Package 'dexter'

December 16, 2017

Type Package

Title Data Management and Analysis of Tests

Version 0.6.0

Author Gunter Maris, Timo Bechger, Jesse Koops, Ivailo Partchev

Maintainer Ivailo Partchev <Ivailo.Partchev@cito.nl>

Description A system for the management, assessment, and psychometric analysis of data from educational and psychological tests. Developed at Cito, The Netherlands, with subsidy from the Dutch Ministry of Education, Culture, and Science.

License GPL-3

Encoding UTF-8

LazyLoad yes

LazyData yes

Depends R (>= 3.3), RSQLite (>= 1.1.2)

Imports DBI, graphics, methods, utils, tidyr, tibble, colorspace, shiny, shinyBS, shinydashboard, DT, fastmatch, purrr, rlang, dplyr (>= 0.7.0), dbplyr (>= 1.1.0), rprintf, RColorBrewer

RoxygenNote 6.0.1

Suggests knitr, rmarkdown, latticeExtra, testthat

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-12-16 16:48:26 UTC

R topics documented:

`ability` *Estimate abilities*

Description

Computes estimates of ability for persons or booklets

Usage

```
ability(dataSrc, parms, predicate = NULL, method = c("MLE", "EAP"),
  prior = c("normal", "Jeffreys"), use_draw = NULL, npv = 500, mu = 0,
  sigma = 4, standard_errors = FALSE, person_level = TRUE)

ability_tables(parms, design = NULL, method = c("MLE", "EAP"),
  prior = c("normal", "Jeffreys"), use_draw = NULL, npv = 500, mu = 0,
  sigma = 4, standard_errors = TRUE)
```

Arguments

`dataSrc`	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
`parms`	An object returned by `fit_enorm` and containing parameter estimates
`predicate`	An optional expression to subset data, if NULL all data is used
`method`	Maximum Likelihood (MLE) or Expected A posteriori (EAP)
`prior`	If an EAP estimate is produced one can choose a normal prior or Jeffreys prior; i.e., a prior proportional to the square root of test information.
`use_draw`	When parms is Bayesian (this is recognised automatically), use_draw is the index of the posterior sample of the item parameters that will be used to generating plausible values. If use_draw=NULL, a posterior mean is used. If outside range, the last iteration will be used.
`npv`	Number of plausible values sampled to calculate EAP with normal prior
`mu`	Mean of the normal prior
`sigma`	Standard deviation of the normal prior
`standard_errors`	If true standard-errors are produced.
`person_level`	deprecated, always TRUE.
`design`	A data.frame with columns item_id and optionally booklet_id. If design is NULL the score transformation table will be computed based on the test design that was used to calibrate the items. If the column booklet_id is not included, the score transformation table will be based on all items found in the design.

Details

MLE estimates of ability will produce an NA for the minimum (=0) or the maximum score on a booklet. If this is undesirable, we advise to use EAP with Jeffreys prior.

Value

ability a data.frame with columns: booklet_id, person_id, sumScore, theta and optionally se (standard error)

ability_tables a data.frame with columns: booklet_id, sumScore, theta and optionally se (standard error)

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db")
add_booklet(db, verbAggrData, "agg")
f = fit_enorm(db)
aa = ability_tables(f,method="MLE",standard_errors=FALSE)
bb = ability_tables(f,method="EAP",standard_errors=FALSE)
cc = ability_tables(f,method="EAP",prior="Jeffreys", standard_errors=FALSE)
plot(bb$sumScore, bb$theta, xlab="test-score",
  ylab="ability est.", pch=19, cex=0.7)
points(aa$sumScore, aa$theta, col="red", pch=19, cex=0.7)
points(aa$sumScore, cc$theta, col="green", pch=19, cex=0.7)
legend("topleft", legend = c("EAP normal prior",
  "EAP Jeffreys prior", "MLE"), bty = "n",
 lwd = 1, cex = 0.7, col = c("black", "green", "red"),
 lty=c(0,0,0), pch = c(19,19,19))

close_project(db)

## End(Not run)
```

`add_booklet` *Add a booklet to a project*

Description

Adds item response data for a test form (a.k.a. booklet)

Usage

```
add_booklet(db, x, booklet_id, auto_add_unknown_rules = TRUE)
```

Arguments

db	A handle to the database, i.e. the output of `start_new_project` or `open_project`
x	A data frame containing the responses and, optionally, person covariates. The data.frame should have one row per respondent and the column names should correspond to the item_id's in the rules or the names of the covariates. See details.

`booklet_id` A (short) string identifying the test form (booklet)

`auto_add_unknown_rules` If FALSE, an error will be generated if some of the responses do not appear in the scoring rules. Default is TRUE.

Details

It is a common practice to keep respons data in tables where each row contains the responses from a single person. This function is provided to input data in that form, one booklet at a time. The starting point is a data frame. Getting the data frame into R is left to the user. We have found packages `readxl` to be very good at reading Excel sheets, and `haven` quite efficient with SPSS files.

If the dataframe `x` contains a variable named `person_id` this variable will be used to identify unique persons. It is assumed that a single person will only make a single booklet once, otherwise an error will be generated.

If a person_id is not supplied, dexter will generate unique person_id's for each row of data.

Any column whose name has an exact match in the scoring rules inputted with function `start_new_project` will be treated as an item; any column whose name has an exact match in the covariates will be treated as covariate. If a name matches both a covariate and an item, the item takes precedence. Variables other than items, covariates and person_id will be ignored.

If `auto_add_unknown_rules=TRUE`, any responses to an item that do not have an exact match in the scoring rules will be automatically given the lowest score of 0. To score missing data differently, or simply abide to good style, the user can include explicit entries for missing value indicators in the scoring rules.

Value

A list of:

`items` The names of the columns in `x` that were treated as items

`covariates` The names of the columns in `x` that were treated as person covariates

`not_listed` A data frame of all responses that will be treated as missing

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db",
  covariates=list(gender="<unknown>"))
head(verbAggrData)
add_booklet(db, verbAggrData, "agg")

close_project(db)

## End(Not run)
```

`add_item_properties` *Add item properties to a project*

Description

Adds item properties to an existing database

Usage

```
add_item_properties(db, item_properties, overwrite = TRUE,
  default_values = list())
```

Arguments

`db`	A handle to the database, e.g. the output of `start_new_project` or `open_project`
`item_properties`	A data frame containing the item properties. See details.
`overwrite`	Whether existing item properties should be overwritten (default=TRUE)
`default_values`	a list where the names are item_properties and the values are defaults. The defaults will be used as the value for an item for which the property is not specified, for example when you add new items using `touch_rules`. Default_values for an item property will only be processed the first time you define an item property.

Details

When entering response data in the form of a rectangular person x item table, it is easy to provide person properties but practically impossible to provide item properties. This function provides a possibility to do so. The order of the rows and columns in the data frame is not important but there must be a column called `item_id` containing the item id's exactly as entered before.

Note that is is not possible to add new items with this function, use `touch_rules` if you want to add new items to your project.

Value

nothing

See Also

`fit_domains`, `profile_plot` for possible uses of item_properties

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db",
  covariates=list(gender="<unknown>"))
head(verbAggrProperties)
add_item_properties(db, verbAggrProperties)
```

```
get_item_properties(db)

close_project(db)

## End(Not run)
```

---

add_test3DC *Add a standard setting booklet to a 3DC database*

Description

See create3DC for more information

Usage

```
add_test3DC(db3dc, parms, design, test_id, standards, mu, sigma,
  population = NULL, group_leader = "admin", omit = 0)
```

Arguments

db3dc	3dc database handle
parms	a parameters object produced by fit_enorm
design	a data.frame with columns item_id and cluster and optionally cluster_nbr and item_nbr. See details.
test_id	name/id of the test as it will be shown in the 3DC application.
standards	vector of standards to be set
mu	expected ability in population, used for scaling of the clusters
sigma	expected standard deviation of ability in population, used for scaling of the clusters
population	optionally a data.frame with columns test_score and frequency to use for visualisation in 3DC application. If NULL, the distribution will be derived from a simulation.
group_leader	Login name of the group leader, default is admin. The default password is always admin, but can be changed in the 3DC application.
omit	the tail probability of the testscores to be omitted. For example, if set to 0.1, the 10 Default is 0.0 (omit nothing)

as.data.frame.prms *deprecated method*

Description

deprecated, please use `coef()`

Usage

```
## S3 method for class 'prms'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	An object produced by function `fit_enorm`
`row.names`	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	Will be set to FALSE, so ignore.
`...`	any other parameters to the as.data.frame method are ignored silently

close_project *Close a project*

Description

This is just an alias for `DBI::dbDisconnect(db)`, included for completeness

Usage

```
close_project(db)
```

Arguments

`db`	a Dexter project db handle

coef.prms *extract enorm item parameters*

Description

extract enorm item parameters

Usage

```
## S3 method for class 'prms'
coef(object, ...)
```

Arguments

object	an enorm parameters object, generated by the function fit_enorm
...	futher arguments to coef, the following are currently supported;
	bayes_hpd_b width of Bayesian highest posterior density interval around mean_beta, value must be between 0 and 1, default is 0.95

Value

depends on the calibration method:

for CML a data.frame with columns: item_id, item_score, beta, SE_b

for Bayes a data.frame with columns: item_id, item_score, mean_beta, SD_beta, <bayes_hpd_b>_hpd_b_left, <bayes_hpd_b>_hpd_b_right

coef.rim *Coefficients for the interaction model*

Description

Extract parameters for the interaction and the Rasch model

Usage

```
## S3 method for class 'rim'
coef(object, ...)
```

Arguments

object	An object produced by function fit_inter
...	Further arguments currently ignored

`create3DC` *Create a database for the 3DC standard setting application*

Description

Creates an empty database for 3DC standard setting application

Usage

```
create3DC(export_name)
```

Arguments

`export_name`	path to a new 3DC database

Details

The data driven direct concensus (3DC) method of standard setting is described in Keuning et. al. (2017). To easily apply this procedure, we advise to use the free digital 3DC application. This application can be downloaded from the Cito website, see the 3DC application download page. The functions create3DC and add_test3DC can be used to produce a standard setting database that can be imported in the 3DC application.

If you want to apply the 3DC method using paper forms instead, you can use the function plot3DC to generate the forms from the 3DC database.

Value

a handle to the 3DC sqlite database

References

Keuning J., Straat J.H., Feskens R.C.W. (2017) The Data-Driven Direct Consensus (3DC) Procedure: A New Approach to Standard Setting. In: Blömeke S., Gustafsson JE. (eds) Standard Setting in Education. Methodology of Educational Measurement and Assessment. Springer, Cham

Examples

```
## Not run:
library(dplyr)
db = start_new_project(verbAggrRules, "verbAggression.db")

add_booklet(db, verbAggrData, "aggression")

par = fit_enorm(db)
pv = plausible_values(db, par)
mu = mean(pv$PV1)
sigma = sd(pv$PV1)
```

```
# We'll use the behavior an item depicts as a basis for making the clusters,
# thus creating clusters of similar items.

design = data.frame(item_id = verbAggrProperties$item_id,
   cluster = verbAggrProperties$behavior)

# specify the actual sample for display in the group_leader page

population = get_testscores(db) %>%
                group_by(test_score) %>%
                summarise(frequency=n())

db3dc = create3DC('test3DC.db')

add_test3DC(db3dc, parms=par, design, mu=mu, sigma=sigma,
            test_id='verbal_aggression', standards='verbally aggressive',
            population=population)

#get a preview
plot3DC(db3dc)


dbDisconnect(db3dc)
close_project(db)

## End(Not run)
```

`design_as_network` *Test design as network*

Description

Export the test design as an incidence matrix and a weight matrix

Usage

```
design_as_network(dataSrc, predicate = NULL, weights = c("items",
  "responses"))
```

Arguments

`dataSrc`	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
`predicate`	An optional expression to subset data, if NULL all data is used
`weights`	Weight the edges between booklets by the number of common `"items"` or `"responses"` (default is items).

Details

The output of this function can be passed to packages for network analysis such as `igraph` or `qgraph`. We prefer to not load these packages automatically as they are fairly large and rely on a number of dependencies.

Value

A list of two data frames:

`im`	incidence matrix
`wm`	weights matrix

Examples

```
## Not run:
dsgn = design_as_network(db)
# Check if design is connected
design_is_connected(dsgn)

## End(Not run)
```

`design_is_connected` *Test if design is connected*

Description

Use the output from design_as_network to check if your design is connected.

Usage

```
design_is_connected(design)
```

Arguments

`design`	Output from design_as_network

Value

TRUE or FALSE

Examples

```
## Not run:
# as an example, turn off some your booklets and see if you are
# still left with a connected design

dsgn = design_as_network(db, !(booklet_id %in% c('b1','b3','b4')))
design_is_connected(dsgn)

## End(Not run)
```

DIF *Exploratory test for Differential Item Functioning*

Description

Exploratory test for Differential Item Functioning

Usage

```
DIF(dataSrc, covariate, predicate = NULL)
```

Arguments

dataSrc	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
covariate	Person covariate name for subsetting
predicate	An optional expression to subset data, if NULL all data is used

Details

Tests for equality of relative item difficulties category locations across groups. Supplements the confirmatory approach of the profile plot

Value

An object of class `DIF_stats` holding statistics for overall-DIF and a matrix of statistics for DIF in the relative position of item-category parameters in the regular parameterization used e.g., by OPLM.

References

Bechger, T. M. and Maris, G (2015); A Statistical Test for Differential Item Pair Functioning. Psychometrika. Vol. 80, no. 2, 317–340.

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db",
  covariates=list(gender='unknown'))
add_booklet(db, verbAggrData, "agg")
dd = DIF(db,covariate="gender")
print(dd)
plot(dd)

close_project(db)

## End(Not run)
```

---

distractor_plot *Distractor plot*

Description

Produce a diagnostic distractor plot for an item

Usage

```
distractor_plot(dataSrc, item, predicate = NULL, nc = 1, nr = 1,
  legend = TRUE, ...)
```

Arguments

dataSrc	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, response, item_score and optionally booklet_id
item	The ID of the item to plot. A separate plot will be produced for each booklet that contains the item, or an error message if the item ID is not known. Each plot contains a non-parametric regression of each possible response on the total score.
predicate	An optional expression to subset data, if NULL all data is used
nc	An integer between 1 and 3. Number of columns when putting mutiple plots on the same page. Default is 1. May be ignored or adjusted if it does not make sense.
nr	An integer between 1 and 3. Number of rows when putting mutiple plots on the same page. Default is 1. May be ignored or adjusted if it does not make sense.
legend	logical, whether to include the legend. default is TRUE
...	further arguments to plot.

Details

Customisation of title and subtitle can be done by using the arguments main and sub. These arguments can contain references to the variables item_id, booklet_id, item_position(only if dataSrc is a dexter db), pvalue, rit and rir. References are made by prefixing these variables with a dollar sign. Variable names can optionally be postfixed with a sprintf style format string, e.g.: distractor_plot(db, main='item: $item_id', sub='Item rest correlation: $rir:.2f')

---

fit_domains *Estimate the Rasch and the Interaction model per domain*

Description

Estimate the parameters of the Rasch model and the Interaction model

Usage

```
fit_domains(dataSrc, item_property, predicate = NULL)
```

Arguments

dataSrc	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
item_property	The item property defining the domains (subtests)
predicate	An optional expression to subset data, if NULL all data is used

Details

We have generalised the interaction model for items having more than two (potentially, a largish number) of response categories. This function represents scores on subtests as super-items and analyses these as normal items.

Value

An object of class imp holding results for the Rasch model and the interaction model.

See Also

plot.rim, fit_inter, add_item_properties

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db")
add_booklet(db, verbAggrData, "agg")
add_item_properties(db, verbAggrProperties)
mSit = fit_domains(db, item_property= "situation")
plot(mSit)
```

```
close_project(db)

## End(Not run)
```

`fit_enorm` *Fit the extended nominal response model*

Description

Fits an Extended NOminal Response Model (ENORM) using conditional maximum likelihood (CML) or a Gibbs sampler for Bayesian estimation.

Usage

```
fit_enorm(dataSrc, predicate = NULL, fixed_params = NULL,
  method = c("CML", "Bayes"), nIterations = 500)
```

Arguments

`dataSrc`	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
`predicate`	An optional expression to subset data, if NULL all data is used
`fixed_params`	Optionally, a prms object from a previous analysis or a data.frame with columns: item_id, item_score (omitting 0 score category) and beta
`method`	If CML, the estimation method will be Conditional Maximum Likelihood; otherwise, a Gibbs sampler will be used to produce a sample from the posterior
`nIterations`	Number of Gibbs samples when estimation method is Bayes. The maximum number of iterations when using CML.

Value

An object of type `prms`. The prms object can be cast to a data.frame of item parameters using the `as.data.frame` built-in function or used directly as input for other Dexter functions.

References

Maris, G., Bechger, T.M. and San-Martin, E. (2015) A Gibbs sampler for the (extended) marginal Rasch model. Psychometrika. 2015; 80(4): 859–879.

See Also

functions that accept a prms object as input: `ability`, `plausible_values`

`fit_inter` *Estimate the Interaction and the Rasch model*

Description

Estimate the parameters of the Interaction model and the Rasch model

Usage

```
fit_inter(dataSrc, predicate = NULL)
```

Arguments

`dataSrc`	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
`predicate`	An optional expression to subset data, if NULL all data is used

Details

Unlike the Rasch model, the interaction model cannot be computed concurrently for a whole design of test forms. This function fits the Rasch model and the interaction model on a complete rectangular array of responses. This typically consist of responses to items in one booklet but can also consist of the intersection (common items) of two or more booklets. If the intersection is empty (no common items for all persons), the function will exit with an error message.

Value

An object of class `rim` holding results for the Rasch model and the interaction model.

See Also

`plot.rim`, `fit_domains`

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db")
add_booklet(db, verbAggrData, "agg")

m = fit_inter(db, booklet_id=='agg')
plot(m, "S1DoScold", show.observed=TRUE)

close_project(db)

## End(Not run)
```

get_booklets *Booklets entered in a project*

Description

Retrieve information about the booklets entered in the db so far

Usage

```
get_booklets(db)
```

Arguments

db	A handle to the database, i.e. the output of `start_new_project` or `open_project`

Value

A data frame showing with columns: booklet_id, n_persons and n_items.

get_design *Test design*

Description

Retrieve all items that have been entered in the db so far by booklet and position in the booklet

Usage

```
get_design(db, format = c("wide", "long"), rows = c("booklet_id", "item_id",
  "item_position"), columns = c("item_id", "booklet_id", "item_position"),
  fill = NA)
```

Arguments

db	A handle to the database, i.e. the output of `start_new_project` or `open_project`
format	return format, see below
rows	variable that defines the rows, ignored if format='long'
columns	variable that defines the columns, ignored if format='long'
fill	If set, missing values will be replaced with this value

Value

A data.frame with the design. The contents depend on the rows, columns and format parameters if ‘format‘ is ‘’long’‘ a data.frame with columns: booklet_id, item_id, item_position if ‘format‘ is ‘’wide’‘ a data.frame with the rows defined by the ‘rows‘ parameter and the columns by the ‘columns‘ parameter, with the remaining variable (i.e. item_id, booklet_id or item_position) making up the cells

get_items *Items in a project*

Description

Retrieve all items that have been entered in the db so far together with the item properties

Usage

```
get_items(db)
```

Arguments

db	A handle to the database, i.e. the output of `start_new_project` or `open_project`

Value

A data frame with column item_id and a column for each item property

get_item_properties *Item properties in a project*

Description

Retrieve information about the item properties defined in the project (if any). This will return a data.frame with one row per unique value of each item property.

Usage

```
get_item_properties(db)
```

Arguments

db	A handle to the database, i.e. the output of `start_new_project` or `open_project`

Value

A data.frame with one row per unique value of each item property and a count of the nbr of items it applies to. The columns are: item_property, value, N

get_persons *Persons in a project*

Description

Retrieve all persons/respondents that have been entered in the db so far together with their covariates

Usage

```
get_persons(db)
```

Arguments

db	A handle to the database, i.e. the output of `start_new_project` or `open_project`

Value

A data frame with columns person_id and columns for each person_property/covariate

get_person_properties *Person properties in a project*

Description

Retrieve information about the item properties defined in the project (if any). This will return a data.frame with one row per unique value of each item property.

Usage

```
get_person_properties(db)
```

Arguments

db	A handle to the database, i.e. the output of `start_new_project` or `open_project`

Value

A data.frame with one row per unique value of each person property and a count of the nbr of persons it applies to. The columns are: person_property, value, N

get_responses *Selecting data*

Description

Extract data from a dexter database

Usage

```
get_responses(dataSrc, predicate = NULL, columns = c("person_id", "item_id",
  "item_score"), env = NULL)
```

Arguments

dataSrc	a dexter project database or data.frame
predicate	an expression to select data on
columns	the columns you wish to select, can include all columns in the project, see: get_variables
env	optionally, an environment to evaluate the expression in

Details

Many functions in Dexter accept a data source and a predicate. Predicates are extremely flexible but they have a few limitations because they work on the individual response level. It is therefore not possible for example, to remove complete person cases from an analysis based on responses to a single item by using just a predicate expression.

For such cases, Dexter supports selecting the data and manipulating it before passing it back to a Dexter function or possibly doing something else with it. The following example will hopefully clarify this.

Value

a data.frame of responses

Examples

```
## Not run:
# goal: fit the extended nominal response model using only persons
# without any missing responses
library(dplyr)

# the following would not work since it will omit only the missing
# responses, not the persons; which is not what we want in this case
wrong = fit_enorm(db, response != 'NA')

# to select on an aggregate level, we need to gather the data and
```

```
# manipulate it ourselves
data = get_responses(db,
   columns=c('person_id','item_id','item_score','response')) %>%
   group_by(person_id) %>%
   mutate(any_missing = any(response=='NA')) %>%
   filter(!any_missing)

correct = fit_enorm(data)


## End(Not run)
```

`get_rules` *Get scoring rules*

Description

Retrieve the scoring rules currently present in the dexter project db

Usage

```
get_rules(db)
```

Arguments

db	handle to a Dexter project database

Value

data.frame of scoring rules containing columns: item_id, response, item_score

---

`get_testscores` *Provide test scores*

Description

Supplies the weighted sum of item scores for each person selected.

Usage

```
get_testscores(dataSrc, predicate = NULL)
```

Arguments

dataSrc	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
predicate	An optional expression to filter data, if NULL all data is used

Value

A tibble with columns person_id, item_id, test_score

---

get_variables *Variables that are defined in the project*

Description

Inspect the variables defined in your dexter project and their datatypes

Usage

```
get_variables(db)
```

Arguments

db	a dexter project database

Details

The variables in Dexter consist of the item properties and person covariates you specified and a number of reserved variables that are automatically defined like `response` and `booklet_id`.

Variables in Dexter are most useful when used in predicate expressions. A number of functions can take a dataSrc argument and an optional predicate. Predicates are a concise and flexible way to filter data for the different psychometric functions in Dexter.

The variables can also be used to retrieve data in `get_responses`

Value

a data.frame with name and type of the variables defined in your dexter project

---

iModels *Interactive model display*

Description

Opens up a shiny application with item statistics and interactive plots for the Rasch and Interaction models

Usage

```
iModels(db, booklet)
```

Arguments

db	A handle to the database, i.e. the output of `create_new_project` or `open_project`
booklet	booklet_id of the booklet that will be shown

Value

An object that represents the application. Printing the object or passing it to `shiny::runApp` will run the app.

---

individual_differences

Test individual differences

Description

Test individual differences

Usage

```
individual_differences(dataSrc, predicate = NULL, degree = 7)
```

Arguments

dataSrc	Data source: a dexter project db handle or a data.frame
predicate	An optional expression to subset data, if NULL all data are used.
degree	The degree of a polynomial used to smooth observed score distribution.

Details

This function uses a score distribution to test whether there are individual differences in ability. First, it estimates ability based on the score distribution. Then, the observed distribution is compared to the one expected from the single estimated ability. The data are typically from one booklet but can also consist of the intersection (i.e., the common items) of two or more booklets. If the intersection is empty (no common items for all persons), the function will exit with an error message.

Value

an object of type tind. Printing the object will show test results. Plotting it will produce a plot of expected and observed score frequencies. The former under the hypothesis that there are no individual differences.

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db")
add_booklet(db, verbAggrData, "agg")
dd = individual_differences(db)
print(dd)
plot(dd)

close_project(db)

## End(Not run)
```

---

iTIA *Interactive test-item analysis*

Description

Open a shiny application to do interactive item-test analysis on the database

Usage

```
iTIA(db)
```

Arguments

db	A handle to the database, i.e. the output of `create_new_project` or `open_project`

Value

An object that represents the application. Printing the object or passing it to `shiny::runApp` will run the app.

---

keys_to_rules *Derive scoring rules from keys*

Description

For multiple choice items that will be scored as 0/1, derive the scoring rules from the keys to the correct responses

Usage

```
keys_to_rules(keys, include_NA_rule = FALSE)
```

Arguments

keys	A data frame containing columns item_id, nOptions, and key (the spelling is important). See details.
include_NA_rule	whether to add an option 'NA' (which is scored 0) to each item

Details

This function might be useful in setting up the scoring rules when all items are multiple-choice and scored as 0/1. (Hint: Because the order in which the scoring rules is not important, one can use the function to generate rules for many MC items and then append per hand the rules for a few complex items.)

The input data frame must contain the exact name of each item, the number of options, and the key. If the keys are all integers, it will be assumed that responses are coded as 1 through nOptions. If they are all uppercase letters, it is assumed that responses are coded as A,B,C,... All other cases result in an error.

Value

A data frame that can be used as input to start_new_project

---

open_project *Open an existing project*

Description

Opens a database created by function start_new_project

Usage

```
open_project(db_name = "dexter.db", convert_old = FALSE)
```

Arguments

db_name	The name of the data base to be opened.
convert_old	automatically try to convert databases from older versions of Dexter that are no longer supported.

Value

A handle to the data base.

PISA_item_class *Item properties in the PISA 2012 example*

Description

A data set of item properties in the PISA 2012 example (see the help screen for function add_booklet)

Format

A data set with 109 rows and 6 columns.

plausible_scores *Generate plausible testscores*

Description

Generate plausible i.e., posterior predictive sumscores on a set of items. A typical use of this function is to generate plausible scores on a complete item bank when data is collected using an incomplete design

Usage

```
plausible_scores(dataSrc, predicate = NULL, parms = NULL, items = NULL,
  covariates = NULL, keep.observed = TRUE, nPS = 1)
```

Arguments

dataSrc	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
predicate	an expression to filter data. If missing, the function will use all data in dataSrc
parms	An object returned by function `fit_enorm` and containing parameter estimates. If parms is given the function provides plausible scores conditional on the item parameters. These are considered known. If `parms` is `NULL`, Bayesian parameters are calculated from the datasrc
items	vector of item_id's, this specifies the itemset to generate the testscores for. If `items` is `NULL` all items occurring in `dataSrc` are used.
covariates	name or a vector of names of the variables to group the population, used to update the prior. A covariate must be a discrete person covariate (e.g. not a float) that indicates nominal categories, e.g. gender or school If dataSrc is a data.frame, it must contain the covariate.
keep.observed	In some cases, responses to one or more of the items have been observed. The user can choose to keep these observations or generate new ones.
nPS	Number of plausible testscore to generate per person.

Value

A data.frame with columns booklet_id, person_id, sumScore and nPS plausible scores named PS1...PSn.

`plausible_values` *Draw plausible values*

Description

Draws plausible values based on sum scores

Usage

```
plausible_values(dataSrc, parms = NULL, predicate = NULL,
  covariates = NULL, nPV = 1, use_draw = NULL)
```

Arguments

`dataSrc`	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
`parms`	An object returned by function `fit_enorm` and containing parameter estimates. If parms is given the function provides plausible values conditional on the item paramaters. These are considered known. If parameter estimates are not given, the user is given plausible value marginalized over the posterior distribution of the item parameters. In plain words, this means that the uncertainty of the item parameters is taken into account.
`predicate`	an expression to filter data. If missing, the function will use all data in dataSrc
`covariates`	name or a vector of names of the variables to group the population, used to update the prior. A covariate must be a discrete person covariate (e.g. not a float) that indicates nominal categories, e.g. gender or school If dataSrc is a data.frame, it must contain the covariate.
`nPV`	Number of plausible values to draw per person.
`use_draw`	When the ENORM was fitted with a Gibbs sampler (this is recognised automatically), the number of the random draw (iteration) to use in generating the PV. If NULL, all draws will be averaged; that is, the posterior means are used for the item parameters. If outside range, the last iteration will be used.

Value

A data.frame with columns booklet_id, person_id, sumScore and nPV plausible values named PV1...PVn.

References

Marsman, M., Maris, G., Bechger, T. M., and Glas, C.A.C. (2016) What can we learn from plausible values? Psychometrika. 2016; 81: 274–289.

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db",
   covariates=list(gender="<unknown>"))
add_booklet(db, verbAggrData, "agg")
add_item_properties(db, verbAggrProperties)
f=fit_enorm(db)
par(mfrow=c(1,2))
pv_M=plausible_values(db,f,(mode=="Do")&(gender=="Male"))
pv_F=plausible_values(db,f,(mode=="Do")&(gender=="Female"))
plot(ecdf(pv_M$PV1),
   main="Do: males versus females", xlab="Ability", col="red")
lines(ecdf(pv_F$PV1), col="green")
legend(-2.2,0.9, c("female", "male") ,
   lty=1, col=c('green', 'red'), bty='n', cex=.75)

pv_M=plausible_values(db,f,(mode=="Want")&(gender=="Male"))
pv_F=plausible_values(db,f,(mode=="Want")&(gender=="Female"))

plot(ecdf(pv_M$PV1),
   main="Want: males versus females", xlab=" Ability", col="red")
lines(ecdf(pv_F$PV1),col="green")
legend(-2.2,0.9, c("female", "male") ,
   lty=1, col=c('green', 'red'), bty='n', cex=.75)

close_project(db)

## End(Not run)
```

`plot.rim` *A plot method for the interaction model*

Description

Plot the item-total regressions fit by the interaction (or Rasch) model

Usage

```
## S3 method for class 'rim'
plot(x, items = NULL, summate = TRUE, overlay = FALSE,
  nc = 1, nr = 1, curtains = 10, show.observed = FALSE, ...)
```

Arguments

`x`	An object produced by function `fit_inter`
`items`	The items to plot (item_id's). If NULL, all items will be plotted
`summate`	If FALSE, regressions for polytomous items will be shown for each response option separately; default is TRUE.

overlay	If TRUE and more than one item is specified, there will be two plots, one for the Rasch model and the other for the interaction model, with all items overlayed; otherwise, one plot for each item with the two models overlayed. Ignored if summate is FALSE. Default is FALSE
nc	An integer between 1 and 3. Number of columns when putting mutiple plots on the same page. Default is 1. May be ignored or adjusted if it does not make sense.
nr	An integer between 1 and 3. Number of rows when putting mutiple plots on the same page. Default is 1. May be ignored or adjusted if it does not make sense.
curtains	100*the tail probability of the sum scores to be shaded. Default is 10. Set to 0 to have no curtains shown at all.
show.observed	If TRUE, the observed proportion correct at each sum score will be shown as dots. Default is FALSE.
...	Any additional plotting parameters.

Details

Customisation of title and subtitle can be done by using the arguments main and sub. These arguments can contain references to the variables item_id (if overlay=FALSE) or model (if overlay=TRUE) by prefixing them with a dollar sign, e.g. plot(m, main='item: $item_id')

---

plot3DC *Show 3DC plots*

Description

Show 3DC plots as used in 3DC standard setting application

Usage

```
plot3DC(db3dc, test_id = NULL, ...)
```

Arguments

db3dc	3dc database handle
test_id	optionally, a vector of test_id's. If omitted, plots for all tests will be shown
...	further arguments to plot. Some of them have useful defaults

print.prms *A print method for ENORM parameters*

Description

A print method for ENORM parameters

Usage

```
## S3 method for class 'prms'
print(x, ...)
```

Arguments

x	An object produced by function fit_enorm
...	Any other parameters to the print method are silently ignored

print.rim *A print method for the interaction model*

Description

Print the available items for plots of the Rasch and the interaction models

Usage

```
## S3 method for class 'rim'
print(x, ...)
```

Arguments

x	An object produced by function fit_inter
...	Further arguments to print

`profile_plot` *Profile plot*

Description

Profile plot

Usage

```
profile_plot(dataSrc, item_property, covariate, predicate = NULL,
  model = "IM", x = NULL, ...)
```

Arguments

`dataSrc`	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score and the item_property and the covariate of interest.
`item_property`	The name of the item property defining the domains. The item property should have exactly two distinct values in your data
`covariate`	name of the person property/covariate used to create the groups. There will be one line for each distinct value.
`predicate`	An optional expression to filter data, if NULL all data is used
`model`	"IM" (default) or "RM" where "IM" is the interaction model and "RM" the Rasch model. The interaction model is the default as it fits the data better or at least as good as the Rasch model.
`x`	Which value of the item_property to draw on the x axis, if NULL, one is chosen automatically
`...`	further arguments to plot, many have useful defaults

Details

Profile plots can be used to investigate whether two (or more) groups of respondents attain the same test score in the same way. The user must provide a (meaningfull) classification of the items in two non-overlapping subsets such that the test score is the sum of the scores on the subsets. The plot shows the probabilities to obtain any combinations of subset scores with thin gray lines indicating the combinations that give the same test score. The thick lines connect the most likely combination for each test score in each group. When applied to educational test data, the plots can be used to detect differences in the relative difficulty of (sets of) items for respondents that belong to different groups and are matched on the test score. This provides a content-driven way to investigate differential item functioning.

Value

Nothing interesting

Examples

```
## Not run:
db = start_new_project(verbAggrRules, "verbAggression.db",
  covariates=list(gender="<unknown>"))
add_booklet(db, verbAggrData, "agg")
add_item_properties(db, verbAggrProperties)
profile_plot(db, item_property='mode', covariate='gender')

close_project(db)

## End(Not run)
```

`read_oplm_par` *Read item parameters from oplm PAR or CML files*

Description

Read item parameters from oplm PAR or CML files

Usage

```
read_oplm_par(par_path)
```

Arguments

`par_path`	path to a file in the (binary) OPLM PAR format or the human readable CML format

Details

It is occasionally useful to calibrate new items on an existing scale. This function offers the possibility to read parameters from the proprietary oplm format so that they can be used to fix a new calibration in Dexter on an existing scale of items that were calibrated in oplm.

Value

depends on the input. For .PAR files a tibble with columns: item_id, item_score, beta, nr; for .CML files also several statistics columns that are outputted by OPLM as part of the calibration.

Examples

```
par = read_oplm_par('/parameters.PAR')
f = fit_enorm(db, fixed_params=par)
```

show_rules *Deprecated function names*

Description

All 'show_<something>' functions have been renamed to 'get_<something>'.

Usage

```
show_rules(db)

show_booklets(db)

show_item_properties(db)

show_person_properties(db)

show_items(db)

show_design(db)

show_persons(db)
```

Arguments

db	handle to a Dexter project database

Value

data.frame

See Also

get_booklets, get_design, get_item_properties, get_items, get_person_properties, get_persons, get_rules

start_new_project *Start a new project*

Description

Imports a complete set of scoring rules and starts a new project (data base)

Usage

```
start_new_project(rules, db = "dexter.db", covariates = NULL)
```

Arguments

rules	A data frame with columns item_id, response, and item_score. The order is not important but spelling is. Any other columns will be ignored.
db	A connection to an existing sqlite database or a string specifying a filename for a new sqlite database to be created. If this name does not contain a path, the file will be created in the work directory. Any existing file with the same name will be overwritten.
covariates	An optional list of person covariates. Names should correspond to covariates intended to be used in the project. Values are used as default (missing) values for these covariates. The datatype will also be inferred from the values. Known covariates will be imported (if supplied) in add_booklet.

Details

This package only works with closed items (e.g. likert, MC or possibly short answer): it does not score any open items. The first step to creating a project is to import an exhaustive list of all items and all admissible responses, along with the score that any of the latter will be given. Responses may be integers or strings but they will always be treated as strings. Scores must be integers, and the minimum score for an item must be 0. When inputting data, all responses not specified in the rules can optionally be treated as missing and ultimately scored 0, but it is good style to include the missing responses in the list. NA values will be treated as the string 'NA'.

Value

If the scoring rules pass a sanity check, a handle to the data base. Otherwise, a data frame listing the problems found, with 4 columns: item_id: id of the problematic item less_than_two_scores: if TRUE, the item has only one distinct score duplicated_responses: if TRUE, the item contains two or more identical response categories min_score_not_zero: if TRUE, the minimum score of the item was not 0

Examples

```
head(verbAggrRules)
db = start_new_project(verbAggrRules, "verbAggression.db",
  covariates=list(gender="<unknown>"))
```

---

start_new_project_from_oplm

Start a new project from oplm files

Description

Creates a dexter project database and fills it with response data based on a .dat and .scr file

Usage

```
start_new_project_from_oplm(dbname, scr_path, dat_path, booklet_position,
  responses_start, response_length = 1, person_id = NULL,
  missing_character = c(" ", "9"), use_discrim = FALSE,
  format = "compressed")
```

Arguments

dbname	filename/path of new dexter project database (will be overwritten if already exists)
scr_path	path to the .scr file
dat_path	path to the .dat file
booklet_position	vector of start and end of booklet position in the dat file, e.g. c(1,4), all positions are counted from 1, start and end are both included.
responses_start	start position of responses in the .dat file
response_length	length of individual responses, default=1
person_id	optionally, a vector of start and end position of person_id in the .dat file. If NULL, person id's will be auto-generated.
missing_character	character used to indicate missing in .dat file, default is to use both a space and a 9 as missing characters.
use_discrim	if TRUE, the scores for the responses will be multiplied by the discrimination parameters of the items
format	not used, at the moment only the compressed format is supported.

Details

start_new_project_from_oplm builds a complete dexter database from a .dat and .scr file in the proprietary oplm format. Three custom variables are added to the database: booklet_on_off, item_local_on_off, item_global_on_off. These are taken from the .scr file and can be used in expressions in the various dexter functions.

Value

a handle to the data base.

Examples

```
db = start_new_project_from_oplm('test.db',
   'path_to_scr_file', 'path_to_dat_file',
   booklet_position=c(1,3), responses_start=101,
   person_id=c(50,62))
```

```
par = fit_enorm(db,
   (item_global_on_off==1)&(item_local_on_off==1)&(booklet_on_off==1))

abilities = ability(db, par,
   (item_global_on_off==1)&(item_local_on_off==1)&(booklet_on_off==1))

head(abilities)
```

`tia_tables` *Simple test-item analysis*

Description

Show simple Classical Test Analysis statistics at item and test level

Usage

```
tia_tables(dataSrc, predicate = NULL, type = c("raw", "averaged",
  "compared"))
```

Arguments

`dataSrc`	Data source: a dexter project db handle or a data.frame with columns: person_id, item_id, item_score
`predicate`	An optional expression to subset data, if NULL all data is used
`type`	How to present the item level statistics: `raw` for each test booklet separately, `averaged` averaged over the test booklet in which the item is included, with the number of persons as weights, or compared, in which case the pvalues, correlations with the sum score (rit), and correlations with the rest score (rit) are shown in separate tables and compared across booklets

Value

A list containing:

`testStats`	a data frame of statistics at test level
`itemStats`	a data frame of statistics at item level

.

touch_rules *Add or modify scoring rules*

Description

Having to alter or add a scoring rule is occasionally necessary, e.g. in case of a key error. This function offers the possibility to do so and also allows you to add new items to your project

Usage

```
touch_rules(db, rules)
```

Arguments

db	handle to a Dexter project database
rules	A data frame with columns item_id, response, and item_score. The order is not important but spelling is. Any other columns will be ignored. See details

Details

The rules should contain all rules that you want to change or add. This means that in case of a key error in a single multiple choice question, you typically have to change two rules.

Value

If the scoring rules pass a sanity check, a small summary of changes is printed and nothing is returned Otherwise this function returns a data frame listing the problems found, with 4 columns: item_id: id of the problematic item less_than_two_scores: if TRUE, the item has only one distinct score duplicated_responses: if TRUE, the item contains two or more identical response categories min_score_not_zero: if TRUE, the minimum score of the item was not 0 Please note that the sanity check is done for the items you change

Examples

```
# given that in your dexter project there is an mc item with id 'itm_01',
# which currently has key 'A' but you want to change it to 'C'.

new_rules = data.frame(item_id='itm_01', response=c('A','C'), item_score=c(0,1))
touch_rules(db, new_rules)
```

verbAggrData *Verbal aggression data*

Description

A data set of self-reported verbal behaviour in different frustrating situations (Vansteelandt, 2000)

Format

A data set with 316 rows and 25 columns.

verbAggrProperties *Item properties in the verbal aggression data*

Description

A data set of item properties related to the verbal aggression data

Format

A data set with 24 rows and 5 columns.

verbAggrRules *Scoring rules for the verbal aggression data*

Description

A set of (trivial) scoring rules for the verbal aggression data set

Format

A data set with 72 rows and 3 columns (item_id, response, item_score).

Index