Package: tmle3mopttx (via r-universe)

October 29, 2024

Title Targeted Maximum Likelihood Estimation of the Mean under Optimal Individualized Treatment

Version 1.0.0

Description This package estimates the optimal individualized treatment rule for the categorical treatment using Super Learner (sl3). In order to avoid nested cross-validation, it uses split-specific estimates of Q and g to estimate the rule as described by Coyle et al. In addition, it provides the Targeted Maximum Likelihood estimates of the mean performance using CV-TMLE under such estimated rules. This is an adapter package for use with the tmle3 framework and the tlverse software ecosystem for Targeted Learning.

Depends R (>= 3.3.0)

License GPL-3

Encoding UTF-8

LazyData true

LazyLoad yes

Imports R6, uuid, methods, data.table, assertthat, sl3, tmle3, future, dplyr, foreach, stats, hal9001

Suggests testthat, knitr, rmarkdown, covr, ggplot2, tidyverse, Rsolnp, nnls, xgboost, speedglm, glmnet, randomForest

Remotes github::tlverse/delayed, github::tlverse/origami, github::tlverse/sl3, github::tlverse/tmle3, github::tlverse/tmle3shift

URL https://tlverse.org/tmle3mopttx

BugReports https://github.com/tlverse/tmle3mopttx/issues

VignetteBuilder knitr

RoxygenNote 7.2.0

Repository https://ictml-project.r-universe.dev

RemoteUrl https://github.com/tlverse/tmle3mopttx

RemoteRef HEAD

RemoteSha 3c0e8437af88ca8c54fcec1e953652feaf6c9003

Contents

`create_mv_learners` *Make SL of multivariate learners*

Description

Make SL of multivariate learners

Usage

```
create_mv_learners(learners)
```

Arguments

`learners`	List of learners supporting multivariate prediction.

data_bin *Mock data set with Binary Treatment*

Description

A dataset with a simple data structure O = (A, Y, W), where exposure (A) and outcome (Y) are binary. This is a simple dataset designed specifically to illustrate the TMLE estimation procedure.

Usage

```
data_bin
```

Format

A `data.frame` with 5 columns.

Y A binary variable representing an outcome of interest.

A A binary variable representing an intervention of interest.

W1 A continuous variable representing a covariate of interest.

W2 A continuous variable representing a covariate of interest.

W3 A continuous variable representing a covariate of interest.

data_cat *Mock data set with Categorical Treatment*

Description

A dataset with a simple data structure O = (A, Y, W), where outcome (Y) is binary and treatment (A) is categorical. This is a simple dataset designed specifically to illustrate the TMLE estimation procedure.

Usage

```
data_cat
```

Format

A `data.frame` with 6 columns.

Y A binary variable representing an outcome of interest.

A A binary variable representing an intervention of interest.

W1 A continuous variable representing a covariate of interest.

W2 A continuous variable representing a covariate of interest.

W3 A continuous variable representing a covariate of interest.

W4 A continuous variable representing a covariate of interest.

data_cat_realistic *Mock data set with Categorical Treatment and rare treatment*

Description

A dataset with a simple data structure O = (A, Y, W), where outcome (Y) is binary and treatment (A) is categorical, with one of the categories having low probability. This is a simple dataset designed specifically to illustrate the TMLE estimation procedure with realistic intervention.

Usage

```
data_cat_realistic
```

Format

A `data.frame` with 6 columns.

Y A binary variable representing an outcome of interest.

A A binary variable representing an intervention of interest.

W1 A categorical variable representing a covariate of interest.

W2 A categorical variable representing a covariate of interest.

W3 A categorical variable representing a covariate of interest.

W4 A categorical variable representing a covariate of interest.

data_cat_vim *Mock data set for Variable Importance Analysis with Categorical Treatment*

Description

A dataset with a simple data structure O = (A, Y, W), where outcome (Y) is binary and treatment (A) is categorical. This is a simple dataset designed specifically to illustrate the TMLE estimation procedure.

Usage

```
data_cat_vim
```

Format

A `data.frame` with 6 columns.

Y A binary variable representing an outcome of interest.

A A binary variable representing an intervention of interest.

W1 A categorical variable representing a covariate of interest.

W2 A categorical variable representing a covariate of interest.

W3 A categorical variable representing a covariate of interest.

W4 A categorical variable representing a covariate of interest.

`LF_rule` *Dynamic Likelihood Factor*

Description

Dynamic Likelihood Factor built on top of `LF_base`.

Format

An `R6Class` object inheriting from `LF_base`.

Value

`LF_base` object.

Super class

`tmle3::LF_base` -> `LF_rule`

Methods

Public methods:

- `LF_rule$new()`
- `LF_rule$get_mean()`
- `LF_rule$get_density()`
- `LF_rule$cf_values()`
- `LF_rule$clone()`

Method `new()`:

Usage:

```
LF_rule$new(name, type = "density", rule_fun, ...)
```

Method `get_mean()`:

Usage:

```
LF_rule$get_mean(tmle_task, fold_number)
```

Method `get_density()`**:**

Usage:

```
LF_rule$get_density(tmle_task, fold_number)
```

Method `cf_values()`**:**

Usage:

```
LF_rule$cf_values(tmle_task)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LF_rule$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`normalize_rows` *Normalize rows*

Description

Normalize rows

Usage

```
normalize_rows(x)
```

Arguments

`x`	Values needed to be normalized.

`Optimal_Rule_Q_learning` *Learns the Optimal Rule given a tmle_task and likelihood, using Q learning.*

Description

Learns the Optimal Rule given a tmle_task and likelihood, using Q learning.

Learns the Optimal Rule given a tmle_task and likelihood, using Q learning.

Super class

`tmle3::tmle3_Spec` -> `Optimal_Rule_Q_learning`

Methods

Public methods:

- `Optimal_Rule_Q_learning$new()`
- `Optimal_Rule_Q_learning$fit_blip()`
- `Optimal_Rule_Q_learning$rule()`
- `Optimal_Rule_Q_learning$clone()`

Method `new()`:

Usage:

```
Optimal_Rule_Q_learning$new(tmle_task, likelihood, maximize = TRUE)
```

Method `fit_blip()`:

Usage:

```
Optimal_Rule_Q_learning$fit_blip()
```

Method `rule()`:

Usage:

```
Optimal_Rule_Q_learning$rule(tmle_task, fold_number = "full")
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Optimal_Rule_Q_learning$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Optimal_Rule_Revere` *Learning the Optimal Rule using the Revere framework*

Description

Functions used to learn the Optimal Rule given a tmle_task and likelihood, using the Revere framework. Complements 'tmle3_Spec_mopttx_blip_revere' class.

Format

An `R6Class` object inheriting from `tmle3_Spec`.

Value

A optimal rule object inheriting from `tmle3_Spec` with methods for learning the optimal rule. For a full list of the available functionality, see the complete documentation of `tmle3_Spec`.

Parameters

- `tmle_task`: Task object specifying the data and node structure. - `tmle_spec`: Spec object of `tmle3`. Allows for different Specs to use the current class for learning the Optimal Rule. - `likelihood`: Likelihood object of `tmle3`, corresponding to the current estimate of the required parts of the likelihood necessary for the target parameter. - `V`: User-specified list of covariates used to define the rule. - `options`: Information on all the variables passed to the original Spec.

Super class

`tmle3::tmle3_Spec` -> `Optimal_Rule_Revere`

Methods

Public methods:

- `Optimal_Rule_Revere$new()`
- `Optimal_Rule_Revere$factor_to_indicators()`
- `Optimal_Rule_Revere$V_data()`
- `Optimal_Rule_Revere$DR_full()`
- `Optimal_Rule_Revere$blip_revere_function()`
- `Optimal_Rule_Revere$bound()`
- `Optimal_Rule_Revere$fit_blip()`
- `Optimal_Rule_Revere$rule()`
- `Optimal_Rule_Revere$rule_stochastic()`
- `Optimal_Rule_Revere$clone()`

Method `new()`:

Usage:

```
Optimal_Rule_Revere$new(
  tmle_task,
  tmle_spec,
  likelihood,
  V,
  options,
  shift_grid = seq(-1, 1, by = 0.5)
)
```

Method `factor_to_indicators()`:

Usage:

```
Optimal_Rule_Revere$factor_to_indicators(x, x_vals)
```

Method `V_data()`:

Usage:

```
Optimal_Rule_Revere$V_data(tmle_task, fold = NULL)
```

Method `DR_full()`:

Usage:

```
Optimal_Rule_Revere$DR_full(v, indx)
```

Method `blip_revere_function()`**:**

Usage:

```
Optimal_Rule_Revere$blip_revere_function(tmle_task, fold_number)
```

Method `bound()`**:**

Usage:

```
Optimal_Rule_Revere$bound(cv_g)
```

Method `fit_blip()`**:**

Usage:

```
Optimal_Rule_Revere$fit_blip()
```

Method `rule()`**:**

Usage:

```
Optimal_Rule_Revere$rule(tmle_task, fold_number = "full")
```

Method `rule_stochastic()`**:**

Usage:

```
Optimal_Rule_Revere$rule_stochastic(tmle_task, fold_number = "full")
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Optimal_Rule_Revere$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
library(sl3)
library(tmle3)
library(data.table)

data("data_bin")
data <- data_bin

Q_lib <- make_learner_stack("Lrnr_mean", "Lrnr_glm_fast")
g_lib <- make_learner_stack("Lrnr_mean", "Lrnr_glm_fast")
B_lib <- make_learner_stack("Lrnr_glm_fast", "Lrnr_xgboost")

metalearner <- make_learner(Lrnr_nnls)
Q_learner <- make_learner(Lrnr_sl, Q_lib, metalearner)
g_learner <- make_learner(Lrnr_sl, g_lib, metalearner)
B_learner <- make_learner(Lrnr_sl, B_lib, metalearner)

learner_list <- list(Y = Q_learner, A = g_learner, B = B_learner)
```

```
node_list <- list(W = c("W1", "W2", "W3"), A = "A", Y = "Y")

tmle_spec <- tmle3_mopttx_blip_revere(
  V = c("W1", "W2", "W3"),
  type = "blip1", learners = learner_list, maximize = TRUE,
  complex = TRUE, realistic = TRUE
)

## End(Not run)
```

`Param_TSM_name` *Treatment Specific Mean with names specifying the covariates the rule depends on.*

Description

Treatment Specific Mean with names specifying the covariates the rule depends on.

Format

An `R6Class` object inheriting from `Param_base`.

Value

TSP with name of the intervention specified.

Super class

`tmle3::Param_base` -> `Param_TSM_name`

Methods

Public methods:

- `Param_TSM_name$new()`
- `Param_TSM_name$clever_covariates()`
- `Param_TSM_name$estimates()`
- `Param_TSM_name$clone()`

Method `new()`:

Usage:

```
Param_TSM_name$new(
  observed_likelihood,
  intervention_list,
  v = NULL,
  ...,
  outcome_node = "Y"
)
```

Method `clever_covariates()`**:**

Usage:

```
Param_TSM_name$clever_covariates(tmle_task = NULL, fold_number = "full")
```

Method `estimates()`**:**

Usage:

```
Param_TSM_name$estimates(tmle_task = NULL, fold_number = "full")
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Param_TSM_name$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Q_learning` *Q learning wrapper*

Description

Q learning wrapper

Usage

```
Q_learning(tmle_spec_Q, learner_list, B = 1000, data, node_list)
```

Arguments

`tmle_spec_Q`	TMLE Spec initializing Q learning.
`learner_list`	List of algorithms used to fit Q
`B`	Number of bootstraps
`data`	Dataset used
`node_list`	List of nodes corresponding to Y, A and W.

`tmle3_mopttx_blip_revere` *Mean under the Optimal Individualized Treatment Rule*

Description

O=(W,A,Y) W=Covariates A=Treatment (binary or categorical) Y=Outcome (binary or bounded continuous)

Usage

```
tmle3_mopttx_blip_revere(
  V = NULL,
  type = "blip1",
  learners,
  maximize = TRUE,
  complex = TRUE,
  realistic = FALSE,
  resource = 1,
  interpret = FALSE,
  likelihood_override = NULL,
  reference = NULL
)
```

Arguments

`V`	Covariates the rule depends on.
`type`	One of three psudo-blip versions developed to accommodate categorical treatment. "Blip1" corresponds to chosing a reference category, and defining the blip for all other categories relative to the specified reference. Note that in the case of binary treatment, "blip1" is just the usual blip. "Blip2$ corresponds to defining the blip relative to the average of all categories. Finally, "Blip3" corresponds to defining the blip relative to the weighted average of all categories.
`learners`	Library for Y (outcome), A (treatment), and B (blip) estimation.
`maximize`	Specify whether we want to maximize or minimize the mean of the final outcome.
`complex`	If `TRUE`, learn the rule using the specified covariates `V`. If `FALSE`, check if a less complex rule is better.
`realistic`	If `TRUE`, it will return a rule what is possible due to practical positivity constraints.
`resource`	Indicates the percent of initially estimated individuals who should be given treatment that get treatment, based on their blip estimate. If resource = 1 all estimated individuals to benefit from treatment get treatment, if resource = 0 none get treatment.
`interpret`	If `TRUE`, returns a HAL fit of the blip, explaining the rule.

likelihood_override
: if estimates of the likelihood are known, override learners.

reference
: reference category for blip1. Default is the smallest numerical category or factor.

`tmle3_mopttx_Q` *Mean under the Optimal Individualized Treatment Rule, using Q learning*

Description

O=(W,A,Y) W=Covariates A=Treatment (binary or categorical) Y=Outcome (binary or bounded continuous)

Usage

```
tmle3_mopttx_Q(maximize)
```

Arguments

maximize
: Specify whether we want to maximize or minimize the mean of the final outcome.

`tmle3_mopttx_vim` *Mean under the Optimal Individualized Treatment Rule*

Description

O=(W,A,Y) W=Covariates A=Treatment (binary or categorical) Y=Outcome (binary or bounded continuous)

Usage

```
tmle3_mopttx_vim(
  V = NULL,
  type = "blip2",
  method = "SL",
  learners = NULL,
  contrast = "linear",
  maximize = TRUE,
  complex = TRUE,
  realistic = FALSE,
  resource = 1,
  reference = NULL
)
```

Arguments

V	Covariates the rule depends on.
type	One of three psudo-blip versions developed to accommodate categorical treatment. "Blip1" corresponds to chosing a reference category, and defining the blip for all other categories relative to the specified reference. Note that in the case of binary treatment, "blip1" is just the usual blip. "Blip2$ corresponds to defining the blip relative to the average of all categories. Finally, "Blip3" corresponds to defining the blip relative to the weighted average of all categories.
method	Specifies which methodology to use for learning the rule. Options are "Q" for Q-learning, and "SL" for the Super-Learner approach using split-specific estimates.
learners	Library for Y (outcome), A (treatment), and B (blip) estimation.
contrast	Defined either a "linear" or "multiplicative" contrast for the delta method.
maximize	Specify whether we want to maximize or minimize the mean of the final outcome.
complex	If TRUE, learn the rule using the specified covariates V. If FALSE, check if a less complex rule is better.
realistic	If TRUE, it will return a rule what is possible due to practical positivity constraints.
resource	Indicates the percent of initially estimated individuals who should be given treatment that get treatment, based on their blip estimate. If resource = 1 all estimated individuals to benefit from treatment get treatment, if resource = 0 none get treatment.
reference	reference category for blip1. Default is the smallest numerical category or factor.

---

tmle3_Spec_mopttx_blip_revere

TMLE for the Mean Under the Optimal Individualized Rule

Description

The functions contained in the class define a TMLE for the Mean Under the Optimal Individualized Rule with Categorical Treatment, learned and estimated under Revere CV-TMLE. For learning the Optimal Rule, see 'Optimal_Rule_Revere' class.

Format

An R6Class object inheriting from tmle3_Spec.

Value

A tmle3 object inheriting from tmle3_Spec with methods for obtaining the TMLE for the Mean Under the Optimal Individualized Rule. For a full list of the available functionality, see the complete documentation of tmle3_Spec.

Parameters

- `V`: User-specified list of covariates used to define the rule. - `type`: Blip type, corresponding to different ways of defining the reference category in learning the blip; mostly applies to categorical treatment. Available categories include "blip1" (reference level of treatment), "blip2" (average level of treatment) and "blip3" (weighted average level of treatment). - `learners`: List of user-defined learners for relevant parts of the likelihood. - `maximize`: Should the average outcome be maximized of minimized? Default is maximize=TRUE. - `complex`: If `TRUE`, the returned mean under the Optimal Rule is based on the full set of covariates provided by the user (parameter "V"). If `FALSE`, simpler rules (including the static rules), are evaluated as well; the returned mean under the Optimal Rule is then a potentially more parsimonious rule, if the mean performance is similar. - `realistic`: If `TRUE`, the optimal rule returned takes into account the probability of treatment given covariates. - `resource`: Indicates the percent of initially estimated individuals who should be given treatment that get treatment, based on their blip estimate. If resource = 1 all estimated individuals to benefit from treatment get treatment, if resource = 0 none get treatment. - `interpret`: If `TRUE`, returns a HAL fit of the blip, explaining the rule. - `reference`: reference category for blip1. Default is the smallest numerical category or factor.

Super class

`tmle3::tmle3_Spec` -> `tmle3_Spec_mopttx_blip_revere`

Methods

Public methods:

- `tmle3_Spec_mopttx_blip_revere$new()`
- `tmle3_Spec_mopttx_blip_revere$vals_from_factor()`
- `tmle3_Spec_mopttx_blip_revere$make_tmle_task()`
- `tmle3_Spec_mopttx_blip_revere$make_initial_likelihood()`
- `tmle3_Spec_mopttx_blip_revere$predict_rule()`
- `tmle3_Spec_mopttx_blip_revere$make_rules()`
- `tmle3_Spec_mopttx_blip_revere$make_est_fin()`
- `tmle3_Spec_mopttx_blip_revere$set_opt()`
- `tmle3_Spec_mopttx_blip_revere$set_rule()`
- `tmle3_Spec_mopttx_blip_revere$data_adapt_psi()`
- `tmle3_Spec_mopttx_blip_revere$get_blip_fit()`
- `tmle3_Spec_mopttx_blip_revere$get_blip_pred()`
- `tmle3_Spec_mopttx_blip_revere$make_params()`
- `tmle3_Spec_mopttx_blip_revere$clone()`

Method `new()`:

Usage:

```
tmle3_Spec_mopttx_blip_revere$new(
  V = NULL,
  type,
  learners,
  maximize = TRUE,
```

```
  complex = TRUE,
  realistic = FALSE,
  resource = 1,
  interpret = FALSE,
  likelihood_override = NULL,
  reference = NULL,
  ...
)
```

Method `vals_from_factor()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$vals_from_factor(x)
```

Method `make_tmle_task()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$make_tmle_task(data, node_list, ...)
```

Method `make_initial_likelihood()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$make_initial_likelihood(
  tmle_task,
  learner_list = NULL
)
```

Method `predict_rule()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$predict_rule(tmle_task_new)
```

Method `make_rules()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$make_rules(V)
```

Method `make_est_fin()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$make_est_fin(fit, max, p.value = 0.35)
```

Method `set_opt()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$set_opt(opt)
```

Method `set_rule()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$set_rule(rule)
```

Method `data_adapt_psi()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$data_adapt_psi(data_tda, node_list, Qbar0)
```

Method `get_blip_fit()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$get_blip_fit()
```

Method `get_blip_pred()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$get_blip_pred(tmle_task, fold_number = "full")
```

Method `make_params()`**:**

Usage:

```
tmle3_Spec_mopttx_blip_revere$make_params(tmle_task, likelihood)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
tmle3_Spec_mopttx_blip_revere$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:
library(sl3)
library(tmle3)
library(data.table)

data("data_bin")
data <- data_bin

Q_lib <- make_learner_stack("Lrnr_mean", "Lrnr_glm_fast")
g_lib <- make_learner_stack("Lrnr_mean", "Lrnr_glm_fast")
B_lib <- make_learner_stack("Lrnr_glm_fast", "Lrnr_xgboost")

metalearner <- make_learner(Lrnr_nnls)
Q_learner <- make_learner(Lrnr_sl, Q_lib, metalearner)
g_learner <- make_learner(Lrnr_sl, g_lib, metalearner)
B_learner <- make_learner(Lrnr_sl, B_lib, metalearner)

learner_list <- list(Y = Q_learner, A = g_learner, B = B_learner)

node_list <- list(W = c("W1", "W2", "W3"), A = "A", Y = "Y")

tmle_spec <- tmle3_mopttx_blip_revere(
  V = c("W1", "W2", "W3"),
  type = "blip1", learners = learner_list, maximize = TRUE,
  complex = TRUE, realistic = TRUE
)

## End(Not run)
```

tmle3_Spec_mopttx_Q *Defines the Mean Under the Optimal Individualized Rule with Categorical Treatment, estimated using Q learning (single step)*

Description

Defines the Mean Under the Optimal Individualized Rule with Categorical Treatment, estimated using Q learning (single step)

Defines the Mean Under the Optimal Individualized Rule with Categorical Treatment, estimated using Q learning (single step)

Super class

tmle3::tmle3_Spec -> tmle3_Spec_mopttx_Q

Methods

Public methods:

- tmle3_Spec_mopttx_Q$new()
- tmle3_Spec_mopttx_Q$vals_from_factor()
- tmle3_Spec_mopttx_Q$make_initial_likelihood_glm()
- tmle3_Spec_mopttx_Q$make_params()
- tmle3_Spec_mopttx_Q$estimate()
- tmle3_Spec_mopttx_Q$clone()

Method new():

Usage:

```
tmle3_Spec_mopttx_Q$new(maximize = TRUE, ...)
```

Method vals_from_factor():

Usage:

```
tmle3_Spec_mopttx_Q$vals_from_factor(x)
```

Method make_initial_likelihood_glm():

Usage:

```
tmle3_Spec_mopttx_Q$make_initial_likelihood_glm(tmle_task)
```

Method make_params():

Usage:

```
tmle3_Spec_mopttx_Q$make_params(tmle_task, likelihood)
```

Method estimate():

Usage:

```
tmle3_Spec_mopttx_Q$estimate(tmle_params, tmle_task)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
tmle3_Spec_mopttx_Q$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`tmle3_Spec_mopttx_vim` *Variable Importance with the Mean Under the Optimal Individualized Rule*

Description

The functions contained in the class define a Variable Importance metric for the TMLE of the Mean Under the Optimal Individualized Rule with Categorical Treatment, learned and estimated under Revere CV-TMLE. For learning the Optimal Rule, see 'Optimal_Rule_Revere' class.

Format

An `R6Class` object inheriting from `tmle3_Spec`.

Value

A tmle3 object inheriting from `tmle3_Spec` with methods for obtaining the Variable Importance metric for the TMLE of the Mean Under the Optimal Individualized Rule. For a full list of the available functionality, see the complete documentation of `tmle3_Spec`.

Parameters

- `V`: User-specified list of covariates used to define the rule. - `type`: Blip type, corresponding to different ways of defining the reference category in learning the blip; mostly applies to categorical treatment. Available categories include "blip1" (reference level of treatment), "blip2" (average level of treatment) and "blip3" (weighted average level of treatment). - `method`: Either "SL" (for the TMLE estimate) or "Q" (for Q-learning). - `learners`: List of user-defined learners for relevant parts of the likelihood. - `contrast`: Defined either a "linear" or "multiplicative" contrast for the delta method. - `maximize`: Should the average outcome be maximized of minimized? Default is maximize=TRUE. - `complex`: If `TRUE`, the returned mean under the Optimal Rule is based on the full set of covariates provided by the user (parameter "V"). If `FALSE`, simpler rules (including the static rules), are evaluated as well; the returned mean under the Optimal Rule is then a potentially more parsimonious rule, if the mean performance is similar. - `realistic`: If `TRUE`, the optimal rule returned takes into account the probability of treatment given covariates. - `resource`: Indicates the percent of initially estimated individuals who should be given treatment that get treatment, based on their blip estimate. If resource = 1 all estimated individuals to benefit from treatment get treatment, if resource = 0 none get treatment.

Super classes

`tmle3::tmle3_Spec` -> `tmle3mopttx::tmle3_Spec_mopttx_blip_revere` -> `tmle3_Spec_mopttx_vim`

Methods

Public methods:

- tmle3_Spec_mopttx_vim$new()
- tmle3_Spec_mopttx_vim$vals_from_factor()
- tmle3_Spec_mopttx_vim$make_tmle_task()
- tmle3_Spec_mopttx_vim$set_opt()
- tmle3_Spec_mopttx_vim$make_params()
- tmle3_Spec_mopttx_vim$clone()

Method `new()`:

Usage:

```
tmle3_Spec_mopttx_vim$new(
  V = NULL,
  type = "blip2",
  method = "SL",
  learners = NULL,
  contrast = "linear",
  maximize = TRUE,
  complex = TRUE,
  realistic = FALSE,
  resource = 1,
  reference = NULL,
  ...
)
```

Method `vals_from_factor()`:

Usage:

```
tmle3_Spec_mopttx_vim$vals_from_factor(x)
```

Method `make_tmle_task()`:

Usage:

```
tmle3_Spec_mopttx_vim$make_tmle_task(data, node_list, ...)
```

Method `set_opt()`:

Usage:

```
tmle3_Spec_mopttx_vim$set_opt(opt)
```

Method `make_params()`:

Usage:

```
tmle3_Spec_mopttx_vim$make_params(tmle_task, likelihood)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
tmle3_Spec_mopttx_vim$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
library(sl3)
library(tmle3)
library(data.table)

data("data_cat_vim")
data <- data_cat_vim
data$A <- as.integer(data$A)

lrn1 <- Lrnr_mean$new()
lrn2 <- Lrnr_glm_fast$new()
lrn3 <- Lrnr_glmnet$new()

Q_learner <- Lrnr_sl$new(learners = list(lrn1, lrn2, lrn3),
metalearner = Lrnr_nnls$new()
)

mn_metalearner <- make_learner(Lrnr_solnp,
loss_function = loss_loglik_multinomial,
learner_function = metalearner_linear_multinomial
)
g_learner <- make_learner(Lrnr_sl, list(lrn1, lrn3),
mn_metalearner)

b_learner <- create_mv_learners(learners = list(lrn1, lrn2))

learner_list <- list(Y = Q_learner, A = g_learner, B = b_learner)

node_list <- list(W = c("W2", "W3", "W4"),
A = c("A", "W1"), Y = "Y")

tmle_spec <- tmle3_mopttx_vim(
V = "W3", learners = learner_list, type = "blip2",
contrast = "multiplicative", maximize = FALSE,
method = "SL", complex = TRUE, realistic = FALSE
)

## End(Not run)
```

---

vals_from_factor *Get factors*

Description

Get factors

Usage

```
vals_from_factor(x)
```

Arguments

x	Values from which we obtain factors.

Index