Package 'dexter'

January 6, 2021

Type Package

Title Data Management and Analysis of Tests

Version 1.1.2

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

Maintainer Jesse Koops <jesse.koops@cito.nl>

Description A system for the management, assessment, and psychometric analysis of data from educational and psychological tests.

License GPL-3

URL https://dexterities.netlify.app/

BugReports https://github.com/jessekps/dexter/issues

Encoding UTF-8

LazyData yes

Depends R (>= 3.4)

Imports RSQLite (>= 2.1), DBI (>= 1.0.0), MASS (>= 7.3), tidyr (>= 0.8.3), rlang (>= 0.4.0), dplyr (>= 0.8.3), Rcpp (>= 1.0.1), graphics, grDevices, methods, utils

LinkingTo Rcpp, RcppArmadillo (>= 0.9.3)

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, latticeExtra, testthat, ggplot2, Cairo

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-01-06 16:00:53 UTC

R topics documented:

`dexter-package` *Dexter: data analyses for educational and psychological tests.*

Description

Dexter provides a comprehensive solution for managing and analyzing educational test data.

Details

The main features are:

- project databases providing a structure for storing data about persons, items, responses and booklets.
- methods to assess data quality using Classical test theory and plots.
- CML calibration of the extended nominal response model and interaction model.

To learn more about dexter, start with the vignettes: ‘browseVignettes(package="dexter")‘

See Also

Useful links:

- https://dexterities.netlify.app/
- Report bugs at https://github.com/jessekps/dexter/issues

`ability` *Estimate abilities*

Description

Computes estimates of ability for persons or for booklet scores

Usage

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

```
  merge_within_persons = FALSE
)

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

Arguments

dataSrc	a connection to a dexter database, a matrix, or a data.frame with columns: person_id, item_id, item_score
parms	object produced by fit_enorm or a data.frame with columns item_id, item_score and, depending on parametrization, a column named either beta/delta, eta or b
predicate	An optional expression to subset data, if NULL all data is used
method	Maximum Likelihood (MLE), Expected A posteriori (EAP) or Weighted Likelihood (WLE)
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, use_draw is the index of the posterior sample of the item parameters that will be used for 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
merge_within_persons	for persons who were administered multiple booklets, whether to provide just one ability value (TRUE) or one per booklet(FALSE)
design	A data.frame with columns item_id and optionally booklet_id. If the column booklet_id is not included, the score transformation table will be based on all items found in the design. If design is NULL and parms is an enorm fit object the score transformation table will be computed based on the test design that was used to fit the items.

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 WLE. The WLE was proposed by Warm (1989) to reduce bias in the MLE and is also known as the Warm estimator.

Value

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

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

References

Warm, T. A. (1989). Weighted likelihood estimation of ability in item response theory. Psychometrika, 54(3), 427-450.

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$booklet_score, bb$theta, xlab="test-score", ylab="ability est.", pch=19, cex=0.7)
points(aa$booklet_score, aa$theta, col="red", pch=19, cex=0.7)
points(aa$booklet_score, cc$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 response data to a project*

Description

Add item response data in long or wide format

Usage

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

add_response_data(
  db,
  data,
  auto_add_unknown_rules = FALSE,
  missing_value = "NA"
)
```

Arguments

db	a connection to a dexter database, i.e. the output of `start_new_project` or `open_project`
x	A data frame containing the responses and, optionally, person_properties. 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 person_properties. See details.
booklet_id	A (short) string identifying the test form (booklet)
auto_add_unknown_rules	If FALSE (the default), an error will be generated if one or more responses do not appear in the scoring rules. If TRUE, unknown responses will be assumed to have a score of 0.
data	response data in normalized (long) format. Must contain columns `person_id`, `booklet_id`, `item_id` and `response` and optionally `item_position` (useful if your data contains new booklets, see details)
missing_value	value to use for responses in missing rows in your data, see details

Details

It is a common practice to keep response data in tables where each row contains the responses from a single person. `add_booklet` is provided to input data in that form, one booklet at a time.

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 person_properties will be treated as a person property. If a name matches both a person_property and an item_id, the item takes precedence. Columns other than items, person properties and person_id will be ignored.

`add_response_data` can be used to add data that is already 'normalized'. This function takes a data.frame in long format with columns `person_id`, `booklet_id`, `item_id` and `response` such as can usually be found in databases for example. The first time a new booklet is encountered, the design (i.e. which items are contained in each booklet at each position) is derived from `data`. In this case it is useful if you specify an extra column named `item_position`, otherwise dexter will

generate the item_positions automatically in some way that may not reflect your actual design (of course, if the item positions in your tests are randomized, that is not a problem).

If there are missing rows (e.g. there are only 9 rows for a person-booklet where the booklet should contain 10 items) `missing_value` will be used for the omitted responses. This can lead to an error in case `missing_value` is not defined in your rules and `auto_add_unknown_rules` is set to FALSE (the default). Please also note that the booklet_design for any specific booklet is derived from the distinct combination of booklet_id and item_id in `data` the first time that booklet is encountered. If subsequent calls to `add_response_data` contain data with more or different items for this same booklet, this will cause an error.

Note that responses are always treated as strings (in both functions), and `NA` values are transformed to the string `"NA"`.

Value

A list with information about the recent import.

Examples

```
db = start_new_project(verbAggrRules, ":memory:",
                       person_properties=list(gender="unknown"))
head(verbAggrData)
add_booklet(db, verbAggrData, "agg")

close_project(db)
```

`add_item_properties` *Add item properties to a project*

Description

Add, change or define item properties in a dexter project

Usage

```
add_item_properties(db, item_properties = NULL, default_values = NULL)
```

Arguments

`db`	a connection to a dexter database, e.g. the output of `start_new_project` or `open_project`
`item_properties`	A data frame containing a column item_id (matching item_id's already defined in the project) and 1 or more other columns with item properties (e.g. item_type, subject)
`default_values`	a list where the names are item_properties and the values are defaults. The defaults will be used wherever the item property is unknown.

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.

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: \donttest{
db = start_new_project(verbAggrRules, "verbAggression.db")
head(verbAggrProperties)
add_item_properties(db, verbAggrProperties)
get_items(db)

close_project(db)
}
## End(Not run)
```

---

add_person_properties *Add person properties to a project*

Description

Add, change or define person properties in a dexter project. Person properties defined here will also be automatically imported with `add_booklet`

Usage

```
add_person_properties(db, person_properties = NULL, default_values = NULL)
```

Arguments

db	a connection to a dexter database, e.g. the output of `start_new_project` or `open_project`
person_properties	A data frame containing a column person_id and 1 or more other columns with person properties (e.g. education_type, birthdate)
default_values	a list where the names are person_properties and the values are defaults. The defaults will be used wherever the person property is unknown.

Details

Due to limitations in the sqlite database backend that we use, the default values for a person property can only be defined once for each person_property

Value

nothing

`close_project` *Close a project*

Description

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

Usage

```
close_project(db)
```

Arguments

`db`	connection to a dexter database

`coef.p2pass` *extract equating information*

Description

extract equating information

Usage

```
## S3 method for class 'p2pass'
coef(object, ...)
```

Arguments

`object`	an p2pass object, generated by `probability_to_pass`
`...`	further arguments are currently ignored

Value

A data.frame with columns:

booklet_id id of the target booklet

score_new score on the target booklet

probability_to_pass probability to pass on the reference test given score_new

true_positive percentages that correctly passes

sensitivity The proportion of positives that are correctly identified as such

specificity The proportion of negatives that are correctly identified as such

proportion proportion in sample with score_new

`coef.prms` *extract enorm item parameters*

Description

extract enorm item parameters

Usage

```
## S3 method for class 'prms'
coef(object, hpd = 0.95, what = c("items", "var", "posterior"), ...)
```

Arguments

`object`	an enorm parameters object, generated by the function `fit_enorm`
`hpd`	width of Bayesian highest posterior density interval around mean_beta, value must be between 0 and 1, default is 0.95
`what`	which coefficients to return. Defaults to ‘items‘ (the item parameters). Can also be ‘var‘ for the variance-covariance matrix (CML only) or ‘posterior‘ for all draws of the item parameters (Bayes only)
`...`	further arguments to coef are ignored

Value

Depends on the calibration method and the value of ’what’. For ’items’#’

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

Bayes a data.frame with columns: item_id, item_score, mean_beta, SD_beta, <hpd_b_left>, <hpd_b_right>

The posterior distribution and variance covariance matrix are returned as matrices.

design_info *Information about the design*

Description

This function is useful to inspect incomplete designs

Usage

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

Arguments

dataSrc	a connection to a dexter database, a matrix, 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

Value

a list with the following components

design a data.frame with columns booklet_id, item_id, item_position, n_persons

connected_booklets a data.frame with columns booklet_id, group; booklets with the same 'group' are connected to each other.

connected TRUE/FALSE indicating whether the design is connected or not

testlets a data.frame with columns item_id and testlet; items within the same testlet always occur together in a booklet

adj_matrix list of two adjacency matrices: *weighted_by_items* and *weighted_by_persons*; These matrices can be useful in visually inspecting the design using a package like *igraph*

DIF *Exploratory test for Differential Item Functioning*

Description

Exploratory test for Differential Item Functioning

Usage

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

Arguments

dataSrc	a connection to a dexter database or a data.frame with columns: person_id, item_id, item_score
person_property	Defines groups of persons to calculate DIF
predicate	An optional expression to subset data, if NULL all data is used

Details

Tests for equality of relative item/category difficulties 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 beta-parameterization where they represent locations on the ability scale where adjacent categories are equally likely. If there is DIF, the function 'plot' can be used to produce an image of the pairwise DIF statistics.

References

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

See Also

A plot of the result is produced by the function `plot.DIF_stats`

Examples

```
db = start_new_project(verbAggrRules, ":memory:", person_properties=list(gender='unknown'))
add_booklet(db, verbAggrData, "agg")
dd = DIF(db,person_property="gender")
print(dd)
plot(dd)
str(dd)

close_project(db)
```

---

distractor_plot *Distractor plot*

Description

Produce a diagnostic distractor plot for an item

Usage

```
distractor_plot(
  dataSrc,
  item_id,
  predicate = NULL,
  legend = TRUE,
  curtains = 10,
  adjust = 1,
  col = NULL,
  ...
)
```

Arguments

dataSrc	a connection to a dexter database or a data.frame with columns: person_id, item_id, response, item_score and optionally booklet_id
item_id	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
legend	logical, whether to include the legend. default is TRUE
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.
adjust	factor to adjust the smoothing bandwidth respective to the default value
col	vector of colors to use for plotting
...	further arguments to plot.

Details

Customization 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(if available), pvalue, rit and rir. References are made by prefixing these variables with a dollar sign. Variable names may be postfixed with a sprintf style format string, e.g. `distractor_plot(db,main='item: $item_id',sub='Item rest correlation: $rir:.2f')`

Value

Silently, a data.frame of response categories and colors used. Potentially useful if you want to customize the legend or print it separately

`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`	a connection to a dexter database 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

```
db = start_new_project(verbAggrRules, ":memory:")
add_booklet(db, verbAggrData, "agg")
add_item_properties(db, verbAggrProperties)
mSit = fit_domains(db, item_property= "situation")
plot(mSit)

close_project(db)
```

`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"),
  nDraws = 1000,
  merge_within_persons = FALSE,
  nIterations = NULL
)
```

Arguments

`dataSrc`	a connection to a dexter database, a matrix, 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 parameters, see details.
`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
`nDraws`	Number of Gibbs samples when estimation method is Bayes.
`merge_within_persons`	whether to merge different booklets administered to the same person, enabling linking over persons as well as booklets.
`nIterations`	deprecated

Details

To support some flexibility in fixing parameters, fixed_params can be a dexter prms object of a data.frame. If a data.frame, it should contain the columns item_id, item_score and a difficulty parameter. Three types of parameters are supported:

delta/beta thresholds between subsequent item categories

eta item-category parameters

b exp(-eta)

Each type corresponds to a different parametrization of the model.

Value

An object of type `prms`. The prms object can be cast to a data.frame of item parameters using function 'coef' 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. 80(4), 859-879.

Koops, J. and Bechger, T.M. and Maris, G. (in press); Bayesian inference for multistage and other incomplete designs. In Research for Practical Issues and Solutions in Computerized Multistage Testing. Routledge, London.

See Also

functions that accept a prms object as input: `ability`, `plausible_values`, `plot.prms`, and `plausible_scores`

`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`	a connection to a dexter database, a matrix, 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 therefore fits the Rasch model and the interaction model on complete data. This typically consist of responses to items in one booklet but can also consist of the intersection (common items) in 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

```
db = start_new_project(verbAggrRules, ":memory:")
add_booklet(db, verbAggrData, "agg")

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

close_project(db)
```

---

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 connection to a dexter database, i.e. the output of `start_new_project` or `open_project`

Value

A data frame 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(
  dataSrc,
  format = c("long", "wide"),
  rows = c("booklet_id", "item_id", "item_position"),
  columns = c("item_id", "booklet_id", "item_position"),
  fill = NA
)
```

Arguments

dataSrc	a dexter database or any object form which a design can be inferred
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, ignored if format='long'

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 available) 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 connection to a dexter database, e.g. 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_persons *Persons in a project*

Description

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

Usage

```
get_persons(db)
```

Arguments

db	a connection to a dexter database, e.g. the output of start_new_project or open_project

Value

A data frame with columns person_id and columns for each person_property

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")
)
```

Arguments

dataSrc	a connection to a dexter database, a matrix, or a data.frame with columns: person_id, item_id, item_score
predicate	an expression to select data on
columns	the columns you wish to select, can include any column in the project, see: get_variables

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_resp_data *Functions for developers*

Description

These functions are meant for people who want to develop their own models based on the data management structure of dexter. Very little input checking is performed, the benefit is some extra speed over using ‘get_responses‘. Regular users are advised not to use these functions as incorrect use can easily crash your R-session or lead to unexpected results.

Usage

```
get_resp_data(
  dataSrc,
  qtpredicate = NULL,
  extra_columns = NULL,
  summarised = FALSE,
  env = NULL,
  protect_x = TRUE,
  retain_person_id = TRUE,
  merge_within_persons = FALSE,
  parms_check = NULL
)

get_resp_matrix(dataSrc, qtpredicate = NULL, env = NULL)
```

Arguments

dataSrc	data.frame, integer matrix, dexter database or 'dx_resp_data' object
qtpredicate	quoted predicate
extra_columns	to be returned in addition to person_id, booklet_id, item_score, item_id
summarised	if TRUE, no item scores are returned, just booklet scores
env	environment for evaluation of qtpredicate, defaults to caller environment
protect_x	best set TRUE (default)
retain_person_id	whether to retain the original person_id levels or just use arbitrary integers
merge_within_persons	merge different booklets for the same person together
parms_check	data.frame of item_id, item_score to check for coverage of data

Value

get_resp_data returns a list with class 'dx_resp_data' with elements

- **x** when summarised is FALSE, a tibble(person_id, booklet_id, item_id, item_score, booklet_score [, extra_columns>]), sorted in such a way that all rows pertaining to the same person-booklet are together
  when summarised is TRUE, a tibble(person_id, booklet_id, booklet_score [, extra_columns])
- **design** tibble(booklet_id, item_id), sorted

get_resp_matrix returns a matrix of item scores as commonly used in other IRT packages, facilitating easy connection of your own package to the data management capabilities of dexter

get_rules *Get scoring rules*

Description

Retrieve the scoring rules currently present in the dexter project db

Usage

```
get_rules(db)
```

Arguments

db	a connection to a Dexter database

Value

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

get_testscores *Provide test scores*

Description

Supplies the sum of item scores for each person selected.

Usage

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

Arguments

dataSrc	a connection to a dexter database, a matrix, 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, booklet_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 properties 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

individual_differences *Test individual differences*

Description

Test individual differences

Usage

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

Arguments

dataSrc	a connection to a dexter database, a matrix, or a data.frame with columns: person_id, item_id, item_score
predicate	An optional expression to subset data, if NULL all data are used.

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 (i.e., 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)
```

---

information *Functions of theta*

Description

returns information function, expected score function, score distribution, or score simulation function for a single item, an arbitrary group of items or all items

Usage

```
information(parms, items = NULL, booklet_id = NULL, which.draw = NULL)

expected_score(parms, items = NULL, booklet_id = NULL, which.draw = NULL)

r_score(parms, items = NULL, booklet_id = NULL, which.draw = NULL)

p_score(parms, items = NULL, booklet_id = NULL, which.draw = NULL)
```

Arguments

parms	object produced by fit_enorm or a data.frame with columns item_id, item_score and, depending on parametrization, a column named either beta/delta, eta or b
items	vector of one or more item_id's. If NULL and booklet_id is also NULL, all items in parms are used
booklet_id	id of a single booklet (e.g. the test information function), if items is not NULL this is ignored
which.draw	the number of the random draw (only applicable if calibration method was Bayes). If NULL, the mean beta parameter will be used

Value

Each function returns a new function which accepts a vector of theta's. These return the following values:

information an equal length vector with the information estimate at each value of theta.

expected_score an equal length vector with the expected score at each value of theta

r_score a matrix with length(theta) rows and one column for each item containing simulated scores based on theta. To obtain test scores, use rowSums on this matrix

p_score a matrix with length(theta) rows and one column for each possible sumscore containing the probability of the score given theta

Examples

```
db = start_new_project(verbAggrRules,':memory:')
add_booklet(db,verbAggrData, "agg")
p = fit_enorm(db)

# plot information function for single item

ifun = information(p, "S1DoScold")

plot(ifun,from=-4,to=4)

# compare test information function to the population ability distribution

ifun = information(p, booklet="agg")

pv = plausible_values(db,p)

op = par(no.readonly=TRUE)
par(mar = c(5,4,2,4))

plot(ifun,from=-4,to=4, xlab='theta', ylab='test information')

par(new=TRUE)

plot(density(pv$PV1), col='green', axes=FALSE, xlab=NA, ylab=NA, main=NA)
```

```
axis(side=4)
mtext(side = 4, line = 2.5, 'population density (green)')

par(op)
close_project(db)
```

---

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 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.

The input data frame must contain the exact id 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 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")
```

Arguments

db_name	The name of the database to be opened.

Value

a database connection object

plausible_scores *Generate plausible test scores*

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,
  parms = NULL,
  predicate = NULL,
  items = NULL,
  covariates = NULL,
  keep.observed = TRUE,
  nPS = 1,
  merge_within_persons = FALSE
)
```

Arguments

dataSrc	a connection to a dexter database, a matrix, 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 scores conditional on the item parameters. These are considered known. If parms is NULL, Bayesian parameters are calculated from the datasrc
predicate	an expression to filter data. If missing, the function will use all data in 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	If 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 testscores to generate per person.
merge_within_persons	If a person took multiple booklets, this indicates whether plausible scores are generated per person (TRUE) or per booklet (FALSE)

Value

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

---

plausible_values *Draw plausible values*

Description

Draws plausible values based on test scores

Usage

```
plausible_values(
  dataSrc,
  parms = NULL,
  predicate = NULL,
  covariates = NULL,
  nPV = 1,
  use_draw = NULL,
  prior.dist = c("normal", "mixture"),
  merge_within_persons = FALSE
)
```

Arguments

dataSrc	a connection to a dexter database, a matrix, or a data.frame with columns: person_id, item_id, item_score
parms	An object returned by function fit_enorm containing parameter estimates. If parms are provided, item parameters are considered known. If parms = NULL, plausible values are marginalized over the posterior distribution of the item parameters and 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 populations used to improve the prior. A covariate must be a discrete person property (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 specifies the use of a particular sample of item parameters used to generate the plausible value(s). If NULL, the posterior means are used. If outside range, the last iteration will be used.
prior.dist	use a normal prior or a mixture of two normals recognised automatically),
merge_within_persons	If a person took multiple booklets, this indicates whether plausible values are generated per person (TRUE) or per booklet (FALSE)

Value

A data.frame with columns booklet_id, person_id, booklet_score 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. See also the vignette.

Examples

```
db = start_new_project(verbAggrRules, ":memory:",
   person_properties=list(gender="<unknown>"))
add_booklet(db, verbAggrData, "agg")
add_item_properties(db, verbAggrProperties)

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

par(mfrow=c(1,2))

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)
```

`plot.DIF_stats` *plot method for pairwise DIF statistics*

Description

plot method for pairwise DIF statistics

Usage

```
## S3 method for class 'DIF_stats'
plot(x, items = NULL, itemsX = items, itemsY = items, alpha = 0.05, ...)
```

Arguments

`x`	object produced by `DIF`
`items`	character vector of item id's for a subset of the plot. Useful if you have many items. If NULL all items are plotted.
`itemsX`	character vector of item id's for the X axis
`itemsY`	character vector of item id's for the Y axis
`alpha`	significance level used to color the plot (two sided)
`...`	further arguments to plot

Details

Plotting produces an image of the matrix of pairwise DIF statistics. The statistics are standard normal deviates and colored to distinguish significant from non-significant values. If there is no DIF, a proportion alpha will be significant be change.

References

Feskens, R., Fox, J. P., & Zwitser, R. (2019). Differential item functioning in PISA due to mode effects. In Theoretical and Practical Advances in Computer-based Educational Measurement (pp. 231-247). Springer, Cham.

plot.p2pass *A plot method for probability_to_pass*

Description

Plot equating information from probability_to_pass

Usage

```
## S3 method for class 'p2pass'
plot(
  x,
  what = c("all", "equating", "sens/spec", "roc"),
  booklet_id = NULL,
  ...
)
```

Arguments

x	An object produced by function probability_to_pass
what	information to plot, 'equating', 'sens/spec', 'roc, or 'all'
booklet_id	vector of booklet_id's to plot, if NULL all booklets are plotted
...	Any additional plotting parameters; e.g., cex = 0.7.

plot.prms *Plot for the extended nominal Response model*

Description

The plot shows 'fit' by comparing the expected score based on the model (grey line) with the average scores based on the data (black line with dots) for groups of students with similar estimated ability.

Usage

```
## S3 method for class 'prms'
plot(
  x,
  item_id = NULL,
  dataSrc = NULL,
  predicate = NULL,
  nbins = 5,
  ci = 0.95,
  ...
)
```

Arguments

x	object produced by fit_enorm
item_id	which item to plot, if NULL, one plot for each item is made
dataSrc	data source, see details
predicate	an expression to subset data in dataSrc
nbins	number of ability groups
ci	confidence interval for the error bars, between 0 and 1. Use 0 to suppress the error bars. Default = 0.95 for a 95% confidence interval
...	further arguments to plot

Details

The standard plot shows the fit against the sample on which the parameters were fitted. If dataSrc is provided, the fit is shown against the observed data in dataSrc. This may be useful for plotting the fit in different subgroups as a visual test for item level DIF. The confidence intervals denote the uncertainty about the predicted pvalues within the ability groups for the sample size in dataSrc (if not NULL) or the original data on which the model was fit.

Value

Silently, a data.frame with observed and expected values possibly useful to create a numerical fit measure.

---

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,
  curtains = 10,
  show.observed = TRUE,
  ...
)
```

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
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

Customization 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')

---

probability_to_pass *The probability to pass on a reference test given a score on a new booklet*

Description

Given response data that form a connected design, compute the probability to pass on the reference set conditional on each score on one or more target tests.

Usage

```
probability_to_pass(
  dataSrc,
  parms,
  ref_items,
  pass_fail,
  predicate = NULL,
  target_booklets = NULL,
  nDraws = 1000
)
```

Arguments

dataSrc	a connection to a dexter database, a matrix, or a data.frame with columns: person_id, item_id, item_score
parms	parameters returned from fit_enorm. If uncertainty about parameter estimation should be included in the computations, use 'method='Bayes'' and nDraws equal or larger than nDraws in probability_to_pass
ref_items	vector with id's of items in the reference set, they must all occur in dataSrc
pass_fail	pass-fail score on the reference set, the lowest score with which one passes
predicate	An optional expression to subset data in dataSrc, if NULL all data is used
target_booklets	The target test booklet(s). A data.frame with columns booklet_id (if multiple booklets) and item_id, if NULL (default) this will be derived from the dataSrc and the probability to pass will be computed for each test score for each booklet in your data.
nDraws	The function uses an Markov-Chain Monte-Carlo method to calculate the probability to pass and this is the number of Monte-Carlo samples used.

Details

Note that this function is computationally intensive and can take some time to run, especially when computing the probability to pass for multiple target booklets. Further technical details can be found in a vignette.

Value

An object of type `p2pass`. Use `coef()` to extract the probablity to pass for each booklet and score. Use `plot()` to plot the probabilities, sensitivity and specificity or a ROC-curve.

See Also

The function used to plot the results: `plot.p2pass`

`profile_plot` *Profile plot*

Description

Profile plot

Usage

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

Arguments

dataSrc	a connection to a dexter database 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 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
col	vector of colors to use for plotting
...	further arguments to plot

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 (meaningful) 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.

Examples

```
db = start_new_project(verbAggrRules, ":memory:",
                       person_properties=list(gender="unknown"))
add_booklet(db, verbAggrData, "agg")
add_item_properties(db, verbAggrProperties)
```

```
profile_plot(db, item_property='mode', covariate='gender')

close_project(db)
```

`profile_tables` *Profile analysis*

Description

Expected and observed domain scores, conditional on the test score, per person or test score. Domains are specified as categories of items using item_properties.

Usage

```
profile_tables(parms, domains, item_property, design = NULL)

profiles(
  dataSrc,
  parms,
  item_property,
  predicate = NULL,
  merge_within_persons = FALSE
)
```

Arguments

`parms`	An object returned by `fit_enorm` or a data.frame of item parameters
`domains`	data.frame with column item_id and a column with name equal to `item_property`
`item_property`	the name of the item property used to define the domains. If `dataSrc` is a dexter db then the item_property must match a known item property. If datasrc is a data.frame, item_property must be equal to one of its column names. For profile_tables item_property must match a column name in `domains`.
`design`	data.frame with columns item_id and optionally booklet_id
`dataSrc`	a connection to a dexter database or a data.frame with columns: person_id, item_id, item_score, an arbitrarily named column containing an item property and optionally booklet_id
`predicate`	An optional expression to subset data in dataSrc, if NULL all data is used
`merge_within_persons`	whether to merge different booklets administered to the same person.

Details

When using a unidimensional IRT Model like the extended nominal response model in dexter (see: `fit_enorm`), the model is as a rule to simple to catch all the relevant dimensions in a test. Nevertheless, a simple model is quite useful in practice. Profile analysis can complement the model in this case by indicating how a test-taker, conditional on her/his test score, performs on a number of pre-specified domains, e.g. in case of a mathematics test the domains could be numbers, algebra and geometry or in case of a digital test the domains could be animated versus non-animated items. This can be done by comparing the achieved score on a domain with the expected score, given the test score.

Value

profiles a data.frame with columns person_id, booklet_id, booklet_score, <item_property>, domain_score, expected_domain_score

profile_tables a data.frame with columns booklet_id, booklet_score, <item_property>, expected_domain_score

References

Verhelst, N. D. (2012). Profile analysis: a closer look at the PISA 2000 reading data. Scandinavian Journal of Educational Research, 56 (3), 315-332.

`ratedData` *Rated data*

Description

A data set with rated data. A number of student performances are rated twice on several aspects by independent judges. The ratings are binary and have been summed following the theory discussed by Maris and Bechger (2006, Handbook of Statistics). Data are a small subset of data collected on the State Exam Dutch as a second language for Speaking.

Format

A data set with 75 rows and 15 columns.

`ratedDataProperties` *Item properties in the rated data*

Description

A data set of item properties related to the rated data. These are the aspects: IH = content, WZ = word choice and phrasing, and WK = vocabulary.

Format

A data set with 14 rows and 2 columns: item_id and aspect

ratedDataRules *Scoring rules for the rated data*

Description

A set of (trivial) scoring rules for the rated data set

Format

A data set with 42 rows and 3 columns (item_id, response, item_score).

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, nbr, for .CML files also several statistics columns that are outputted by OPLM as part of the calibration.

Examples

```
## Not run:
\donttest{
par = read_oplm_par('/parameters.PAR')
f = fit_enorm(db, fixed_params=par)
}
## End(Not run)
```

r_score_IM *Simulation from the interaction model*

Description

Simulate item scores conditional on test scores using the interaction model

Usage

```
r_score_IM(m, scores)
```

Arguments

m	an object produced by function fit_inter
scores	vector of test scores

Value

a matrix with item scores, one column per item and one row per test score. Row order equal to scores

standards_3dc *Standard setting*

Description

Set performance standards on one or more test forms using the data driven direct consensus (3DC) method

Usage

```
standards_3dc(parms, design)

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

## S3 method for class 'sts_par'
plot(x, booklet_id = NULL, ...)
```

Arguments

parms	parameters object returned from fit_enorm
design	a data.frame with columns ‘cluster_id‘, ‘item_id‘ and optionally ‘booklet_id‘
object	an object containing parameters for the 3DC standard setting procedure
...	ignored Optionally you can include a column ‘booklet_id‘ to specify multiple test forms for standard setting and/or columns ‘cluster_nbr‘ and ‘item_nbr‘ to specify ordering of clusters and items in the forms and application.
x	an object containing parameters for the 3DC standard setting procedure
booklet_id	which test form to plot

Details

The data driven direct consensus (3DC) method of standard setting was invented by Gunter Maris and 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. 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.

Although the 3DC method is used as explained in Keuning et. al., the method we use for computing the forms is a simple maximum likelihood scaling from an IRT model, described in Moe and Verhelst (2017)

Value

an object of type ‘sts_par‘

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: Blomeke S., Gustafsson JE. (eds) Standard Setting in Education. Methodology of Educational Measurement and Assessment. Springer, Cham

Moe E., Verhelst N. (2017) Setting Standards for Multistage Tests of Norwegian for Adult Immigrants In: Blomeke S., Gustafsson JE. (eds) Standard Setting in Education. Methodology of Educational Measurement and Assessment. Springer, Cham

See Also

how to make a database for the 3DC standard setting application: standards_db

Examples

```
library(dplyr)
db = start_new_project(verbAggrRules, ":memory:")

add_booklet(db, verbAggrData, "agg")
add_item_properties(db, verbAggrProperties)
```

```
design = get_items(db) %>%
  rename(cluster_id='behavior')

f = fit_enorm(db)

sts_par = standards_3dc(f, design)

plot(sts_par)


# db_sts = standards_db(sts_par,'test.db',c('mildly aggressive','dangerously aggressive'))
```

`standards_db` *Export a standard setting database for use by the free 3DC application*

Description

This function creates an export (an sqlite database file) which can be used by the 3DC application. This is a free application with which a standard setting session can be facilitated through a LAN network using the Chrome browser. The 3DC application can be downloaded from 3DC application download page

Usage

```
standards_db(
  par.sts,
  file_name,
  standards,
  population = NULL,
  group_leader = "admin"
)
```

Arguments

`par.sts`	an object containing parameters for the 3DC standard setting procedure produced by `standards_3dc`
`file_name`	name of the exported database file
`standards`	vector of 1 or more standards. In case there are multiple test forms and they should use different performance standards, a list of such vectors. The names of this list should correspond to the names of the testforms
`population`	optional, a data.frame with three columns: 'booklet_id','booklet_score','n' (where n is a count)
`group_leader`	login name of the group leader. The login password will always be 'admin' but can be changed in the 3DC application

`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_name = "dexter.db", person_properties = 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_name`	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. For an in-memory database you can use the string `":memory:"`.
`person_properties`	An optional list of person properties. Names should correspond to person_properties intended to be used in the project. Values are used as default (missing) values. The datatype will also be inferred from the values. Known person_properties will be automatically imported when adding response data with `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

a database connection object.

Examples

```
head(verbAggrRules)
db_name = tempfile(fileext='.db')
db = start_new_project(verbAggrRules, db_name,
                       person_properties = 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 = NULL,
  responses_start = NULL,
  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 inclusive. If NULL, this is read from the scr file.
responses_start	start position of responses in the .dat file. If NULL, this is read from the scr 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	vector of character(s) 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 predicates in the various dexter functions.

Booklet_position and responses_start are usually inferred from the scr file but since they are sometimes misspecified in the scr file they can be overridden. Response_length is not inferred from the scr file since anything other than 1 is most often a mistake.

Value

a database connection object.

Examples

```
## Not run: \donttest{
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))

prms = fit_enorm(db,
   item_global_on_off==1 & item_local_on_off==1 & booklet_on_off==1)


}
## End(Not run)
```

---

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"),
  max_scores = c("observed", "theoretical")
)
```

Arguments

dataSrc	a connection to a dexter database, a matrix, 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
max_scores	use the observed maximum item score or the theoretical maximum items score according to the scoring rules in the database to compute pvalues and maximum scores

Value

A list containing:

testStats	a data.frame of statistics at test level
itemStats	a data.frame (or list if type='compared') 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	a connection 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

Examples

```
## Not run: \donttest{
# 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)
}
## End(Not run)
```

---

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 26 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