Package 'ItemResponseTrees'

May 7, 2020

Title IR-Tree Modeling in 'mirt', 'Mplus', or 'TAM'

Version 0.2.5

Description Item response tree (IR-tree) models are a class of item response theory (IRT) models that assume that the responses to polytomous items can best be explained by multiple psychological processes; see Böckenholt (2012) <doi:10.1037/a0028111> for details. The package 'ItemResponseTrees' allows to fit such IR-tree models in 'mirt', 'Mplus', or 'TAM'. The package automates some of the hassle of IR-tree modeling by means of a consistent syntax. This allows new users to quickly adopt this model class, and this allows experienced users to fit many complex models effortlessly.

License MIT + file LICENSE

URL https://github.com/hplieninger/ItemResponseTrees

BugReports https://github.com/hplieninger/ItemResponseTrees/issues

Depends R (>= 3.6.0)

Imports checkmate (>= 1.9), dplyr, generics, glue, magrittr, MASS, Matrix, methods, mirt (>= 1.30), purrr, rlang (>= 0.4.0), sets, stringr, tibble, tidyr (>= 1.0.0), tidyselect

Suggests covr, future, knitr, listenv, lme4, modeltests (>= 0.1.0), MplusAutomation (>= 0.7), progress, rmarkdown, roxygen2, simsalapar, sfsmisc, spelling, TAM (>= 3.5-19), testthat (>= 2.1.0)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Hansjörg Plieninger [aut, cre] (<https://orcid.org/0000-0002-4416-300X>)

Maintainer Hansjörg Plieninger <mail@hansjoerg.me>

Repository CRAN

Date/Publication 2020-05-06 22:00:15 UTC

R topics documented:

`augment.irtree_fit` *Augment data with information from an irtree_fit object*

Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset, namely, predicted values in the .fitted column. New columns always begin with a . prefix to avoid overwriting columns in the original dataset.

Usage

```
## S3 method for class 'irtree_fit'
augment(x = NULL, data = NULL, se_fit = TRUE, method = "EAP", ...)
```

Arguments

`x`	object of class `irtree_fit` as returned from `fit()`.
`data`	Optional data frame that is returned together with the predicted values. Argument is not needed since the data are contained in the fitted object.
`se_fit`	Logical indicating whether standard errors for the fitted values should be returned as well.
`method`	This is passed to `mirt::fscores()` or `TAM:::IRT.factor.scores()` (as argument `type`) if applicable.

... Additional arguments passed to mirt::fscores() or TAM:::IRT.factor.scores() if applicable.

Details

Note that argument method is used only for engines mirt and TAM.

Value

Returns a tibble with one row for each observation and one (two) additional columns for each latent variable if se_fit = FALSE (if se_fit = TRUE). The names of the new columns start with .fit (and .se.fit).

See Also

generics::augment()

Examples

```
data("jackson")
df1 <- jackson[1:234, paste0("C", 1:5)]
irtree_create_template(df1)
m1 <- "
IRT:
t BY C1@1, C2@1, C3@1, C4@1, C5@1;
Class:
GRM"
fit1 <- fit(irtree_model(m1), data = df1)

tidy(fit1, par_type = "difficulty")

glance(fit1)

augment(fit1)
```

---

control_mirt *Control aspects of fitting a model in mirt*

Description

This function should be used to generate the control argument of the fit() function.

Usage

```
control_mirt(
  SE = TRUE,
  method = "EM",
  quadpts = NULL,
  control = list(),
```

```
  technical = list(),
  ...
)
```

Arguments

SE, method, quadpts, ...	These arguments are passed to and documented in mirt::mirt(). They can be used to tweak the estimation algorithm.
control	List of arguments passed to argument control of mirt::mirt().
technical	List of arguments passed to argument technical of mirt::mirt().

Value

A list with one element for every argument of control_mirt().

Examples

```
control_mirt(SE = FALSE,
             method = "QMCEM",
             quadpts = 4455,
             technical = list(NCYCLES = 567),
             TOL = .001)
control_mirt(method = "MHRM",
             draws = 5544)
```

---

control_mplus *Control aspects of fitting a model in Mplus*

Description

This function should be used to generate the control argument of the fit() function.

Usage

```
control_mplus(
  file = tempfile("irtree_"),
  overwrite = FALSE,
  cleanup = run,
  run = TRUE,
  estimator = "MLR",
  quadpts = 15,
  save_fscores = TRUE,
  analysis_list = list(COVERAGE = "0"),
  Mplus_command = "Mplus",
  warnings2messages = FALSE
)
```

Arguments

file	String naming the file (or path) of the Mplus files and the data file. Do not provide file endings, those will be automatically appended.
overwrite	Logical value indicating whether data and input (if present) files should be overwritten.
cleanup	Logical, whether the Mplus files should be removed on exit.
run	Logical, whether to indeed run Mplus.
estimator	String, passed to argument 'ESTIMATOR' in Mplus.
quadpts	This is passed to argument 'INTEGRATION' of Mplus. Thus, it may be an integer specifying the number of integration points for the Mplus default of rectangular numerical integration (e.g., `quadpts = 15`). Or it may be a string, which gives more fine grained control (e.g., `quadpts = "MONTECARLO(2000)"`).
save_fscores	Logical, whether to save FSCORES or not.
analysis_list	Named list of strings passed to Mplus' argument ANALYSIS. See examples below.
Mplus_command	optional. N.B.: No need to pass this parameter for most users (has intelligent defaults). Allows the user to specify the name/path of the Mplus executable to be used for running models. This covers situations where Mplus is not in the system's path, or where one wants to test different versions of the Mplus program.
warnings2messages	Logical, whether Mplus errors and warnings should be signaled as warnings (the default) or messages.

Value

A list with one element for every argument of `control_mplus()`.

Examples

```
control_mplus(file = tempfile("irtree_", tmpdir = "."),
              quadpts = "GAUSS(10)",
              analysis_list = list(COVERAGE = "0",
                                   MITERATIONS = "500",
                                   MCONVERGENCE = ".001"))
```

---

control_tam *Control aspects of fitting a model in TAM*

Description

This function should be used to generate the `control` argument of the `fit()` function.

Usage

```
control_tam(
  set_min_to_0 = FALSE,
  control = list(snodes = 0, maxiter = 1000, increment.factor = 1, fac.oldxsi = 0),
  ...
)
```

Arguments

set_min_to_0	Logical. TAM::tam.mml() expects the data to be scored 0, ..., K. If set_min_to_0 = TRUE, the minimum of the data is subtracted from each response, which will likely both satisfy TAM and do no harm to the data.
control	List of arguments passed to argument control of TAM::tam.mml(). See examples below.
...	Other arguments passed to TAM::tam.mml().

Value

A list with one element for every argument of control_tam().

Examples

```
control_tam(set_min_to_0 = TRUE,
            control = list(snodes = 0,
                           maxiter = 1000,
                           increment.factor = 1,
                           fac.oldxsi = 0),
            constraint = "items")
```

---

fit.irtree_model *Fit an ItemResponseTrees model*

Description

This function takes a data frame and an object of class irtree_model and runs the model in either mirt, Mplus, or TAM.

Usage

```
## S3 method for class 'irtree_model'
fit(
  object = NULL,
  data = NULL,
  engine = c("mirt", "mplus", "tam"),
  ...,
  link = c("logit", "probit"),
  verbose = interactive(),
```

```
  control = NULL,
  improper_okay = FALSE
)
```

Arguments

object	Object of class irtree_model. See irtree_model for more information.
data	Data frame containing containing one row per respondent and one column per variable. The variable names must correspond to those used in object.
engine	String specifying whether to use mirt, Mplus, or TAM for estimation.
...	Not currently used. Use control instead.
link	String specifying the link function. May be either logit, or (in case of Mplus), probit.
verbose	Logical indicating whether output should be printed to the console.
control	List. The allowed elements of this list depend on the engine. Use control_mirt(), control_mplus(), or control_tam() for convenience. Note that the fit() function does not use . . . , but that you can use the control_*() functions to pass additional arguments.
improper_okay	Logical indicating whether the model should also be fit if it is not a proper IR-tree model. Set this only to TRUE if you really know what you are doing.

Value

Returns a list of class irtree_fit. The first list element is the return value of either mirt::mirt(), MplusAutomation::readModels(), or TAM::tam.mml(). Further information is provided in the element spec.

Methods

The methods coef(), summary(), and print() are implemented for objects of class irtree_fit, and those wrap the respective functions of mirt, MplusAutomation, or TAM. However, glance(), tidy(), and augment() may be more helpful.

Examples

```
# Running these examples may take a while

data("jackson")
df1 <- jackson[1:456, paste0("C", 1:5)]
df2 <- jackson[1:456, c(paste0("C", 1:5), paste0("E", 1:5))]

irtree_create_template(df1)

# Graded Response Model ------------------------------------------------

m1 <- "
IRT:
```

```
t BY C1@1, C2@1, C3@1, C4@1, C5@1;

Class:
GRM
"

model1 <- irtree_model(m1)

fit1 <- fit(model1, data = df1)

glance(fit1)
tidy(fit1, par_type = "difficulty")
augment(fit1)

# IR-Tree Models ----------------------------------------------------------

##### IR-tree model for 1 target trait #####

m2 <- "
Equations:
1 = (1-m)*(1-t)*e
2 = (1-m)*(1-t)*(1-e)
3 = m
4 = (1-m)*t*(1-e)
5 = (1-m)*t*e

IRT:
t BY C1@1, C2@1, C3@1, C4@1, C5@1;
e BY C1@1, C2@1, C3@1, C4@1, C5@1;
m BY C1@1, C2@1, C3@1, C4@1, C5@1;

Class:
Tree
"

model2 <- irtree_model(m2)

# See ?mirt::mirt for details on method argument
fit2 <- fit(model2, data = df1, control = control_mirt(method = "MHRM"))

##### IR-tree model for 2 target traits #####

m3 <- "
Equations:
1 = (1-m)*(1-t)*e
2 = (1-m)*(1-t)*(1-e)
3 = m
4 = (1-m)*t*(1-e)
5 = (1-m)*t*e

IRT:
t1 BY C1@1, C2@1, C3@1, C4@1, C5@1;
t2 BY E1@1, E2@1, E3@1, E4@1, E5@1;
```

```
e  BY C1@1, C2@1, C3@1, C4@1, C5@1, E1@1, E2@1, E3@1, E4@1, E5@1;
m  BY C1@1, C2@1, C3@1, C4@1, C5@1, E1@1, E2@1, E3@1, E4@1, E5@1;

Class:
Tree

Constraints:
t = t1 | t2
"

model3 <- irtree_model(m3)

fit3 <- fit(model3, data = df2, control = control_mirt(method = "MHRM"))

##### IR-tree model constrained to Steps Model #####

m4 <- "
Equations:
1 = (1-a1)
2 = a1*(1-a2)
3 = a1*a2*(1-a3)
4 = a1*a2*a3*(1-a4)
5 = a1*a2*a3*a4

IRT:
a1 BY C1@1, C2@1, C3@1, C4@1, C5@1;
a2 BY C1@1, C2@1, C3@1, C4@1, C5@1;
a3 BY C1@1, C2@1, C3@1, C4@1, C5@1;
a4 BY C1@1, C2@1, C3@1, C4@1, C5@1;

Class:
Tree

Constraints:
a1 = a2
a1 = a3
a1 = a4
"

model4 <- irtree_model(m4)

fit4 <- fit(model4, data = df1)

# Partial Credit Model ----------------------------------------------------

##### Ordinary PCM #####

m5 <- "
IRT:
t BY C1@1, C2@1, C3@1, C4@1, C5@1;

Weights:
t = c(0, 1, 2, 3, 4)
```

```
Class:
PCM
"

model5 <- irtree_model(m5)

fit5 <- fit(model5, data = df1 - 1, engine = "tam")

##### Multidimensional PCM with constraints #####

m6 <- "
IRT:
t1 BY C1@1, C2@1, C3@1, C4@1, C5@1;
t2 BY E1@1, E2@1, E3@1, E4@1, E5@1;
e  BY C1@1, C2@1, C3@1, C4@1, C5@1, E1@1, E2@1, E3@1, E4@1, E5@1;
m  BY C1@1, C2@1, C3@1, C4@1, C5@1, E1@1, E2@1, E3@1, E4@1, E5@1;

Weights:
t = c(0, 1, 2, 3, 4)
e = c(1, 0, 0, 0, 1)
m = c(0, 0, 1, 0, 0)

Class:
PCM

Constraints:
t = t1 | t2
"

model6 <- irtree_model(m6)

fit6 <- fit(model6, data = df2 - 1, engine = "tam",
            control = control_tam(control = list(snodes = 1234)))
```

`glance.irtree_fit` *Glance at an irtree_fit object*

Description

Glance accepts an `irtree_fit` object and returns a tibble with exactly one row of model summaries.

Usage

```
## S3 method for class 'irtree_fit'
glance(x = NULL, ...)
```

Arguments

x	object of class irtree_fit as returned from fit().
...	Additional arguments. Not used.

Value

A one-row tibble with columns such as AIC and BIC.

Converged:

The column converged indicates whether the model converged or not. For Mplus, this is TRUE if the output contained the phrase "The model estimation terminated normally". For mirt, this is equal to the output of mirt::extract.mirt(x,"converged"). For TAM, this is NA if no clear signs of non-convergence were observed. You are encouraged to check any warnings or errors in any case.

Iterations:

iterations is NA for Mplus models since respective information is not easily obtained from the output.

See Also

generics::glance(), mirt::extract.mirt(x,"secondordertest")

Examples

```
data("jackson")
df1 <- jackson[1:234, paste0("C", 1:5)]
irtree_create_template(df1)
m1 <- "
IRT:
t BY C1@1, C2@1, C3@1, C4@1, C5@1;
Class:
GRM"
fit1 <- fit(irtree_model(m1), data = df1)

tidy(fit1, par_type = "difficulty")

glance(fit1)

augment(fit1)
```

---

irtree_create_template

Create a template of a model string

Description

This function prints a template of a model string to the console based on the supplied data frame. This template can be copy-pasted and modified to define an irtree_model.

Usage

```
irtree_create_template(data = NULL, mapping_matrix = NULL, rasch = TRUE)
```

Arguments

data	Data frame.
mapping_matrix	Matrix of so-called pseudo-items, optional. The observed response categories must appear in the first column. The other columns contain the pseudo-items and each entry may be either 1, 0, or NA.
rasch	Logical. The string @1 will be appended to each variable name if TRUE with no effect otherwise.

Examples

```
irtree_create_template(jackson[, c(1, 6, 11)])
#> m1 <- "
#> Equations:
#> 1 = ...
#> 2 = ...
#> 3 = ...
#> 4 = ...
#> 5 = ...
#>
#> IRT:
#> ... BY E1@1, E2@1, E3@1;
#> ... BY E1@1, E2@1, E3@1;
#>
#> Class:
#> Tree
#> "

irtree_create_template(jackson[, c(1, 6, 11)],
                       cbind(1:5,
                             m = c(0, 0, 1, 0, 0),
                             t = c(1, 1, NA, 0, 0),
                             e = c(1, 0, NA, 0, 1)))
#> m1 <- "
#> Equations:
#> 1 = (1-m)*t*e
#> 2 = (1-m)*t*(1-e)
#> 3 = m
#> 4 = (1-m)*(1-t)*(1-e)
#> 5 = (1-m)*(1-t)*e
#>
#> IRT:
#> m BY E1@1, E2@1, E3@1;
#> t BY E1@1, E2@1, E3@1;
#> e BY E1@1, E2@1, E3@1;
#>
#> Class:
#> Tree
```

```
#> "
```

`irtree_gen_data` *Generate data*

Description

This function generates data from an ItemResponseTrees model.

Usage

```
irtree_gen_data(
  object = NULL,
  N = NULL,
  sigma = NULL,
  theta = NULL,
  itempar = NULL,
  link = c("logit", "probit"),
  na_okay = TRUE,
  skip = FALSE
)
```

Arguments

`object`	Object of class `irtree_model`. See irtree_model for more information.
`N`	Integer, the number of persons.
`sigma`	Either a matrix or a function that returns a matrix. This matrix is the variance-covariance matrix of the person parameters that is passed to `MASS::mvrnorm()`. Note that the order of the person parameters is taken from the section Processes in the model `object` (see irtree_model).
`theta`	Optional numeric matrix of person parameters with one row per person and one column per dimension (i.e., `object$S`). If provided, this overrides `N` and `sigma`.
`itempar`	Either a list or a function that returns a list. The list has an element `beta` and an element `alpha`. Each of these is a matrix of item parameters. Note that the order of items (rows) is taken from the section Items and the order of processes (columns) is taken from the section Processes in the `model` (see irtree_model).
`link`	Character. Link function to use.
`na_okay`	Logical indicating whether variables with unobserved response categories are permitted. If `FALSE`, rejection sampling is used to ensure that all categories are observed.
`skip`	Logical. Some features of the irtree_model syntax, which are available for model fitting (e.g., `Addendum`), are not implemented for data generation. Those parts of the model are ignored if `skip = TRUE`.

Value

A list with element `data` containing the data and an element `spec` containing the true parameter values etc.

Examples

```
# IR-Tree Model ------------------------------------------------------------

m1 <- "
Equations:
1 = (1-m)*(1-t)*e
2 = (1-m)*(1-t)*(1-e)
3 = m
4 = (1-m)*t*(1-e)
5 = (1-m)*t*e

IRT:
t BY x1, x2, x3;
e BY x1, x2, x3;
m BY x1, x2, x3;

Class:
Tree
"

model1 <- irtree_model(m1)

dat1 <- irtree_gen_data(model1, N = 5, sigma = diag(3),
                        itempar = list(beta = matrix(rnorm(9), 3, 3),
                                       alpha = matrix(1, 3, 3)))
dat1$data

# Partial Credit Model -----------------------------------------------------

m2 <- "
IRT:
t BY x1@1, x2@1, x3@1;
e BY x1@1, x2@1, x3@1;
m BY x1@1, x2@1, x3@1;

Weights:
t = c(0, 1, 2, 3, 4)
e = c(1, 0, 0, 0, 1)
m = c(0, 0, 1, 0, 0)

Class:
PCM
"
model2 <- irtree_model(m2)
dat2 <- irtree_gen_data(model2, N = 5, sigma = diag(3),
                        itempar = list(beta = matrix(sort(rnorm(12)), 3, 4)))
dat2$data
```

```
m3 <- "
IRT:
t BY x1@1, x2@1, x3@1;

Weights:
t = 0:4

Class:
PCM
"

model3 <- irtree_model(m3)

dat3 <- irtree_gen_data(model3, N = 5, sigma = diag(1),
                        itempar = list(beta = matrix(sort(rnorm(12)), 3, 4)))
dat3$data
```

`irtree_model` *ItemResponseTrees model syntax*

Description

The ItemResponseTrees model syntax describes the statistical model. The function `irtree_model()` turns a user-defined model string into an object of class `irtree_model` that represents the full model as needed by the package.

Usage

```
irtree_model(model = NULL)
```

Arguments

`model`	String with a specific structure as described below.

Value

List of class `irtree_model`. It contains the information extracted from `model`. Side note: The returned list contains an element `mappping_matrix` that contains the pseudoitems. This information is instructive, and it might be used as an input to the `dendrify()` function of the irtrees package.

Overview of the Model Syntax

1. The `model` string must contain at least the sections **IRT**, **Class**, and (if class is tree) **Equations**.
2. Section headings must appear on a separate line ending with a colon (:).
3. The model may contain empty lines and commented lines, which begin with # (do not use inline comments).

4. Line breaks are only allowed in section **IRT**.

Details for all the required and optional sections of the `model` string are given in the following.

Equations:

The `model` must contain a section with heading **Equations** if **Class** is Tree. Therein, the model equations are described. They have a structure similar to `Cat = p1*(1-p2)`, where `Cat` is any observed response category in the data set, and where `p1` is a freely chosen name of a parameter. These names must match the names of the latent variables in the section **IRT** (combined with **Constraints** if specified).

If you prefer to work with pseudo-items, `irtree_create_template()` can generate the equations for you.

The equations may contain only products and not sums. That is, it is not possible to estimate genuine mixture models as, for example, in the package mpt2irt.

Each equation must appear on a separate, non-broken line. For example:

```
Equations:
1 = (1-m)*(1-t)*e
2 = (1-m)*(1-t)*(1-e)
3 = m
4 = (1-m)*t*(1-e)
5 = (1-m)*t*e
```

IRT:

The `model` must contain a section with heading **IRT**. Therein, the IRT structure of the model is described in a way resembling the MODEL part of an Mplus input file. It has a structure of `LV BY item1*,item2@1`, where `LV` is the name of the latent variable/parameter/process, and `item` is the name of the observed variable in the data set, which is followed by the loading. The loading may either be fixed (e.g., to 1) using `@1` or it may be set free using `*` (which is equivalent to omitting a loading altogether).

Each measurement model (i.e., the LV and its items) must appear on a separate line ending with a semicolon. Items must be separated by commas. Line breaks are allowed. For example:

```
IRT:
t BY x1, x2, x3, x4, x5, x6;
e BY x1@1, x2@1, x3@1, x4@1, x5@1, x6@1;
m BY x1@1, x2@1, x3@1, x4@1, x5@1, x6@1;
```

Class:

The `model` must contain a section with heading **Class** to specify the type/class of IRT model to use. Currently, may be either `Tree`, `GRM`, or `PCM`. For example:

```
Class:
Tree
```

Constraints:

The `model` may contain a section with heading **Constraints** to specify equality constraints of latent variables. Constraints may be useful for multidimensional questionnaires to link **IRT** and **Equations** in a specific way. Or, latent variables in **IRT** may be constrained to equality.

Constraints in order to link sections IRT and Equations:
A process in the model equations (section **Equations**) may correspond to multiple latent variables (section **IRT**). For example, when analyzing a Big Five data set, one may wish to specify only one extremity process *e* for all items but multiple target traits *t*, namely, one for each of the five scales. In such a case, the section **Equations** would list only the parameter *t*, while the section **IRT** would list the parameters *t1*, ..., *t5*.
In the framework of MPT, one would think of such a situation in terms of multiple albeit similar trees with specific parameter contraints across trees. For example, one would use one tree for each Big Five scale and fix the response style parameters to equality across trees but not the target trait parameters.
Each line in this section has a structure of `Param = LV1 | LV2`, where `Param` is the name of the process used only in section **Equations** and `LV1` it the name of the process used only in section **IRT**. Use one line for each definition. For example:

```
Constraints:
t = t1 | t2 | t3 | t4 | t5
```

Constraints within section IRT:
For example, in a sequential model as proposed by Tutz as well as Verhelst, one would specify two processes for a 3-point item. The first process would correspond to a pseudoitem of `0-1-1` and the second process to a pseudoitem of `NA-0-1`. However, the latent variables corresponding to these two processes would typically be assumed to be equal and need thus be constrained accordingly.
Each line in this section has a structure of `LV1 = LV2`, where `LV1` and `LV2` are the names of the latent variables used in section **IRT**. Use one line for each definition. For example:

```
Constraints:
LV1 = LV2
LV1 = LV3
```

Addendum:

The `model` may contain a section with heading **Addendum** if `engine = "mplus"` is used for estimation. Any code in this section is directly pasted in the `MODEL` section of the Mplus input file. Use a semicolon at the end of each line; lines must not exceed 90 characters. Note that the addendum is ignored in `irtree_gen_data()`. For example:

```
Addendum:
e WITH t@0;
m WITH t@0;
```

Weights:

The `model` may contain a section with heading **Weights** if model **Class** is PCM. This allows to specify (uni- and) multidimensional partial credit models. They have been proposed, for example, by Wetzel and Carstensen (2017), as an alternative to IR-tree models. Note that fitting these models is only implemented for `engine = "tam"`.

Each line in this section has a structure of `LV = weights`, where `LV` is the name of the latent variable used in section **IRT**. `weights` must be valid R code, namely, a vector of weights (see, e.g., Table 1 in Wetzel & Carstensen, 2017, or Table 2 in Falk & Cai, 2015). Use one line for each definition. For example:

```
Weights:
t = c(0, 1, 2, 3, 4)
```

```
e = c(1, 0, 0, 0, 1)
m = c(0, 0, 1, 0, 0)
```

Examples

```
m1 <- "
# Random comment

Equations:
1 = (1-m)*(1-t)*e
2 = (1-m)*(1-t)*(1-e)
3 = m
4 = (1-m)*t*(1-e)
5 = (1-m)*t*e

IRT:
t1 BY x1@1, x2*, x3*;
t2 BY x4@1, x5*, x6*;
e  BY x1@1, x2@1, x3@1, x4@1, x5@1, x6@1;
m  BY x1@1, x2@1, x3@1, x4@1, x5@1, x6@1;

Constraints:
t = t1 | t2

Class:
Tree
"

model <- irtree_model(m1)
```

`irtree_recode` *Recode data into pseudoitems*

Description

This function takes a data set with polytomous items and an `irtree_model` and returns the recoded items, the so-called pseudoitems.

Usage

```
irtree_recode(object = NULL, data = NULL, keep = FALSE)
```

Arguments

`object`	Object of class `irtree_model`. See irtree_model for more information.
`data`	Data frame containing containing one row per respondent and one column per variable. The variable names must correspond to those used in `object`.
`keep`	Logical indicating whether to append the original items to the data frame of the generated pseudoitems

Value

Data frame

Examples

```
m1 <- "
IRT:
t BY x1;
e BY x1;
m BY x1;
Equations:
1 = (1-m)*(1-t)*e
2 = (1-m)*(1-t)*(1-e)
3 = m
4 = (1-m)*t*(1-e)
5 = (1-m)*t*e
Class:
Tree
"
model1 <- irtree_model(m1)
dat <- data.frame(x1 = 1:5)
irtree_recode(model1, dat, keep = TRUE)
```

`irtree_sim` *Run a simulation by generating from and fitting an ItemResponseTrees model*

Description

The function `irtree_sim()` generates data from an irtree_model and fits one or more models to these data. This process is repeated `R` times, and the argument `plan` allows to run the simulation in parallel.

Usage

```
irtree_sim(
  R = 1,
  gen_model = NULL,
  fit_model = gen_model,
  N = NULL,
  sigma = NULL,
  itempar = NULL,
  link = c("logit", "probit"),
  na_okay = TRUE,
  engine = c("mirt", "mplus", "tam"),
  verbose = FALSE,
  control = NULL,
  improper_okay = FALSE,
```

```
  par_type = "difficulty",
  plan = NULL,
  plan_args = list(),
  file = NULL,
  dir = tempdir(),
  save_rdata = TRUE,
  in_memory = c("reduced", "everything", "nothing")
)
```

Arguments

R	Number of replications. Can be either a single number indicating the number of replications (e.g., R = 100), or can be a range (e.g., R = 1:100).
gen_model	Object of class irtree_model describing the data-generating model. See irtree_model for more information.
fit_model	Object of class irtree_model describing the model that should be fit to the data. May be a list of multiple objects of class irtree_model if different models should be fit to the same data set. See irtree_model for more information.
N	Integer, the number of persons.
sigma	Either a matrix or a function that returns a matrix. This matrix is the variance-covariance matrix of the person parameters that is passed to MASS::mvrnorm(). Note that the order of the person parameters is taken from the section Processes in the model object (see irtree_model).
itempar	Either a list or a function that returns a list. The list has an element beta and an element alpha. Each of these is a matrix of item parameters. Note that the order of items (rows) is taken from the section Items and the order of processes (columns) is taken from the section Processes in the model (see irtree_model).
link	Character. Link function to use.
na_okay	Logical indicating whether variables with unobserved response categories are permitted. If FALSE, rejection sampling is used to ensure that all categories are observed.
engine	String specifying whether to use mirt, Mplus, or TAM for estimation.
verbose	Logical indicating whether output should be printed to the console.
control	List. The allowed elements of this list depend on the engine. Use control_mirt(), control_mplus(), or control_tam() for convenience. Note that the fit() function does not use ..., but that you can use the control_*() functions to pass additional arguments.
improper_okay	Logical indicating whether the model should also be fit if it is not a proper IR-tree model. Set this only to TRUE if you really know what you are doing.
par_type	Only used if the fit engine was mirt. Item parameters (or thresholds) can be either of type easiness (the mirt default) or difficulty (as in Mplus and TAM).
plan	Parameter passed as argument strategy to future::plan(). May be set to, for example, multiprocess in order to run the simulations in parallel.
plan_args	Named list. Parameters passed future::plan().

file	String giving the file path used to save the output if save_rdata = TRUE. Note that the file ending is automatically set to .rda. This argument is also passed to irtree_fit_mplus() if applicable.
dir	Path name that is used to save the results of every run if save_rdata = TRUE.
save_rdata	Logical indicating whether to save the results to an RData file.
in_memory	Character string indicating what output should be kept in memory (note the argument save_rdata, which is not affected by in_memory). If "reduced", the output of fit() is discarded and only summary information is retained. The alternative is to keep "everything" in memory, or to keep "nothing" in memory (which makes only sense in combination with save_rdata = TRUE).

Value

Returns a list of length R. For each replication, a list is returned with two elements. The element spec contains various specifications (such as the data). The element fits is a list with one element for each fit_model that contains the output of fit() as well as the elements glanced, tidied, and augmented (see glance(), tidy(), and augment()). Thus, res$sim3$fits$m2$glanced gives model-fit information such as AIC for the second model in the third replication, and res$sim3$spec$data contains the corresponding data set.

If in_memory = "nothing", returns NULL.

See Also

The data are generated via irtree_gen_data(), and the models are fit via fit().

Examples

```
# Running these examples may take a while

m1 <- "
Equations:
1 = 1-a
2 = a*(1-b)
3 = a*b

IRT:
a BY x1@1, x2@1, x3@1, x4@1, X5@1, X6@1, X7@1;
b BY x1@1, x2@1, x3@1, x4@1, X5@1, X6@1, X7@1;

Class:
Tree
"

m2 <- "
IRT:
a BY x1@1, x2@1, x3@1, x4@1, X5@1, X6@1, X7@1;

Class:
GRM
```

```
"

model1 <- irtree_model(m1)
model2 <- irtree_model(m2)

res <- irtree_sim(
    ### Data generation ###
    gen_model = model1,
    link = "logit",
    N = 500,
    sigma = function(x) diag(2),
    itempar = function(x) list(
        beta = matrix(sort(runif(model1$J*model1$P, -2, 2)),
                      model1$J, model1$P),
        alpha = matrix(1, model1$J, model1$P)),
    na_okay = FALSE,

    ### Estimation ###
    fit_model = list(model1, model2),
    engine = "mirt",
    control = control_mirt(SE = FALSE),
    par_type = "difficulty",

    ### Replications ###
    R = 2,
    save_rdata = FALSE,

    ### Optional parallelization ###
    plan = "multiprocess",
    plan_args = list(workers = future::availableCores() - 1)
)

tab1 <- matrix(NA, 0, 4, dimnames = list(NULL, c("Rep", "Model", "AIC", "BIC")))

for (ii in seq_along(res)) {
    for (jj in seq_along(res[[ii]]$fits)) {
        IC <- res[[ii]]$fits[[jj]]$glanced
        tab1 <- rbind(tab1, c(ii, jj, round(IC$AIC, -1), round(IC$BIC, -1)))
    }
}
tab1
#>      Rep Model  AIC  BIC
#> [1,]   1     1 6900 6970
#> [2,]   1     2 7000 7060
#> [3,]   2     1 6810 6880
#> [4,]   2     2 6880 6940
```

---

jackson *IPIP Big Five personality test answers (data set)*

Description

"This is data from an online big five personality test: http://personality-testing.info/tests/BIG5.php. The following items were rated on a likert scale from 1=disagree to 5=agree in relation to how much they applied to the test taker, they were presented to the taker 5 per page" (Jackson, 2012).

The following items are reverse keyed and were already recoded: `A1`, `E2`, `C2`, `N2`, `O2`, `A3`, `E4`, `C4`, `N4`, `O4`, `A5`, `E6`, `C6`, `O6`, `A7`, `E8`, `C8`, and `E10`.

Usage

```
data("jackson")
```

Format

A data frame with 9051 rows and 58 variables:

E1 Am the life of the party.

A1 Feel little concern for others.

C1 Am always prepared.

N1 Get stressed out easily.

O1 Have a rich vocabulary.

E2 Don't talk a lot.

A2 Am interested in people.

C2 Leave my belongings around.

N2 Am relaxed most of the time.

O2 Have difficulty understanding abstract ideas.

E3 Feel comfortable around people.

A3 Insult people.

C3 Pay attention to details.

N3 Worry about things.

O3 Have a vivid imagination.

E4 Keep in the background.

A4 Sympathize with others' feelings.

C4 Make a mess of things.

N4 Seldom feel blue.

O4 Am not interested in abstract ideas.

E5 Start conversations.

A5 Am not interested in other people's problems.

C5 Get chores done right away.

N5 Am easily disturbed.

O5 Have excellent ideas.

E6 Have little to say.

A6 Have a soft heart.

C6 Often forget to put things back in their proper place.

N6 Get upset easily.

O6 Do not have a good imagination.

E7 Talk to a lot of different people at parties.

A7 Am not really interested in others.

C7 Like order

N7 Change my mood a lot.

O7 Am quick to understand things.

E8 Don't like to draw attention to myself.

A8 Take time out for others.

C8 Shirk my duties.

N8 Have frequent mood swings.

O8 Use difficult words.

E9 Don't mind being the center of attention.

A9 Feel others' emotions.

C9 Follow a schedule.

N9 Get irritated easily.

O9 Spend time reflecting on things.

E10 Am quiet around strangers.

A10 Make people feel at ease.

C10 Am exacting in my work.

N10 Often feel blue.

O10 Am full of ideas.

gender Gender, either female, male, or other

age Age

SecondsElapsed Seconds elapsed

E Scale mean for extraversion

C Scale mean for conscientiousness

N Scale mean for neuroticism

O Scale mean for openness

A Scale mean for agreeableness

Details

The data set included here is Version 3 from 15.10.2012. It was released by Andrew Jackson under the CC BY 4.0 license.

Source

Jackson, A. (2012). IPIP Big Five personality test answers. https://doi.org/10.6084/m9.figshare.96542.v3

pseudoitems *Pseudo-Items*

Description

IR-tree models can be fit on the basis of so-called pseudo-items. To this end, the original, polytomous items are recoded into binary pseudo-items. Whether a pseudo-item is coded as `1`, `0`, or `NA` depends on the model equations (e.g., Böckenholt, 2012; Plieninger, 2020).

The ItemResponseTrees package internally works with pseudo-items as well. However, the user has to specify the model equations rather than the pseudo-items in the irtree_model syntax. Internally, the original responses are recoded on the basis of the model supplied by the user by the function `irtree_recode()`. This function may also be used directly if desired.

As an alternative to specifying the model equations themselves, users may also use the function `irtree_create_template()` with a mapping matrix (that specifies the structure of the pseudo-items) to generate the model equations automatically.

References

Böckenholt, U. (2012). Modeling multiple response processes in judgment and choice. *Psychological Methods*, *17*(4), 665–678. https://doi.org/10.1037/a0028111

Plieninger, H. (2020). Developing and applying IR-tree models: Guidelines, caveats, and an extension to multiple groups. *Organizational Research Methods*. Advance online publication. https://doi.org/10.1177/1094428120911096

Examples

```
# Mapping matrix for data with three response categories:
(mm <- cbind(cat = 0:2,
             p1 = c(0,  1, 1),
             p2 = c(NA, 0, 1)))
#>      cat p1 p2
#> [1,]   0  0 NA
#> [2,]   1  1  0
#> [3,]   2  1  1

irtree_create_template(data.frame(x1 = 0:2, x2 = 0:2),
                       mapping_matrix = mm)
#>
#> m1 <- "
#> Equations:
#> 0 = (1-p1)
#> 1 = p1*(1-p2)
#> 2 = p1*p2
#>
#> IRT:
#> p1 BY x1@1, x2@1;
#> p2 BY x1@1, x2@1;
#>
```

```
#> Class:
#> Tree
#> "
#>
```

`tidy.irtree_fit` *Tidy an irtree_fit object*

Description

Tidy summarizes information about the parameter estimates of an ItemResponseTrees model.

Usage

```
## S3 method for class 'irtree_fit'
tidy(x = NULL, par_type = NULL, ...)
```

Arguments

`x`	object of class `irtree_fit` as returned from `fit()`.
`par_type`	Only used if the fit engine was mirt. Item parameters (or thresholds) can be either of type `easiness` (the mirt default) or `difficulty` (as in Mplus and TAM).
`...`	Not currently used.

Value

A tibble with one row for each model parameter and the following columns:

`term` The name of the model parameter.

`estimate` The estimated value of the term.

`std.error` The standard error of the term.

`statistic` The value of the test statistic of the term (Mplus only).

`p.value` The p-value associated with the statistic (Mplus only).

See Also

`generics::tidy()`

Examples

```
data("jackson")
df1 <- jackson[1:234, paste0("C", 1:5)]
irtree_create_template(df1)
m1 <- "
IRT:
t BY C1@1, C2@1, C3@1, C4@1, C5@1;
Class:
GRM"
fit1 <- fit(irtree_model(m1), data = df1)

tidy(fit1, par_type = "difficulty")

glance(fit1)

augment(fit1)
```

Index