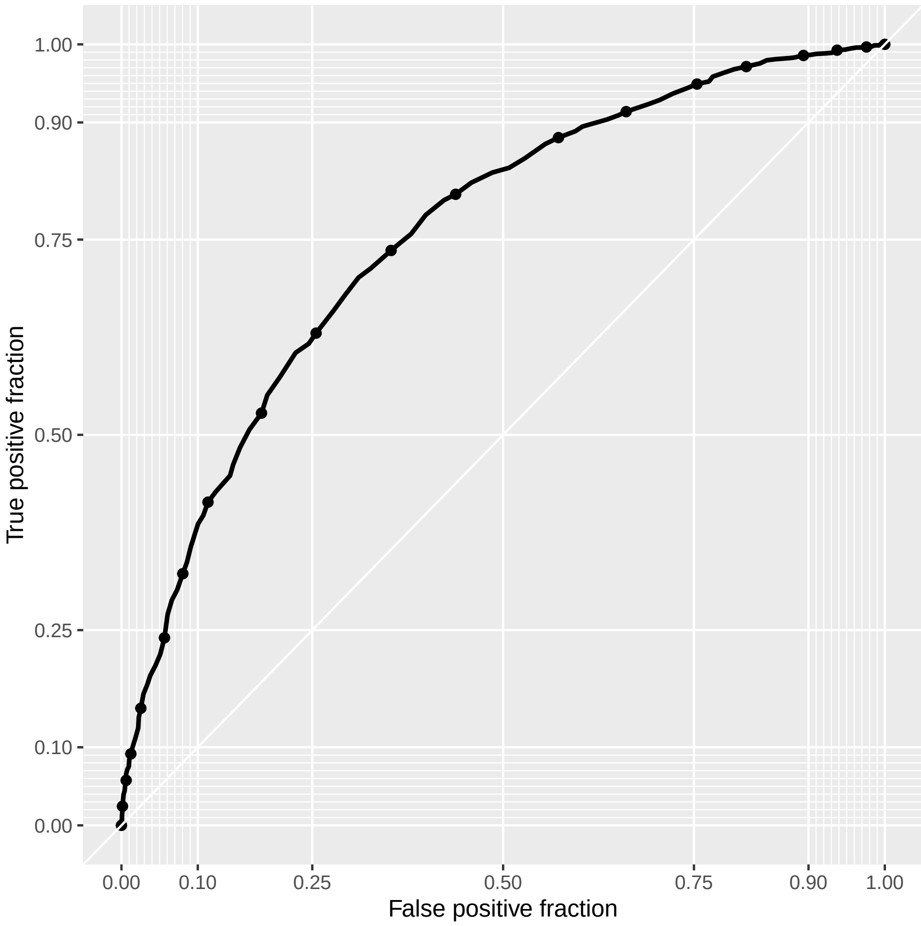
```
> pred <- predict(model_train, overview = FALSE)
> head(pred, n = 5)
```

```
  id choiceid    A    B true predicted correct
1  1        1 0.92 0.08    A         A    TRUE
2  1        2 0.64 0.36    A         A    TRUE
3  1        3 0.79 0.21    A         A    TRUE
4  1        4 0.18 0.82    B         B    TRUE
5  1        5 0.55 0.45    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)
```

[13]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 a test subsample extracted via 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 0.50 0.50          A
2  2        1 0.35 0.65          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 0.50 0.50          A
2  2        1 0.57 0.43          A
```

6. Model selection

RprobitB provides several tools to identify the most appropriate model among competing one, including the information criteria AIC (Akaike 1974), BIC (Schwarz 1978), WAIC (Watanabe and Opper 2010), and the Bayes factor.

The WAIC is a Bayesian version of AIC and BIC and defined as $-2 \cdot \text{lppd} + 2 \cdot p_{\text{WAIC}}$, where $\text{lppd} = \sum_i \log\left(S^{-1} \sum_s p_{si}\right)$ is the log-pointwise predictive density, $p_{\text{WAIC}} = \sum_i \mathbb{V}_\theta \log(p_{si})$ is a penalty term proportional to the variance in the posterior distribution, and $p_{si} = \Pr(y_i \mid \theta_s)$ be the probability of choice y_i given the s-th set θ_s of parameter samples from the posterior (McElreath 2020, p. 220). 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 total number of choices. Both WAIC value and its standard error can be computed via the `WAIC()` method.

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 M_1 and M_2, it is defined as

$$BF(M_1, M_2) = \frac{\Pr(M_1 \mid y)}{\Pr(M_2 \mid y)} = \frac{\Pr(y \mid M_1)}{\Pr(y \mid M_2)} \Big/ \frac{\Pr(M_1)}{\Pr(M_2)},$$

where per default $\Pr(M_1) = \Pr(M_2) = 0.5$. The value $\Pr(y \mid M)$ denotes the marginal model likelihood, which has no closed form and must be approximated numerically. **RprobitB** uses the posterior Gibbs samples derived from the `fit_model()` function to approximate the likelihood via the posterior harmonic mean estimator (Newton and Raftery 1994) in combination with the prior arithmetic mean estimator (Hammersley and Handscomb 1964). Both estimators converge with rising posterior samples to the marginal model likelihood by the law of large numbers. Convergence is fast if the prior and posterior distribution have a similar shape and strong overlap (Gronau *et al.* 2017). The estimators are implemented in the function `mml`. **RprobitB** provides plotting methods for analyzing the convergence behavior, see `help(mml, package = "RprobitB")` for details.

Example 1: Train trips (cont.). We revisit the probit model of travelers deciding between two fictional train route alternatives. As a competing model to `model_train`, we consider explaining the choices only by the alternative's price, i.e. the probit model with the formula `choice ~ price | 0`‘. The `nested_model()` function helps in estimating such a nested model:

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

RprobitB provides the convenience function `model_selection()`, which takes an arbitrary number of '`RprobitB_fit`' objects and returns a matrix of model selection criteria. The `criteria` input is a vector of `"npar"` (for the number of model parameters), `"LL"` (for the model's log-likelihood value, computed with the point estimates obtained from the Gibbs sample means), `"AIC"`, `"BIC"`, `"WAIC"`, `"MMLL"` (the marginal model log-likelihood), `"BF"` (for the Bayes factor), and `"pred_acc"` (the prediction accuracy). In order to compute WAIC, the marginal model likelihood, and the Bayes factor, the probabilities $p_{si} = \Pr(y_i \mid \theta_s)$ must be pre-computed via the `compute_p_si()` function:

```
> model_train <- compute_p_si(model_train)
> model_train_sparse <- compute_p_si(model_train_sparse)
> model_selection(
+    model_train, model_train_sparse,
+    criteria = c("npar", "LL", "AIC", "BIC", "WAIC", "MMLL", "BF", "pred_acc")
+  )
```

```
                          model_train model_train_sparse
npar                                4                  1
LL                           -1727.72           -1865.86
AIC                           3463.45            3733.73
BIC                           3487.38            3739.71
WAIC                          3462.95            3734.29
se(WAIC)                         0.16               0.08
pWAIC                            3.93               1.35
MMLL                         -1730.71           -1866.89
BF(*,model_train)                   1             < 0.01
BF(*,model_train_sparse)        > 100                  1
pred_acc                       69.55%             63.40%
```

7. Conclusion

The **RprobitB** package aims at making probit models accessible to R users with an interest in choice behavior heterogeneity. It contains functions to prepare and simulate choice data, to fit models, to update the class size, to use a fitted model for choice prediction, and to perform model selection. The **RprobitB** package has a user-friendly design: the different package objects can be seamlessly passed between functions and its usage follows a clear workflow (see Figure 1). In this paper, we demonstrated for examples that serves as a starting point for R users who want to apply latent class mixed probit models to their own choice data.

Current limitations of the **RprobitB** package include... We plan to overcome these limitations and invite the community to suggest further features that we can implement in future package versions.

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

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**.

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**.

Bierlaire M (2020). "A short introduction to PandasBiogeme." *A short introduction to PandasBiogeme.*

Brüderl J, Drobnič S, Hank K, Neyer FJ, Walper S, Alt P, Borschel E, Bozoyan C, Garrett M, Geissler S, Gonzalez Avilés T, Gröpler N, Hajek K, Herzig M, Huyer-May B, Lenke R, Lorenz R, Lutz K, Minkus L, Peter T, Phan T, Preetz R, Reim J, Sawatzki B, Schmiedeberg C, Schütze P, Schumann N, Thönnissen C, Timmermann K, Wetzel M (2020). "Beziehungs- und Familienpanel (pairfam)." *GESIS Datenarchiv, Köln. ZA5678 Datenfile Version 11.0.0.* doi:10.4232/pairfam.5678.11.0.0.

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.

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.

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.

Hess S, Palma D (2019). *Apollo: a flexible, powerful and customisable freeware package for choice model estimation and application*. Choice Modelling Centre. R package version 0.2.7, URL http://www.ApolloChoiceModelling.com.

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 D (2022). *MNP: Fitting the Multinomial Probit Model*. R package version 3.1-3, URL https://CRAN.R-project.org/package=MNP.

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

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.

Rossi P (2019). "bayesm: Bayesian Inference for Marketing/Micro-Econometrics." R package version 3.1-4, URL https://CRAN.R-project.org/package=bayesm.

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.

Sarrias M (2016). "Discrete Choice Models with Random Parameters in R: The Rchoice Package." *Journal of Statistical Software*, **74**(10), 1–31. doi:10.18637/jss.v074.i10.

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

Train K (2001). "A comparison of hierarchical Bayes and maximum simulated likelihood for mixed logit." *University of California, Berkeley*, pp. 1–13.

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

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).

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

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