Package ‘dexterMST’

February 14, 2022

Type Package
Title CML and Bayesian Calibration of Multistage Tests
Version 0.9.3
Date 2022-02-14
Maintainer Timo Bechger <tmbechger@gmail.com>
License GPL (>= 2)
URL https://dexter-psychometrics.github.io/dexter/
BugReports https://github.com/dexter-psychometrics/dexter/issues
Encoding UTF-8
Depends R (>= 3.4)
Imports Rcpp, dexter (>= 1.1.4), dplyr, RSQLite, rlang, igraph (>= 1.2.1), tidyr, DBI, crayon, graphics, methods, stats, utils
LinkingTo Rcpp, RcppArmadillo
RoxygenNote 7.1.2
Suggests knitr, rmarkdown, testthat, mirt, ggplot2, Cairo
VignetteBuilder knitr
NeedsCompilation yes
Author Timo Bechger [aut, cre], Jesse Koops [aut], Ivailo Partchev [aut], Gunter Maris [aut], Robert Zwitser [ctb]
Repository CRAN
Date/Publication 2022-02-14 19:40:13 UTC
DexterMST is a generalization of the most important functionality in dexter to multi stage tests. Function names are typically the same as in dexter with `'_mst'` added. CML calibration of real life mst tests is tricky, especially if one considers the need to condition on the design in combination with data selections and corrections of key errors. DexterMST aims to handle these things automatically and protect the user from making mistakes by working from a local database which enforces some restrictions.

The main features are:

- project databases providing a structure for storing data about persons, items, responses and booklets.
- CML calibration of the extended nominal response model and interaction model.

To learn more about dexterMST, start with the vignette: `vignette('dexterMST', package='dexterMST')`
add_item_properties_mst

Description

Add item properties to an dextermst project

Usage

add_item_properties_mst(db, item_properties)

Arguments

db
dexterMST project database

item_properties
data.frame with a column item_id and other columns containing the item properties
add_person_properties_mst

Add person properties to a mst project

Description

Add person properties to a mst project

Usage

add_person_properties_mst(db, person_properties)

Arguments

db  dextermst project database
person_properties  data.frame with a column person_id and other columns containing the person properties

add_response_data_mst

Add multistage response data

Description

Multistage response data can be entered in long format for one or multiple booklets simultaneously or in wide format one booklet at a time.

Usage

add_response_data_mst(db, rsp_data, auto_add_unknown_rules = FALSE)

add_booklet_mst(
  db,
  booklet_data,
  test_id,
  booklet_id,
  auto_add_unknown_rules = FALSE
)
add_scoring_rules_mst

Arguments

- **db**
  a dextermst db handle

- **rsp_data**
  data.frame with columns (person_id, test_id, booklet_id, item_id, response)

- **auto_add_unknown_rules**
  if FALSE, unknown responses (i.e. not defined in the scoring rules) will generate an error and the function will abort. If TRUE unknown responses will be automatically added to the scoring rules with a score of 0

- **booklet_data**
  data.frame with a column person_id and other columns which names correspond to item_id’s

- **test_id**
  id of a test known in the database

- **booklet_id**
  id of a booklet known in the database

Details

Users familiar with dexter might expect to be able to enter new booklets here. Because mst tests have a more complicated design that cannot be (easily) derived from the data, in dexterMST the test designs have to be entered beforehand.

See Also

create_mst_test

Description

add scoring rules to an mst project

Usage

add_scoring_rules_mst(db, rules)

Arguments

- **db**
  a dextermst db connection

- **rules**
  dataframe (item_id, response, item_score), listing all permissible responses to an item and their scores
**alter_scoring_rules_mst**

*alter scoring rules in an mst project*

**Description**

It is only possible to change item_scores for existing items and responses through this function. Scoring rules can only be changed for items that are in the last module of a (mst) test.

**Usage**

```r
alter_scoring_rules_mst(db, rules)
```

**Arguments**

- `db` : a dexter_mst db connection
- `rules` : data.frame (item_id, response, item_score), see dexter

**close_mst_project**

*Close an mst project*

**Description**

Close an mst project

**Usage**

```r
close_mst_project(db)
```

**Arguments**

- `db` : dexter_mst project db connection
**create_mst_project**

create a new (empty) mst project

**Description**

create a new (empty) mst project

**Usage**

```r
create_mst_project(pth)
```

**Arguments**

- **pth**
  path and filename to save project file

**Value**

handle to project database

---

**create_mst_test**

Define a new multi stage test

**Description**

Before you can enter data, dexterMST needs to know the design of your test.

**Usage**

```r
create_mst_test(
  db,
  test_design,
  routing_rules,
  test_id,
  routing = c("all", "last")
)
```

**Arguments**

- **db**
  output of `open_mst_project` or `create_mst_project`
- **test_design**
  data.frame with columns `item_id`, `module_id`, `item_position`
- **routing_rules**
  output of `mst_rules`
- **test_id**
  id of the mst test
- **routing**
  all or last routing (see details)
Details

In dexterMST we use the following terminology:

- **test**: collection of modules and rules to go from one module to the other. A test must have one starting module.
- **booklet**: a specific path through a mst test.
- **module**: a block of items that is always administered together. Each item has a specific position in a module.
- **routing rules**: rules to go from one module to another based on score on the current and possibly previous modules.

Additionally, there are two possible types of routing:

- **all**: the routing rules are based on the sum of the current and previous modules.
- **last**: the routing rules are based only on the current module.

The type of routing must be defined for a test as a whole so it is not possible to mix routing types. In CML (as opposed to MML) the routing rules are actually used in the calibration so it is important they are correctly specified. DexterMST includes multiple checks, both when defining the test and when entering data, to make sure your routing rules are valid and your data conform to them.

Examples

```r
# extended example
# we:
# 1) define an mst design
# 2) simulate mst data
# 3) create a project, enter scoring rules and define the MST test
# 4) do an analysis

library(dplyr)

items = data.frame(item_id=sprintf("item%02d",1:70), item_score=1, delta=sort(runif(70,-1,1)))
design = data.frame(item_id=sprintf("item%02d",1:70),
                    module_id=rep(c("M4","M2","M5","M1","M6","M3","M7"),each=10))

routing_rules = mst_rules(
  `124` = M1[0:5] ---+ M2[0:10] ---+ M4,
theta = rnorm(3000)
dat = sim_mst(items, theta, design, routing_rules,'all')
dat$test_id='sim_test'
dat$response=dat$item_score

scoring_rules = data.frame(
```
```r
item_id = rep(items$item_id,2),
item_score= rep(0:1,each=nrow(items)),
response= rep(0:1,each=nrow(items))) # dummy response

db = create_mst_project(":memory:"
add_scoring_rules_mst(db, scoring_rules)

create_mst_test(db,
    test_design = design,
    routing_rules = routing_rules,
    test_id = 'sim_test',
    routing = "all"
)

add_response_data_mst(db, dat)

design_plot(db)

f = fit_enorm_mst(db)

head(coef(f))

abl = ability(get_responses_mst(db), f) %>%
    inner_join(tibble(person_id=as.character(1:3000), theta.sim=theta), by='person_id')

plot(abl$theta, abl$theta.sim)

abl = filter(abl, is.finite(theta))

cor(abl$theta, abl$theta.sim)
```

---

**design_plot**

*Plot the routing design of mst tests*

**Description**

Plot the routing design of mst tests

**Usage**

```r
design_plot(db, predicate = NULL, by_booklet = FALSE, ...)
```

**Arguments**

- `db`: dexterMST project database connection
- `predicate`: logical predicate to select data (tests, booklets, responses) to include in the design plot
- `by_booklet`: plot and color the paths in a test per booklet
- `...`: further arguments to plot.igraph
Details

You can use this function to plot routing designs for tests before or after they are administered. There are some slight differences.

If you have entered response data already, the thickness of the line will indicate the numbers of respondents that took the respective paths through the test. Paths not taken will not be drawn. You can use the predicate (see examples) to include or exclude items, tests and respondents.

If you have not entered response data, all lines will have equal thickness. Variables you can use in the predicate are limited to test_id and booklet_id in this case.

Examples

```r
# Not run:
# plot test designs for all tests in the project
design_plot(db)

# plot design for a test with id 'math'
design_plot(db, test_id == 'math')

# plot design for test math with item 'circumference' turned off
# (this plot will only work if you have response data)
design_plot(db, test_id == 'math' & item_id != 'circumference')
```

## End(Not run)

---

**DIF_mst**

*Exploratory test for Differential Item Functioning*

**Description**

Compares two parameter objects and produces a test for DIF based on equality of relative item difficulties category locations

**Usage**

```
DIF_mst(db, person_property, predicate = NULL)
```

**Arguments**

- `db`: an dexterMST db handle
- `person_property`: name of a person property defined in your dexterMST project
- `predicate`: logical predicate to select data to include in the analysis
References


Examples

```r
## Not run:

dif = DIF_mst(db, person_property = 'test_mode')
print(dif)
plot(dif)

## End(Not run)
```

fit_enorm_mst

**Fit the extended nominal response model on MST data**

Description

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

Usage

```r
fit_enorm_mst(
  db, 
  predicate = NULL, 
  fixed_parameters = NULL, 
  method = c("CML", "Bayes"), 
  nDraws = 1000 
)
```

Arguments

- `db`: an dextermst db handle
- `predicate`: logical predicate to select data to include in the analysis, see details
- `fixed_parameters`: data.frame with columns ‘item_id’, ‘item_score’ and ‘beta’
- `method`: If CML, the estimation method will be Conditional Maximum Likelihood. If Bayes, a Gibbs sampler will be used to produce a sample from the posterior.
- `nDraws`: Number of Gibbs samples when estimation method is Bayes.
Details

You can use the predicate to include or omit responses from the analysis, e.g. `p = fit_enorm_mst(db, item_id != 'some_item' & student_birthdate > '2005-01-01')`.

DexterMST will automatically correct the routing rules for the purpose of the current analysis. There are some caveats though. Predicates that lead to many different designs, e.g. a predicate like `response != 'NA'` (which is perfectly valid but can potentially create almost as many tests as there are students) might take very long to compute.

Predicates that remove complete modules from a test, e.g. `module_nbr !=2 or module_id != 'RU4'` will cause an error and should be avoided.

Value

object of type 'mst_enorm’. Can be cast to a data.frame of item parameters using function ‘coef’ or used in dexter’s ability functions

References


---

**fit_inter_mst**  
Fit the interaction model on a single multi-stage booklet

Description

Fit the interaction model on a single multi-stage booklet

Usage

`fit_inter_mst(db, test_id, booklet_id)`

Arguments

db  
a db handle

test_id  
id of the test as defined in `create_mst_test`

booklet_id  
id of the booklet as defined in `create_mst_test`
**get_booklets_mst**

Retrieve information from a MST database

**Description**

Retrieve information from a MST database

**Usage**

```r
get_booklets_mst(db)
get_design_mst(db)
get_routing_rules_mst(db)
get_scoring_rules_mst(db)
get_items_mst(db)
get_persons_mst(db)
```

**Arguments**

- `db` dexterMST project database connection

**get_responses_mst**

Extract response data from a dexterMST database

**Description**

Extract response data from a dexterMST database

**Usage**

```r
get_responses_mst(db, predicate = NULL, columns = c("person_id", "test_id", "booklet_id", "item_id", "item_score"))
```

**Arguments**

- `db` a dexterMST project database connection
- `predicate` an expression to select data on
- `columns` the columns you wish to select, can include any column in the project
Value

a data.frame of responses

**Description**

This function will import items, scoring rules, persons, test designs and responses from a dexter database into the dexterMST database.

**Usage**

```r
import_from_dexter(db, dexter_db, dx_response_prefix = "")
```

**Arguments**

- `db` - dextermst project db connection
- `dexter_db` - path to a dexter database file or open dexter db connection
- `dx_response_prefix` - string to prefix responses from dexter with (usually not necessary, see details)

**Details**

DexterMST has no problem calibrating data from linear tests. However, dexter and dexterMST have differently structured project databases. If you already have response data from linear tests in a dexter database, you can easily import it into your dexterMST database from there.

The dexterMST variables test_id, module_id and booklet_id will all be set to the dexter variable booklet_id (i.e. a linear test becomes a multistage test with one booklet and one module only).

It is assumed that items with equal id’s in your dexter and dexterMST project refer to the same items. If an item in dexter has different score categories compared to an existing item with the same item_id in dexterMST an error will be generated. If the same response to the same item has a different score, this will also generate an error. However, it is possible for an item in dexter to have scoring rules for responses not defined in dexterMST and vice versa.

In the unusual and unfortunate situation that the same response to the same item should have a different score in dexter than in dexterMST, you can use the parameter `dx_response_prefix` to prefix the responses in dexter with some unique combination of characters, e.g. "dexter". In practice this sometimes happens when old archived data is only available in scored form (i.e. response 0 has score 0, response 1 has score 1) and new data is available in raw form but the actual response can also be 0 or 1, etc. causing a conflict.
Examples

```r
## Not run:
library(dexter)
dbDex = start_new_project(verbAggrRules, "verbAggression.db",
    person_properties=list(gender="unknown"))
add_booklet(dbDex, verbAggrData, "agg")
add_item_properties(dbDex, verbAggrProperties)
db = create_mst_project("::memory::")
import_from_dexter(db, dbDex)
f_mst = fit_enorm_mst(db)
f_dexter = fit_enorm(dbDex)
close_mst_project(db)
close_project(dbDex)

## End(Not run)
```

---

**mst_rules**  
Define routing rules

**Description**
Define routing rules for use in `create_mst_test`

**Usage**
mst_rules(...)  

**Arguments**
...

routing rules defined using a a dot-like syntax, read → as an arrow and [:] as a range of score to move to the next stage

**Details**
Each scoring rule in ‘...’ defines one or more routing rules together making up a booklet. For example, ‘route1 = a[0:5] → d[9:15] → f’ means a start at module ‘a’, continue to module ‘d’ when the score on ‘a’ is between 0 and 5 (inclusive) and continue to ‘g’ when the score on modules ‘a + b’ is between 0 and 8 (for ‘All’ routing) or the score on just module ‘b’ is between 0 and 8 (for ‘Last’ routing). ‘route1’ becomes the id of the specific path or booklet, which must be supplied with the data later.

A routing design for a linear (non-multistage) booklet can simply be entered as `mst_rules(my_booklet = my_single_module)`.

**Value**
data.frame with columns...
See Also

create_mst_test for a description of all and last routing and add_response_data_mst to see how to enter data

Examples

# a (complicated) three stage (1-3-3) routing design with 9 booklets and 7 modules

```r
routing_rules = mst_rules(bk1 = M1[0:61] --+ M2[0:136] --+ M5,
                         bk2 = M1[0:61] --+ M2[137:183] --+ M6,
                         bk3 = M1[0:61] --+ M2[184:Inf] --+ M7,
                         bk4 = M1[62:86] --+ M3[0:98] --+ M5,
                         bk5 = M1[62:86] --+ M3[99:149] --+ M6,
                         bk6 = M1[62:86] --+ M3[150:Inf] --+ M7,
                         bk7 = M1[87:Inf] --+ M4[0:98] --+ M5,
                         bk8 = M1[87:Inf] --+ M4[99:130] --+ M6,
```

---

open_mst_project open an existing mst project

Description

open an existing mst project

Usage

open_mst_project(pth)

Arguments

pth path to project file

---

plot.DIF_stats_mst plot method for DIF_mst

Description

plot method for DIF_mst

Usage

```r
## S3 method for class 'DIF_stats_mst'
plot(x, items = NULL, itemsX = items, itemsY = items, ...)
```
Arguments

- **x**: object produced by DIF_mst
- **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
- **...**: further arguments to plot

Description

plots for the interaction model

Usage

```r
# S3 method for class 'im_mst'
plot(x, item_id = NULL, show.observed = TRUE, curtains = 10, zoom = FALSE, ...)
```

Arguments

- **x**: output of `fit_inter_mst`
- **item_id**: id of the item to plot
- **show.observed**: plot the observed mean item scores for each test score
- **curtains**: percentage of most extreme values to cover with curtains, 0 to omit curtains
- **zoom**: if TRUE, limits the plot area to the test score range allowed by the routing rules
- **...**: further arguments to plot

Description

Expected and observed domain scores per booklet and test score

Usage

`profile_tables_mst(parms, domains, item_property)`
Arguments

- **parms**: An object returned by `fit_enorm_mst`
- **domains**: data.frame with column `item_id` and a column whose name matches ‘item_property’
- **item_property**: the name of the item property used to define the domains.

Value

- a data.frame with expected score per domain, booklet and booklet_score

---

**sim_mst**

*Simulate multistage testing data*

---

Description

Simulates data from an extended nominal response model according to an mst design

Usage

```r
sim_mst(pars, theta, test_design, routing_rules, routing = c("last", "all"))
```

Arguments

- **pars**: item parameters, can be either: a data.frame with columns `item_id`, `item_score`, `beta` or a dexter or dexterMST parameters object
- **theta**: vector of person abilities
- **test_design**: data.frame with columns `item_id`, `module_id`
- **routing_rules**: output of `mst_rules`
- **routing**: 'all' or 'last' routing
Index

ability, 12
add_booklet_mst
   (add_response_data_mst), 4
add_item_properties_mst, 3
add_person_properties_mst, 4
add_response_data_mst, 4, 16
add_scoring_rules_mst, 5
alter_scoring_rules_mst, 6
close_mst_project, 6
create_mst_project, 7, 7
create_mst_test, 5, 7, 12, 15, 16
design_plot, 9
dexterMST (dexterMST-package), 2
dexterMST-package, 2
DIF_mst, 10
fit_enorm_mst, 11, 18
fit_inter_mst, 12, 17
get_booklets_mst, 13
get_design_mst (get_booklets_mst), 13
get_items_mst (get_booklets_mst), 13
get_persons_mst (get_booklets_mst), 13
get_responses_mst, 13
get_routing_rules_mst
   (get_booklets_mst), 13
get_scoring_rules_mst
   (get_booklets_mst), 13
import_from_dexter, 14
mst_rules, 7, 15, 18
open_mst_project, 7, 16
plot.DIF_stats_mst, 16
plot.igraph, 9
plot.im_mst, 17
profile_tables_mst, 17
sim_mst, 18