Package 'ria.test'

September 15, 2026

Type Package

Title Testing Equality of Natural Mediation Effects and Their Randomized Interventional Analogues

Version 0.3.0

Maintainer Ang Yu <ang_yu@outlook.com>

Description Implements the empirical test introduced by Yu, Ge, and Elwert (2026) for detecting when randomized interventional analogues should not be interpreted as natural mediation effects. The package estimates natural effects, their randomized interventional analogues, and TE - TE^R, the difference between the total effect and its randomized interventional analogue. Rejecting TE - TE^R = 0 falsifies the composite null that the natural indirect and direct effects equal their randomized interventional analogues. The procedure remains valid in settings where the natural effects themselves are not identified, and |TE - TE^R| provides a lower bound on the total divergence between the natural and randomized interventional decompositions.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.2.0)

URL https://github.com/ang-yu/ria.test

BugReports https://github.com/ang-yu/ria.test/issues

Imports checkmate, Matrix, origami, torch, Rsymphony, purrr, cli, S7, data.table, coro, generics, mlr3superlearner, progressr, ife (>= 0.2.1)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Ang Yu [aut, cre, cph] (Author and copyright holder of ria.test-specific extensions), Nicholas Williams [aut, cph] (ORCID: <https://orcid.org/0000-0002-1378-4831>, Author and copyright

holder of code derived from crumble),
Richard Liu [ctb] (Contributor to code derived from crumble),
Iván Díaz [aut, cph] (ORCID: <https://orcid.org/0000-0001-9056-2047>,
Author and copyright holder of code derived from crumble)

Repository CRAN

Date/Publication 2026-09-15 00:20:02 UTC

Contents

`ria.test` *Estimate randomized interventional analogues and their falsification test*

Description

Estimate the total effect, its randomized interventional analogue (RIA), their difference, and the randomized interventional indirect and direct effects. $TE - TE^R$ is the test statistic for the composite null $NIE = NIE^R$ and $NDE = NDE^R$.

Usage

```
ria.test(
  data,
  trt,
  outcome,
  mediators,
  pre,
  post,
  obs = NULL,
  id = NULL,
  d0 = NULL,
  d1 = NULL,
  weights = rep(1, nrow(data)),
  learners = "glm",
  nn_module = sequential_module(),
  control = ria.test.control()
)
```

Arguments

`data`	[`data.frame`] A `data.frame` in wide format containing all necessary variables for the estimation problem.
`trt`	[`character`] A vector containing the column names of treatment variables (D).
`outcome`	[`character(1)`] The column name of the outcome variable.
`mediators`	[`character`] A vector containing the column names of the mediator variables.
`pre`	[`character`] A vector containing the column names of baseline covariates (C) to be controlled for.
`post`	[`character`] A vector containing the column names of post-treatment confounders (L).
`obs`	[`character(1)`] An optional column name (with values coded as 0 or 1) for whether or not the `outcome` is observed. Must be provided if there is missingness in the outcome! Default is `NULL`.
`id`	[`character(1)`] An optional column name containing cluster level identifiers.
`d0`	[`closure`] A two argument function that specifies how treatment variables should be shifted. See examples for how to specify shift functions for continuous, binary, and categorical exposures.
`d1`	[`closure`] A two argument function that specifies how treatment variables should be shifted. See examples for how to specify shift functions for continuous, binary, and categorical exposures.
`weights`	[`numeric`] A optional vector of survey weights.
`learners`	[`character`] A vector of `mlr3superlearner` algorithms for estimation of the outcome regressions. Default is `"glm"`, a main effects GLM.
`nn_module`	[`function`] A function that returns a neural network module.
`control`	[`ria.test.control`] Control parameters for the estimation procedure. Use `ria.test.control()` to set these values.

Value

A `ria.test` object containing TE, TE^R, $TE{-}TE^R$, NIE^R, and NDE^R, along with the matched call.

Examples

```
if (torch::torch_is_installed()) {
 set.seed(123)
 n <- 200
 C <- rnorm(n)
 D <- rbinom(n, 1, plogis(C))
 L <- rnorm(n, D + C)
 M <- rnorm(n, D + L + C)
 Y <- rnorm(n, D + L + M + C)
 dat <- data.frame(C, D, L, M, Y)

 res <- ria.test(
  data = dat,
  trt = "D",
  outcome = "Y",
  mediators = "M",
  pre = "C",
  post = "L",
  d0 = \(data, trt) rep(0, nrow(data)),
  d1 = \(data, trt) rep(1, nrow(data)),
  control = ria.test.control(
   crossfit_folds = 1L,
   mlr3superlearner_folds = 2L,
   lprime_folds = 2L,
   epochs = 2L,
   torch_seed = 123L
  )
 )

 print(res)
 tidy(res)
}
```

`ria.test.control` *ria.test control parameters*

Description

ria.test control parameters

Usage

```
ria.test.control(
  crossfit_folds = 10L,
  mlr3superlearner_folds = 10L,
  lprime_folds = 1L,
  epochs = 100L,
  learning_rate = 0.01,
```

```
  batch_size = 64,
  device = c("cpu", "cuda", "mps"),
  torch_seed = NULL
)
```

Arguments

`crossfit_folds`	[numeric(1)] The number of crossfit folds.
`mlr3superlearner_folds`	[numeric(1)] The number of 'mlr3superlearner' folds.
`lprime_folds`	[numeric(1)] The number of folds to split that data into for calculating L'. With larger sample sizes, a larger number will increase speed.
`epochs`	[numeric(1)] The number of epochs to train the neural network.
`learning_rate`	[numeric(1)] The learning rate for the neural network.
`batch_size`	[numeric(1)] The batch size for mini-batch gradient descent.
`device`	[character(1)] Object representing the device on which a `torch_tensor` is or will be allocated.
`torch_seed`	[integer(1)] Optional seed for Torch's random-number generator. This controls neural-network initialization and dropout but does not affect R's random-number generator. Use `set.seed()` separately to control R-level randomness.

Value

A list of control parameters

Examples

```
if (torch::torch_is_installed()) ria.test.control(crossfit_folds = 5)
```

`sequential_module` *Sequential neural network module function factory*

Description

Sequential neural network module function factory

Usage

```
sequential_module(layers = 1, hidden = 20, dropout = 0.1)
```

Arguments

layers	[numeric(1)] Number of hidden layers.
hidden	[numeric(1)] Number of hidden units.
dropout	[numeric(1)] Dropout rate.

Value

A function that returns a sequential neural network module.

Examples

```
if (torch::torch_is_installed()) sequential_module()
```

---

tidy.ria.test *Tidy a ria.test object*

Description

Tidy a ria.test object

Usage

```
## S3 method for class 'ria.test'
tidy(x, ...)
```

Arguments

x	A ‘ria.test‘ object produced by a call to [ria.test::ria.test()].
...	Unused, included for generic consistency only.

Value

A data frame summarizing the requested effect estimates.

Examples

```
if (torch::torch_is_installed()) {
 set.seed(123)
 n <- 200
 C <- rnorm(n)
 D <- rbinom(n, 1, plogis(C))
 L <- rnorm(n, D + C)
 M <- rnorm(n, D + L + C)
 Y <- rnorm(n, D + L + M + C)
```

```
dat <- data.frame(C, D, L, M, Y)

res <- ria.test(
 data = dat,
 trt = "D",
 outcome = "Y",
 mediators = "M",
 pre = "C",
 post = "L",
 d0 = \(data, trt) rep(0, nrow(data)),
 d1 = \(data, trt) rep(1, nrow(data)),
 control = ria.test.control(
  crossfit_folds = 1L,
  mlr3superlearner_folds = 2L,
  lprime_folds = 2L,
  epochs = 2L,
  torch_seed = 123L
 )
)

print(res)
tidy(res)
}
```

Index