Package ‘crumble’

December 2, 2024

Type Package

Title Flexible and General Mediation Analysis Using Riesz Representers

Version 0.1.2

Maintainer Nicholas Williams <ntwilliams.personal@gmail.com>

Description Implements a modern, unified estimation strategy for common mediation estimands (natural effects, organic effects, interventional effects, and recanting twins) in combination with modified treatment policies as described in Liu, Williams, Rudolph, and Díaz (2024) <doi:10.48550/arXiv.2408.14620>. Estimation makes use of recent advancements in Riesz-learning to estimate a set of required nuisance parameters with deep learning. The result is the capability to estimate mediation effects with binary, categorical, continuous, or multivariate exposures with high-dimensional mediators and mediator-outcome confounders using machine learning.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.0.0)

RoxygenNote 7.3.2

Imports checkmate, Matrix, origami, torch, Rsymphony, purrr, cli, S7, data.table, coro, generics, lmtp, mlr3superlearner, progressr, ife (>= 0.1.0)

Suggests testthat (>= 3.0.0), truncnorm, mma

Config/testthat/edition 3

NeedsCompilation no

Author Nicholas Williams [aut, cre, cph] (<https://orcid.org/0000-0002-1378-4831>), Richard Liu [ctb], Iván Díaz [aut, cph] (<https://orcid.org/0000-0001-9056-2047>)

Repository CRAN

Date/Publication 2024-12-02 17:50:02 UTC

Contents

`crumble` *Flexible and general mediation analysis*

Description

General estimator for common mediation causal effects, including recanting twins, natural effects, organic effects, and randomized interventional effects. Interventions are specified using modified treatment policies. Nuisance parameters are estimated using the 'super learner' algorithm and 'Riesz learning'. Supports binary, categorical, and continuous exposures.

Usage

```
crumble(
  data,
  trt,
  outcome,
  mediators,
  moc = NULL,
  covar,
  obs = NULL,
  id = NULL,
  d0 = NULL,
  d1 = NULL,
  effect = c("RT", "N", "RI", "O"),
  weights = rep(1, nrow(data)),
  learners = "glm",
  nn_module = sequential_module(),
  control = crumble_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.
`outcome`	`[character(1)]` The column name of the outcome variable.

`mediators`	`[character]` A vector containing the column names of the mediator variables.
`moc`	`[character]` An optional vector containing the column names of the mediator-outcome confounders.
`covar`	`[character]` An vector containing the column names of baseline covariates to be controlled for.
`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.
`effect`	`[character(1)]` The type of effect to estimate. Options are `"RT"` for recanting twins, `"N"` for natural effects, `"RI"` for randomized interventional effects, and `"O"` for organic effects. If `"RT"` or `"RI"` is selected, `moc` must be provided. If `"N"` or `"O"` is selected, `moc` must be `NULL`.
`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`	`[crumble_control]` Control parameters for the estimation procedure. Use `crumble_control()` to set these values.

Value

A `crumble` object containing the following components:

`estimates`	A list of parameter estimates.
`outcome_reg`	Predictions from the outcome regressions.
`alpha_n`	A list of density ratio estimates.

alpha_r	A list of density ratio estimates.
fits	A list of the fitted values from the outcome regressions.
call	The matched call.
effect	The estimated effect type.

Examples

```
if (require("mma") && torch::torch_is_installed()) {
library(mma)
data(weight_behavior)

weight_behavior <- na.omit(weight_behavior)

res <- crumble(
data = weight_behavior,
trt = "sports",
outcome = "bmi",
covar = c("age", "sex", "tvhours"),
mediators = c("exercises", "overweigh"),
moc = "snack",
d0 = \(data, trt) factor(rep(1, nrow(data)), levels = c("1", "2")),
d1 = \(data, trt) factor(rep(2, nrow(data)), levels = c("1", "2")),
learners = c("mean", "glm"),
nn_module = sequential_module(),
control = crumble_control(crossfit_folds = 1L, zprime_folds = 5L, epochs = 10L)
)

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

---

crumble_control *Crumble control parameters*

Description

Crumble control parameters

Usage

```
crumble_control(
  crossfit_folds = 10L,
  mlr3superlearner_folds = 10L,
  zprime_folds = 1L,
  epochs = 100L,
  learning_rate = 0.01,
  batch_size = 64,
  device = c("cpu", "cuda", "mps")
)
```

Arguments

`crossfit_folds`	`[numeric(1)]` The number of crossfit folds.
`mlr3superlearner_folds`	`[numeric(1)]` The number of ‘mlr3superlearner‘ folds.
`zprime_folds`	`[numeric(1)]` The number of folds to split that data into for calculating Z’. 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.

Value

A list of control parameters

Examples

```
if (torch::torch_is_installed()) crumble_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.crumble *Tidy a(n) crumble object*

Description

Tidy a(n) crumble object

Usage

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

Arguments

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

Value

A tidy [tibble::tibble()] summarizing information about the model.

Examples

```
if (require("mma") && torch::torch_is_installed()) {
library(mma)
data(weight_behavior)

weight_behavior <- na.omit(weight_behavior)

res <- crumble(
data = weight_behavior,
trt = "sports",
outcome = "bmi",
covar = c("age", "sex", "tvhours"),
mediators = c("exercises", "overweigh"),
moc = "snack",
d0 = \(data, trt) factor(rep(1, nrow(data)), levels = c("1", "2")),
d1 = \(data, trt) factor(rep(2, nrow(data)), levels = c("1", "2")),
learners = c("mean", "glm"),
nn_module = sequential_module(),
control = crumble_control(crossfit_folds = 1L, zprime_folds = 5L, epochs = 10L)
)
print(res)
tidy(res)
}
```

Index