Package 'MPTmultiverse'

March 12, 2019

Title Multiverse Analysis of Multinomial Processing Tree Models

Version 0.2-0

Description Statistical or cognitive modeling usually requires a number of more or less arbitrary choices creating one specific path through a 'garden of forking paths'. The multiverse approach (Steegen, Tuerlinckx, Gelman, & Vanpaemel, 2016, <doi:10.1177/1745691616658637>) offers a principled alternative in which results for all possible combinations of reasonable modeling choices are reported. MPTmultiverse performs a multiverse analysis for multinomial processing tree (MPT, Riefer & Batchelder, 1988, <doi:10.1037/0033-295X.95.3.318>) models combining maximum-likelihood/frequentist and Bayesian estimation approaches with different levels of pooling (i.e., data aggregation). For the frequentist approaches, no pooling (with and without parametric or nonparametric bootstrap) and complete pooling are implemented using MPTinR <https://cran.r-project.org/package=MPTinR>. For the Bayesian approaches, no pooling, complete pooling, and three different variants of partial pooling are implemented using TreeBUGS <https://cran.r-project.org/package=TreeBUGS>. The main function is fit_mpt() who performs the multiverse analysis in one call.

URL https://github.com/mpt-network/MPTmultiverse

BugReports https://github.com/mpt-network/MPTmultiverse/issues

Depends R (>= 2.11.1),

Imports parallel, magrittr, tidyr, dplyr, tibble, rlang, reshape2, ggplot2, MPTinR, TreeBUGS, runjags, coda, purrr, readr, limSolve

Suggests knitr, rmarkdown, testthat

LazyData yes

VignetteBuilder knitr

RoxygenNote 6.1.1

License GPL-2

NeedsCompilation no

Author Henrik Singmann [aut, cre] (<https://orcid.org/0000-0002-4842-3657>),
Daniel W. Heck [aut],
Marius Barth [aut],
Frederik Aust [ctb] (<https://orcid.org/0000-0003-4900-788X>)

Maintainer Henrik Singmann <singmann@gmail.com>

Repository CRAN

Date/Publication 2019-03-11 23:52:42 UTC

R topics documented:

`check_results` *Check results from a multiverse analysis*

Description

This is a helper function to see if the model estimation worked as intended.

Usage

```
check_results(results)
```

Arguments

`results`	An object of class multiverseMPT.

`fit_mpt` *Multiverse Analysis for MPT Models*

Description

Performs a multiverse analysis for multinomial processing tree (MPT) models across different levels of pooling (i.e., data aggregation) and across maximum-likelihood/frequentist and Bayesian estimation approaches. For the frequentist approaches, no pooling (with and without parametric or nonparametric bootstrap) and complete pooling are implemented using **MPTinR**. For the Bayesian approaches, no pooling, complete pooling, and three different variants of partial pooling are implemented using **TreeBUGS**. Requires `data` on a by-participant level with each row corresponding to data from one participant (i.e., different response categories correspond to different columns) and the data can contain a single between-subjects condition. Model equations need to be passed as a `.eqn` model file and category labels (first column in `.eqn` file) need to match the column names in `data`. Results are returned in one `tibble` with one row per estimation method.

Usage

```
fit_mpt(model, dataset, data, id = NULL, condition = NULL,
  core = NULL, method)
```

Arguments

`model`	A model definition, typically the path to an `.eqn` model file containing the model equations. Category names need to match column names in `data`.
`dataset`	scalar `character` vector. Name of the data set that will be copied to the results `tibble`.
`data`	A `data.frame` containing the data. Column names need to match category names in `model` (i.e., different from **MPTinR** behavior, order of categories is not important, matching is done via name).
`id`	scalar `character` vector. Name of the column that contains the subject identifier. If not specified, it is assumed that each row represents observations from one participant.
`condition`	scalar `character` vector. Name of the column specifying a between-subjects factor. If not specified, no between-subjects comparisons are performed.
`core`	`character` vector defining the core parameters of interest, e.g., `core = c("Dn", "Do")`. All other parameters are treated as auxiliary parameters.
`method`	`character` vector specifying which analysis approaches should be performed (see Description below). Defaults to all available methods.

Details

This functions is a fancy wrapper for packages **MPTinR** and **TreeBUGS** applying various frequentist and Bayesian estimation methods to the same data set with different levels of pooling/aggregation using a single MPT model and collecting the results in one `tibble` where each row corresponds to

one estimation method. Note that parameter restrictions (e.g., equating different parameters or fixing them to a constant) need to be part of the model (i.e., the `.eqn` file) and cannot be passed as an argument.

The settings for the various methods are specified via function `mpt_options`. The default settings use all available cores for calculating the boostrap distribution as well as independent MCMC chains and should be appropriate for most situations.

The data can have a single between-subjects condition (specified via `condition`). This condition can have more than two levels. If specified, the pairwise differences between each level, the standard error of the differences, and confidence-intervals of the differences are calculated for each parameter. Please note that `condition` is silently converted to `character` in the output. Thus, a specific ordering of the `factor` levels in the output cannot be guaranteed. If the data has more than one between-subjects condition, these need to be combined into one condition for this function.

Parameter differences or other support for within-subject conditions is not provided. The best course of action for within-subjects conditions is to simply include separate trees and separate sets of parameters for each within-subjects condition. This allows to at least compare the estimates for each within-subjects condition across levels of pooling and estimation methods.

Pooling: The following pooling levels are provided (not all by all estimation approaches, see below).

- *Complete pooling:* The traditional analysis approach in the MPT literature in which data is aggregated across participants within each between-subjects condition. This approach assumes that there are no individual-dfferences. Produces one set of model parameters per condition.
- *No pooling:* The model is fitted to the individual-level data in an independent manner (i.e., no data aggregation). This approach assumes that there is no similarity across participants and usually requires considerable amounts of data on the individual-level. Produces one set of model parameters per participant. Group-level estimates are based on averaging the individual-level estimates.
- *Partial pooling:* Data is fitted simultaneously to the individual-level data assuming that the individual-level parameters come from a group-level distribution. Individual-level parameters are often treated as random-effects which are nested in the group-level parameters, which is why this approach is also called hierarchical modeling. This approach assumes both individual-level differences and similarities. Produces one set of model parameters per participant plus one set of group-level parameters. Thus, although partial pooling models usually have more parameters than the no-pooling approaches, they are usually less flexible as the hierarchical-structure provides regularization of the individual-level parameters.

Implemented Estimation Methods: Maximum-likelihood estimation with **MPTinR** via `fit.mpt`:

- `"asymptotic_complete"`: Asymptotic ML theory, complete pooling
- `"asymptotic_no"`: Asymptotic ML theory, no pooling
- `"pb_no"`: Parametric bootstrap, no pooling
- `"npb_no"`: Nonparametric bootstrap, no pooling

Bayesian estimation with **TreeBUGS**

- `"simple"`: Bayesian estimation, no pooling (C++, simpleMPT)
- `"simple_pooling"`: Bayesian estimation, complete pooling (C++, simpleMPT)
- `"trait"`: latent-trait model, partial pooling (JAGS, traitMPT)

- `"trait_uncorrelated"`: latent-trait model without correlation parameters, partial pooling (JAGS, traitMPT)
- `"beta"`: beta-MPT model, partial pooling (JAGS, betaMPT)
- `"betacpp"`: beta-MPT model, partial pooling (C++, betaMPTcpp)

Frequentist/Maximum-Likelihood Methods: For the *complete pooling asymptotic approach*, the group-level parameter estimates and goodness-of-fit statistics are the maximum-likelihood and G-squared values returned by `MPTinR`. The parameter differences are based on these values, the standard errors of the difference is simply the pooled standard error of the individual parameters. The overall fit (column `gof`) is based on an additional fit to the completely aggregated data.

For the *no pooling asymptotic approach*, the individual-level maximum-likelihood estimates are reported in column `est_indiv` and `gof_indiv` and provide the basis for the other results. Whether or not an individual-level parameter estimate is judged as identifiable (column `identifiable`) is based on separate fits with different random starting values. If, in these separate, fits the same objective criterion is reached several times (i.e., `Log.Likelihood` within .01 of best fit), but the parameter estimate differs (i.e., different estimates within .01 of each other), then an estimate is flagged as non-identifiable. If they are the same (i.e., within .01 of each other) they are marked as identifiable. The group-level parameters are simply the means of the identifiable individual-level parameters, the SE is the SE of the mean for these parameter (i.e., SD/sqrt(N), where N excludes non-identifiable parameters and thise estimated as NA), and the CI is based on mean and SE. The group-level and overall fit is the sum of the individual G-squares, sum of individual-level df, and corresponding chi-square df. The difference between the conditions and corresponding statistics are based on a t-test comparing the individual-level estimates (again, after excluding non-identifiable estimates). The CIs of the difference are based on the SEs (which are derived from a linear model equivalent to the t-test).

The individual-level estimates of the `bootstrap based no-pooling` approaches are identical to the asymptotic ones. However, the SE is the SD of the bootstrapped distribution of parameter estimates, the CIs are the corresponding quantiles of the bootstrapped distribution, and the p-value is obtained from the bootstrapped G-square distribution. Identifiability of individual-level parameter estimates is also based on the bootstrap distribution of estimates. Specifically, we calculate the range of the CI (i.e., maximum minus minimum CI value) and flag those parameters as non-identifiable for which the range is larger than `mpt_options()$max_ci_indiv`, which defaults to `0.99`. Thus, in the default settings we say a parameter is non-identifiable if the bootstrap based CI extends from 0 to 1. The group-level estimates are the mean of the identifiable individual-level estimates. And difference between conditions is calculated in the same manner as for the asymptotic case using the identifiable individual-level parameter esatimates.

Bayesian Methods: The *simple approaches* fit fixed-effects MPT models. `"simple"` uses no pooling and thus assumes independent uniform priors for the individual-level parameters. Group-level means are obtained as generated quantities by averaging the posterior samples across participants. `"simple_pooling"` aggregates observed frequencies across participants and assumes a uniform prior for the group-level parameters.

The *latent-trait approaches* transform the individual-level parameters to a latent probit scale using the inverse cumulative standard normal distribution. For these probit values, a multivariate normal distribution is assumed at the group level. Whereas `"trait"` estimates the corresponding correlation matrix of the parameters (reported in the column `est_rho`), `"trait_uncorrelated"` does not estimate this correlation matrix (i.e., parameters can still be correlated across individuals, but this is not accounted for in the model).

For all Bayesian methods, the posterior distribution of the parameters is summarized by the posterior mean (in the column `est`), posterior standard deviation (`se`), and credbility intervals (`ci_*`). For parameter differences (`test_between`) and correlations (`est_rho`), Bayesian p-values are computed (column `p`) by counting the relative proportion of posterior samples that are smaller than zero. Goodness of fit is tested with the T1 statistic (observed vs. posterior-predicted average frequencies, `focus = "mean"`) and the T2 statistic (observed vs. posterior-predicted covariance of frequencies, `focus = "cov"`).

Value

A `tibble` with one row per estimation `method` and the following columns:

1. `model`: Name of model file (copied from `model` argument), `character`
2. `dataset`: Name of data set (copied from `dataset` argument), `character`
3. `pooling`: `character` specifying the level of pooling with three potential values: `c("complete", "no", "partial")`
4. `package`: `character` specifying the package used for estimation with two potential values: `c("MPTinR", "TreeBUGS")`
5. `method`: `character` specifying the method used with the following potential values: `c("asymptotic", "PB/MLE", "NI`
6. `est_group`: Group-level parameter estimates per condition/group.
7. `est_indiv`: Individual-level parameter estimates (if provided by method).
8. `est_rho`: Estimated correlation of individual-level parameters on the probit scale (only in `method="trait"`).
9. `test_between`: Parameter differences between the levels of the between-subjects condition (if specified).
10. `gof`: Overall goodness of fit across all individuals.
11. `gof_group`: Group-level goodness of fit.
12. `gof_indiv`: Individual-level goodness of fit.
13. `fungibility`: Posterior correlation of the group-level means `pnorm(mu)` (only in `method="trait"`).
14. `test_homogeneity`: Chi-square based test of participant homogeneity proposed by Smith and Batchelder (2008). This test is the same for each estimation method.
15. `convergence`: Convergence information provided by the respective estimation method. For the asymptotic frequentist methods this is a `tibble` with rank of the Fisher matrix, the number of parameters (which should match the rank of the Fisgher matrix), and the convergence code provided by the optimization algorithm (which is `nlminb`). The boostrap methods contain an additional column, `parameter`, that contains the information which (if any) parameters are empirically non-identifiable based on the bootstrapped distribution of parameter estimates (see above for exact description). For the Bayesian methods this is a `tibble` containing information of the posterior dsitribution (i.e., mean, quantiles, SD, SE, `n.eff`, and R-hat) for each parameter.
16. `estimation`: Time it took for each estimation method and group.
17. `options`: Options used for estimation. Obtained by running `mpt_options()`

With the exception of the first five columns (i.e., after `method`) all columns are `list` columns typically holding one `tibble` per cell. The simplest way to analyze the results is separately per column using `unnest`. Examples for this are given below.

References

Smith, J. B., & Batchelder, W. H. (2008). Assessing individual differences in categorical data. *Psychonomic Bulletin & Review*, 15(4), 713-731. https://doi.org/10.3758/PBR.15.4.713

Examples

```
# ------------------------------------------------------------------------------
# MPT model definition & Data

EQN_FILE <- system.file("extdata", "prospective_memory.eqn", package = "MPTmultiverse")
DATA_FILE <- system.file("extdata", "smith_et_al_2011.csv", package = "MPTmultiverse")

### if .csv format uses semicolons ";" (e.g., German format):
# data <- read.csv2(DATA_FILE, fileEncoding = "UTF-8-BOM")
### if .csv format uses commata "," (international format):
data <- read.csv(DATA_FILE, fileEncoding = "UTF-8-BOM")
data <- data[c(1:10, 113:122),]  ## select only subset of data for example
head(data)

COL_CONDITION <- "WM_EX"  # name of the variable encoding group membership

# experimental condition should be labeled meaningfully ----
unique(data[[COL_CONDITION]])

data[[COL_CONDITION]] <- factor(
  data[[COL_CONDITION]]
  , levels = 1:2
  , labels = c("low_WM", "high_WM")
)

# define core parameters:
CORE <- c("C1", "C2")

## Not run:
op <- mpt_options()
## to reset default options (which you would want) use:
mpt_options("default")

mpt_options() # to see the settings
## Note: settings are also saved in the results tibble

## without specifying method, all are used per default
fit_all <- fit_mpt(
  model = EQN_FILE
  , dataset = DATA_FILE
  , data = data
  , condition = COL_CONDITION
  , core = CORE
)

mpt_options(op) ## reset options
```

```
## End(Not run)

load(system.file("extdata", "prospective_memory_example.rda", package = "MPTmultiverse"))

# Although we requested all 10 methods, only 9 worked:
fit_all$method
# Jags variant of beta MPT is missing.

# the returned method has a plot method. For example, for the group-level estimates:
plot(fit_all, which = "est")

## Not run:
### Full analysis of results requires dplyr and tidyr (or just 'tidyverse')
library("dplyr")
library("tidyr")

## first few columns identify model, data, and estimation approach/method
## remaining columns are list columns containing the results for each method
## use unnest to work with each of the results columns
glimpse(fit_all)

## Let us inspect the group-level estimates
fit_all %>%
  select(method, pooling, est_group) %>%
  unnest()

## which we can plot again
plot(fit_all, which = "est")

## Next we take a look at the GoF
fit_all %>%
  select(method, pooling, gof_group) %>%
  unnest() %>%
  as.data.frame()

# Again, we can plot it as well
plot(fit_all, which = "gof2")  ## use "gof1" for overall GoF

## Finally, we take a look at the differences between conditions
fit_all %>%
  select(method, pooling, test_between) %>%
  unnest()

# and then we plot it
plot(fit_all, which = "test_between")


### Also possible to only use individual methods:
only_asymptotic <- fit_mpt(
  model = EQN_FILE
  , dataset = DATA_FILE
  , data = data
```

```
  , condition = COL_CONDITION
  , core = CORE
  , method = "asymptotic_no"
)
only_asymptotic$est_group

bayes_complete <- fit_mpt(
  model = EQN_FILE
  , dataset = DATA_FILE
  , data = data
  , condition = COL_CONDITION
  , core = CORE
  , method = "simple_pooling"
)
bayes_complete$est_group


## End(Not run)
```

`mpt_options` *Options Settings for MPT Comparison*

Description

Set and examine a variety of *options* which affect the way MPT models are estimated.

Usage

```
mpt_options(...)
```

Arguments

`...`	Named parameters to set. Possible values are:

- `bootstrap_samples`: Numeric. The number of bootstrap samples to be drawn for the calculation parametric bootstrap confidence intervals.
- `n.optim`: Numeric. The number of optimization runs for the models estimated with maximum-likelihood methods.
- `n.chains`: Numeric. The number of MCMC chains for the Bayesian models.
- `n.adapt`: Numeric. The number of iterations for adaptation.
- `n.burnin`: Numeric. The number of burn-in/warm-up iterations.
- `n.iter`: Numeric. The total number of iterations to be drawn *after* adaptation (including burnin).
- `n.thin`: Numeric. Thinning interval.
- `Rhat_max`: Numeric. The maximum rhat.
- `Neff_min`: Numeric. The minimum number of effective samples you are willing to accept.

- extend_max: Numeric.
- n.PPP: Numeric. The number of posterior predictive samples drawn for the calculation of fit statistics T_1 and T_2.
- n.CPU: Numeric. The number of CPU cores to use for obtaining the parametric bootstrap dsitribution. Defaults to the number of available cores on your machine.
- ci_size: Numeric.
- max_ci_indiv: Numeric. Used for excluding individual parameter estimates in the bootstrap approaches. If the range of the CI (i.e., distance between minimum and maximum) is larger than this value, the estimate is excluded from the group-level estimates.
- silent_jags: Logical. Whether to suppress JAGS output.
- save_models: Logical. Default is FALSE which does not save the individual MCMC samples in .RData files. Instead only summairzes are retained in results object.

Examples

```
# Examine options:
mpt_options()

# Set number of MCMC chains to 20:
mpt_options(n.chains = 20)
mpt_options()
```

---

plot.multiverseMPT *Plot multiverseMPT*

Description

Plot the results from a multiverse MPT analysis.

Usage

```
## S3 method for class 'multiverseMPT'
plot(x, which = "est", save = FALSE, ...)
```

Arguments

x	An object of class multiverseMPT.
which	Character. Which information should be plotted? Possible values are "est" for parameter estimates, "test_between" for between-subjects comparisions, "gof1" for overall goodness-of-fit statistics, and "gof2" for group-wise goodness-of-fit statistics.
save	Logical. Indicates whether the plot should also be saved as a .pdf file.
...	ignored.

write_check_results *Write check_results*

Description

Helper function to write the output from check_results() to a file.

Usage

```
write_check_results(DATA_FILE, results)
```

Arguments

DATA_FILE	Character. File name to use.
results	An object of class multiverseMPT.

write_results *Write Results of Multiverse Analysis to csv-Files*

Description

Exports the results to csv format.

Usage

```
write_results(results, path = "MPTmultiverse_")
```

Arguments

results	An object of class multiverseMPT.
path	a path where to save the files (e.g., "C:/results/modelX_dataY_")

Index