Package 'tidyhte'

August 14, 2023

Title Tidy Estimation of Heterogeneous Treatment Effects

Version 1.0.2

Description Estimates heterogeneous treatment effects using tidy semantics on experimental or observational data. Methods are based on the doubly-robust learner of Kennedy (n.d.) <arXiv:2004.14497>. You provide a simple recipe for what machine learning algorithms to use in estimating the nuisance functions and 'tidyhte' will take care of cross-validation, estimation, model selection, diagnostics and construction of relevant quantities of interest about the variability of treatment effects.

URL https://github.com/ddimmery/tidyhte

https://ddimmery.github.io/tidyhte/index.html

BugReports https://github.com/ddimmery/tidyhte/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Suggests covr, devtools, estimatr, ggplot2, glmnet, knitr, mockr, nprobust, palmerpenguins, quadprog, quickblock, rmarkdown, testthat (>= 3.0.0), vimp, WeightedROC

Config/testthat/edition 3

Imports checkmate, dplyr, lifecycle, magrittr, progress, purrr, R6, rlang, SuperLearner, tibble

VignetteBuilder knitr

NeedsCompilation no

Author Drew Dimmery [aut, cre, cph] (<https://orcid.org/0000-0001-9602-6325>)

Maintainer Drew Dimmery <drew.dimmery@univie.ac.at>

Repository CRAN

Date/Publication 2023-08-14 11:30:02 UTC

R topics documented:

`add_effect_diagnostic` *Add an additional diagnostic to the effect model*

Description

This adds a diagnostic to the effect model.

Usage

```
add_effect_diagnostic(hte_cfg, diag)
```

Arguments

hte_cfg	HTE_cfg object to update.
diag	Character indicating the name of the diagnostic to include. Possible values are "MSE", "RROC" and, for SuperLearner ensembles, "SL_risk" and "SL_coefs".

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   add_effect_diagnostic("RROC") -> hte_cfg
```

---

add_effect_model *Add an additional model to the joint effect ensemble*

Description

This adds a learner to the ensemble used for estimating a model of the conditional expectation of the pseudo-outcome.

Usage

```
add_effect_model(hte_cfg, model_name, ...)
```

Arguments

hte_cfg	HTE_cfg object to update.
model_name	Character indicating the name of the model to incorporate into the joint effect ensemble. Possible values use SuperLearner naming conventions. A full list is available with SuperLearner::listWrappers("SL")
...	Parameters over which to grid-search for this model class.

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   add_effect_model("SL.glm.interaction") -> hte_cfg
```

add_known_propensity_score *Uses a known propensity score*

Description

This replaces the propensity score model with a known value of the propensity score.

Usage

```
add_known_propensity_score(hte_cfg, covariate_name)
```

Arguments

hte_cfg	HTE_cfg object to update.
covariate_name	Character indicating the name of the covariate name in the dataframe corresponding to the known propensity score.

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   add_known_propensity_score("ps") -> hte_cfg
```

add_moderator *Adds moderators to the configuration*

Description

This adds a definition about how to display a moderators to the MCATE config. A moderator is any variable that you want to view information about CATEs with respect to.

Usage

```
add_moderator(hte_cfg, model_type, ..., .model_arguments = NULL)
```

Arguments

hte_cfg	HTE_cfg object to update.
model_type	Character indicating the model type for these moderators. Currently two model types are supported: "Stratified" for discrete moderators and "KernelSmooth" for continuous ones.
...	The (unquoted) names of the moderator variables.
.model_arguments	A named list from argument name to value to pass into the constructor for the model. See Stratified_cfg and KernelSmooth_cfg for more details.

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   add_moderator("Stratified", x2, x3) %>%
   add_moderator("KernelSmooth", x1, x4, x5) -> hte_cfg
```

---

add_outcome_diagnostic

Add an additional diagnostic to the outcome model

Description

This adds a diagnostic to the outcome model.

Usage

```
add_outcome_diagnostic(hte_cfg, diag)
```

Arguments

hte_cfg	HTE_cfg object to update.
diag	Character indicating the name of the diagnostic to include. Possible values are "MSE", "RROC" and, for SuperLearner ensembles, "SL_risk" and "SL_coefs".

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   add_outcome_diagnostic("RROC") -> hte_cfg
```

add_outcome_model *Add an additional model to the outcome ensemble*

Description

This adds a learner to the ensemble used for estimating a model of the conditional expectation of the outcome.

Usage

```
add_outcome_model(hte_cfg, model_name, ...)
```

Arguments

hte_cfg	HTE_cfg object to update.
model_name	Character indicating the name of the model to incorporate into the outcome ensemble. Possible values use SuperLearner naming conventions. A full list is available with SuperLearner::listWrappers("SL")
...	Parameters over which to grid-search for this model class.

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   add_outcome_model("SL.glm.interaction") -> hte_cfg
```

add_propensity_diagnostic *Add an additional diagnostic to the propensity score*

Description

This adds a diagnostic to the propensity score.

Usage

```
add_propensity_diagnostic(hte_cfg, diag)
```

Arguments

hte_cfg	HTE_cfg object to update.
diag	Character indicating the name of the diagnostic to include. Possible values are "MSE", "AUC" and, for SuperLearner ensembles, "SL_risk" and "SL_coefs".

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   add_propensity_diagnostic(c("AUC", "MSE")) -> hte_cfg
```

`add_propensity_score_model` *Add an additional model to the propensity score ensemble*

Description

This adds a learner to the ensemble used for estimating propensity scores.

Usage

```
add_propensity_score_model(hte_cfg, model_name, ...)
```

Arguments

hte_cfg	HTE_cfg object to update.
model_name	Character indicating the name of the model to incorporate into the propensity score ensemble. Possible values use SuperLearner naming conventions. A full list is available with SuperLearner::listWrappers("SL")
...	Parameters over which to grid-search for this model class.

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   add_propensity_score_model("SL.glmnet", alpha = c(0, 0.5, 1)) -> hte_cfg
```

add_vimp *Adds variable importance information*

Description

This adds a variable importance quantity of interest to the outputs.

Usage

```
add_vimp(hte_cfg, sample_splitting = TRUE, linear_only = FALSE)
```

Arguments

hte_cfg	HTE_cfg object to update.
sample_splitting	Logical indicating whether to use sample splitting or not. Choosing not to use sample splitting means that inference will only be valid for moderators with non-null importance.
linear_only	Logical indicating whether the variable importance should use only a single linear-only model. Variable importance measure will only be consistent for the population quantity if the true model of pseudo-outcomes is linear.

Value

Updated HTE_cfg object

References

- Williamson, B. D., Gilbert, P. B., Carone, M., & Simon, N. (2021). Nonparametric variable importance assessment using machine learning techniques. Biometrics, 77(1), 9-22.
- Williamson, B. D., Gilbert, P. B., Simon, N. R., & Carone, M. (2021). A general framework for inference on algorithm-agnostic variable importance. Journal of the American Statistical Association, 1-14.

Examples

```
library("dplyr")
basic_config() %>%
   add_vimp(sample_splitting = FALSE) -> hte_cfg
```

attach_config *Attach an* HTE_cfg *to a dataframe*

Description

This adds a configuration attribute to a dataframe for HTE estimation. This configuration details the full analysis of HTE that should be performed.

Usage

```
attach_config(data, .HTE_cfg)
```

Arguments

data	dataframe
.HTE_cfg	HTE_cfg object representing the full configuration of the HTE analysis.

Details

For information about how to set up an HTE_cfg object, see the Recipe API documentation basic_config().

To see an example analysis, read vignette("experimental_analysis") in the context of an experiment, vignette("experimental_analysis") for an observational study, or vignette("methodological_details") for a deeper dive under the hood.

See Also

basic_config(), make_splits(), produce_plugin_estimates(), construct_pseudo_outcomes(), estimate_QoI()

Examples

```
library("dplyr")
if(require("palmerpenguins")) {
data(package = 'palmerpenguins')
penguins$unitid = seq_len(nrow(penguins))
penguins$propensity = rep(0.5, nrow(penguins))
penguins$treatment = rbinom(nrow(penguins), 1, penguins$propensity)
cfg <- basic_config() %>%
add_known_propensity_score("propensity") %>%
add_outcome_model("SL.glm.interaction") %>%
remove_vimp()
attach_config(penguins, cfg) %>%
make_splits(unitid, .num_splits = 4) %>%
produce_plugin_estimates(outcome = body_mass_g, treatment = treatment, species, sex) %>%
construct_pseudo_outcomes(body_mass_g, treatment) %>%
estimate_QoI(species, sex)
}
```

basic_config *Create a basic config for HTE estimation*

Description

This provides a basic recipe for HTE estimation that can be extended by providing additional information about models to be estimated and what quantities of interest should be returned based on those models. This basic model includes only linear models for nuisance function estimation, and basic diagnostics.

Usage

```
basic_config()
```

Details

Additional models, diagnostics and quantities of interest should be added using their respective helper functions provided as part of the Recipe API.

To see an example analysis, read `vignette("experimental_analysis")` in the context of an experiment, `vignette("experimental_analysis")` for an observational study, or `vignette("methodological_details")` for a deeper dive under the hood.

Value

HTE_cfg object

See Also

add_propensity_score_model(), add_known_propensity_score(), add_propensity_diagnostic(), add_outcome_model(), add_outcome_diagnostic(), add_effect_model(), add_effect_diagnostic(), add_moderator(), add_vimp()

Examples

```
library("dplyr")
basic_config() %>%
   add_known_propensity_score("ps") %>%
   add_outcome_model("SL.glm.interaction") %>%
   add_outcome_model("SL.glmnet", alpha = c(0.05, 0.15, 0.2, 0.25, 0.5, 0.75)) %>%
  add_outcome_model("SL.glmnet.interaction", alpha = c(0.05, 0.15, 0.2, 0.25, 0.5, 0.75)) %>%
   add_outcome_diagnostic("RROC") %>%
   add_effect_model("SL.glm.interaction") %>%
   add_effect_model("SL.glmnet", alpha = c(0.05, 0.15, 0.2, 0.25, 0.5, 0.75)) %>%
  add_effect_model("SL.glmnet.interaction", alpha = c(0.05, 0.15, 0.2, 0.25, 0.5, 0.75)) %>%
   add_effect_diagnostic("RROC") %>%
   add_moderator("Stratified", x2, x3) %>%
   add_moderator("KernelSmooth", x1, x4, x5) %>%
   add_vimp(sample_splitting = FALSE) -> hte_cfg
```

Constant_cfg *Configuration of a Constant Estimator*

Description

Constant_cfg is a configuration class for estimating a constant model. That is, the model is a simple, one-parameter mean model.

Super class

tidyhte::Model_cfg -> Constant_cfg

Public fields

model_class The class of the model, required for all classes which inherit from Model_cfg.

Methods

Public methods:

- Constant_cfg$new()
- Constant_cfg$clone()

Method new(): Create a new Constant_cfg object.

Usage:

```
Constant_cfg$new()
```

Returns: A new Constant_cfg object.

Examples:

```
Constant_cfg$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Constant_cfg$new`
## ------------------------------------------------

Constant_cfg$new()
```

construct_pseudo_outcomes *Construct Pseudo-outcomes*

Description

construct_pseudo_outcomes takes a dataset which has been prepared with plugin estimators of nuisance parameters and transforms these into a "pseudo-outcome": an unbiased estimator of the conditional average treatment effect under exogeneity.

Usage

```
construct_pseudo_outcomes(data, outcome, treatment, type = "dr")
```

Arguments

data	dataframe (already prepared with attach_config, make_splits, and produce_plugin_estimates)
outcome	Unquoted name of outcome variable.
treatment	Unquoted name of treatment variable.
type	String representing how to construct the pseudo-outcome. Valid values are "dr" (the default), "ipw" and "plugin". See "Details" for more discussion of these options.

Details

Taking averages of these pseudo-outcomes (or fitting a model to them) will approximate averages (or models) of the underlying treatment effect.

See Also

attach_config(), make_splits(), produce_plugin_estimates(), estimate_QoI()

Diagnostics_cfg *Configuration of Model Diagnostics*

Description

Diagnostics_cfg is a configuration class for estimating a variety of diagnostics for the models trained in the course of HTE estimation.

Public fields

ps Model diagnostics for the propensity score model.

outcome Model diagnostics for the outcome models.

effect Model diagnostics for the joint effect model.

params Parameters for any requested diagnostics.

Methods

Public methods:

- Diagnostics_cfg$new()
- Diagnostics_cfg$add()
- Diagnostics_cfg$clone()

Method `new()`**:** Create a new `Diagnostics_cfg` object with specified diagnostics to estimate.

Usage:

```
Diagnostics_cfg$new(ps = NULL, outcome = NULL, effect = NULL, params = NULL)
```

Arguments:

`ps` Model diagnostics for the propensity score model.

`outcome` Model diagnostics for the outcome models.

`effect` Model diagnostics for the joint effect model.

`params` List providing values for parameters to any requested diagnostics.

Returns: A new `Diagnostics_cfg` object.

Examples:

```
Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE", "RROC"),
   ps = c("SL_risk", "SL_coefs", "AUC")
)
```

Method `add()`**:** Add diagnostics to the `Diagnostics_cfg` object.

Usage:

```
Diagnostics_cfg$add(ps = NULL, outcome = NULL, effect = NULL)
```

Arguments:

`ps` Model diagnostics for the propensity score model.

`outcome` Model diagnostics for the outcome models.

`effect` Model diagnostics for the joint effect model.

Returns: An updated `Diagnostics_cfg` object.

Examples:

```
cfg <- Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE", "RROC"),
   ps = c("SL_risk", "SL_coefs")
)
cfg <- cfg$add(ps = "AUC")
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE", "RROC"),
   ps = c("SL_risk", "SL_coefs", "AUC")
)

## ------------------------------------------------
## Method `Diagnostics_cfg$new`
## ------------------------------------------------

Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE", "RROC"),
   ps = c("SL_risk", "SL_coefs", "AUC")
)

## ------------------------------------------------
## Method `Diagnostics_cfg$add`
## ------------------------------------------------

cfg <- Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE", "RROC"),
   ps = c("SL_risk", "SL_coefs")
)
cfg <- cfg$add(ps = "AUC")
```

`estimate_QoI` *Estimate Quantities of Interest*

Description

`estimate_QoI` takes a dataframe already prepared with split IDs, plugin estimates and pseudo-outcomes and calculates the requested quantities of interest (QoIs).

Usage

```
estimate_QoI(data, ...)
```

Arguments

`data`	data frame (already prepared with `attach_config`, `make_splits`, `produce_plugin_estimates` and `construct_pseudo_outcomes`)
`...`	Unquoted names of moderators to calculate QoIs for.

Details

To see an example analysis, read `vignette("experimental_analysis")` in the context of an experiment, `vignette("experimental_analysis")` for an observational study, or `vignette("methodological_details")` for a deeper dive under the hood.

See Also

attach_config(), make_splits(), produce_plugin_estimates(), construct_pseudo_outcomes(),

Examples

```
library("dplyr")
if(require("palmerpenguins")) {
data(package = 'palmerpenguins')
penguins$unitid = seq_len(nrow(penguins))
penguins$propensity = rep(0.5, nrow(penguins))
penguins$treatment = rbinom(nrow(penguins), 1, penguins$propensity)
cfg <- basic_config() %>%
add_known_propensity_score("propensity") %>%
add_outcome_model("SL.glm.interaction") %>%
remove_vimp()
attach_config(penguins, cfg) %>%
make_splits(unitid, .num_splits = 4) %>%
produce_plugin_estimates(outcome = body_mass_g, treatment = treatment, species, sex) %>%
construct_pseudo_outcomes(body_mass_g, treatment) %>%
estimate_QoI(species, sex)
}
```

HTE_cfg *Configuration of Quantities of Interest*

Description

HTE_cfg is a configuration class that pulls everything together, indicating the full configuration for a given HTE analysis. This includes how to estimate models and what Quantities of Interest to calculate based off those underlying models.

Public fields

`outcome` Model_cfg object indicating how outcome models should be estimated.

`treatment` Model_cfg object indicating how the propensity score model should be estimated.

`effect` Model_cfg object indicating how the joint effect model should be estimated.

`qoi` QoI_cfg object indicating what the Quantities of Interest are and providing all necessary detail on how they should be estimated.

`verbose` Logical indicating whether to print debugging information.

Methods

Public methods:

- HTE_cfg$new()
- HTE_cfg$clone()

Method `new()`**:** Create a new `HTE_cfg` object with all necessary information about how to carry out an HTE analysis.

Usage:

```
HTE_cfg$new(
  outcome = NULL,
  treatment = NULL,
  effect = NULL,
  qoi = NULL,
  verbose = FALSE
)
```

Arguments:

`outcome` `Model_cfg` object indicating how outcome models should be estimated.

`treatment` `Model_cfg` object indicating how the propensity score model should be estimated.

`effect` `Model_cfg` object indicating how the joint effect model should be estimated.

`qoi` `QoI_cfg` object indicating what the Quantities of Interest are and providing all necessary detail on how they should be estimated.

`verbose` Logical indicating whether to print debugging information.

Examples:

```
mcate_cfg <- MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))
pcate_cfg <- PCATE_cfg$new(
   cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)),
   model_covariates = c("x1", "x2", "x3"),
   num_mc_samples = list(x1 = 100)
)
vimp_cfg <- VIMP_cfg$new()
diag_cfg <- Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE"),
   ps = c("SL_risk", "SL_coefs", "AUC")
)
qoi_cfg <- QoI_cfg$new(
    mcate = mcate_cfg,
    pcate = pcate_cfg,
    vimp = vimp_cfg,
    diag = diag_cfg
)
ps_cfg <- SLEnsemble_cfg$new(
  learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)
y_cfg <- SLEnsemble_cfg$new(
  learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)
fx_cfg <- SLEnsemble_cfg$new(
  learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)
HTE_cfg$new(outcome = y_cfg, treatment = ps_cfg, effect = fx_cfg, qoi = qoi_cfg)
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `HTE_cfg$new`
## ------------------------------------------------

mcate_cfg <- MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))
pcate_cfg <- PCATE_cfg$new(
   cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)),
   model_covariates = c("x1", "x2", "x3"),
   num_mc_samples = list(x1 = 100)
)
vimp_cfg <- VIMP_cfg$new()
diag_cfg <- Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE"),
   ps = c("SL_risk", "SL_coefs", "AUC")
)
qoi_cfg <- QoI_cfg$new(
    mcate = mcate_cfg,
    pcate = pcate_cfg,
    vimp = vimp_cfg,
    diag = diag_cfg
)
ps_cfg <- SLEnsemble_cfg$new(
   learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)
y_cfg <- SLEnsemble_cfg$new(
   learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)
fx_cfg <- SLEnsemble_cfg$new(
   learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)
HTE_cfg$new(outcome = y_cfg, treatment = ps_cfg, effect = fx_cfg, qoi = qoi_cfg)
```

KernelSmooth_cfg *Configuration for a Kernel Smoother*

Description

KernelSmooth_cfg is a configuration class for non-parametric local-linear regression to construct a smooth representation of the relationship between two variables. This is typically used for displaying a surface of the conditional average treatment effect over a continuous covariate.

Kernel smoothing is handled by the nprobust package.

Super class

tidyhte::Model_cfg -> KernelSmooth_cfg

Public fields

model_class The class of the model, required for all classes which inherit from Model_cfg.

neval The number of points at which to evaluate the local regression. More points will provide a smoother line at the cost of somewhat higher computation.

eval_min_quantile Minimum quantile at which to evaluate the smoother.

Methods

Public methods:

- KernelSmooth_cfg$new()
- KernelSmooth_cfg$clone()

Method new(): Create a new KernelSmooth_cfg object with specified number of evaluation points.

Usage:

```
KernelSmooth_cfg$new(neval = 100, eval_min_quantile = 0.05)
```

Arguments:

neval The number of points at which to evaluate the local regression. More points will provide a smoother line at the cost of somewhat higher computation.

eval_min_quantile Minimum quantile at which to evaluate the smoother. A value of zero will do no clipping. Clipping is performed from both the top and the bottom of the empirical distribution. A value of alpha would evaluate over [alpha, 1 - alpha].

Returns: A new KernelSmooth_cfg object.

Examples:

```
KernelSmooth_cfg$new(neval = 100)
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

nprobust::lprobust

Examples

```
## ------------------------------------------------
## Method `KernelSmooth_cfg$new`
## ------------------------------------------------

KernelSmooth_cfg$new(neval = 100)
```

Known_cfg *Configuration of Known Model*

Description

Known_cfg is a configuration class for when a particular model is known a-priori. The prototypical usage of this class is when heterogeneous treatment effects are estimated in the context of a randomized control trial with known propensity scores.

Super class

tidyhte::Model_cfg -> Known_cfg

Public fields

covariate_name The name of the column in the dataset which corresponds to the known model score.

model_class The class of the model, required for all classes which inherit from Model_cfg.

Methods

Public methods:

- Known_cfg$new()
- Known_cfg$clone()

Method new(): Create a new Known_cfg object with specified covariate column.

Usage:

```
Known_cfg$new(covariate_name)
```

Arguments:

covariate_name The name of the column, a string, in the dataset corresponding to the known model score (i.e. the true conditional expectation).

Returns: A new Known_cfg object.

Examples:

```
Known_cfg$new("propensity_score")
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Known_cfg$new`
## ------------------------------------------------

Known_cfg$new("propensity_score")
```

`make_splits` *Define splits for cross-fitting*

Description

This takes a dataset, a column with a unique identifier and an arbitrary number of covariates on which to stratify the splits. It returns the original dataset with an additional column `.split_id` corresponding to an identifier for the split.

Usage

```
make_splits(data, identifier, ..., .num_splits)
```

Arguments

`data`	dataframe
`identifier`	Unquoted name of unique identifier column
`...`	variables on which to stratify (requires that `quickblock` be installed.)
`.num_splits`	number of splits to create. If VIMP is requested in `QoI_cfg`, this must be an even number.

Details

To see an example analysis, read `vignette("experimental_analysis")` in the context of an experiment, `vignette("experimental_analysis")` for an observational study, or `vignette("methodological_details")` for a deeper dive under the hood.

Value

original dataframe with additional `.split_id` column

See Also

`attach_config()`, `produce_plugin_estimates()`, `construct_pseudo_outcomes()`, `estimate_QoI()`

Examples

```
library("dplyr")
if(require("palmerpenguins")) {
data(package = 'palmerpenguins')
penguins$unitid = seq_len(nrow(penguins))
penguins$propensity = rep(0.5, nrow(penguins))
penguins$treatment = rbinom(nrow(penguins), 1, penguins$propensity)
cfg <- basic_config() %>%
add_known_propensity_score("propensity") %>%
add_outcome_model("SL.glm.interaction") %>%
remove_vimp()
attach_config(penguins, cfg) %>%
make_splits(unitid, .num_splits = 4) %>%
produce_plugin_estimates(outcome = body_mass_g, treatment = treatment, species, sex) %>%
construct_pseudo_outcomes(body_mass_g, treatment) %>%
estimate_QoI(species, sex)
}
```

MCATE_cfg *Configuration of Marginal CATEs*

Description

MCATE_cfg is a configuration class for estimating marginal response surfaces based on heterogeneous treatment effect estimates. "Marginal" in this context implies that all other covariates are marginalized. Thus, if two covariates are highly correlated, it is likely that their MCATE surfaces will be extremely similar.

Public fields

`cfgs` Named list of covariates names to a `Model_cfg` object defining how to present that covariate's CATE surface (while marginalizing over all other covariates).

`std_errors` Boolean indicating whether the results should be returned with standard errors or not.

`estimand` String indicating the estimand to target.

Methods

Public methods:

- `MCATE_cfg$new()`
- `MCATE_cfg$add_moderator()`
- `MCATE_cfg$clone()`

Method `new()`**:** Create a new `MCATE_cfg` object with specified model name and hyperparameters.

Usage:

```
MCATE_cfg$new(cfgs, std_errors = TRUE)
```

Arguments:

`cfgs` Named list from moderator name to a `Model_cfg` object defining how to present that covariate's CATE surface (while marginalizing over all other covariates)

`std_errors` Boolean indicating whether the results should be returned with standard errors or not.

Returns: A new `MCATE_cfg` object.

Examples:

```
MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))
```

Method `add_moderator()`**:** Add a moderator to the `MCATE_cfg` object. This entails defining a configuration for displaying the effect surface for that moderator.

Usage:

```
MCATE_cfg$add_moderator(var_name, cfg)
```

Arguments:

`var_name` The name of the moderator to add (and the name of the column in the dataset).

`cfg` A `Model_cfg` defining how to display the selected moderator's effect surface.

Returns: An updated `MCATE_cfg` object.

Examples:

```
cfg <- MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))
cfg <- cfg$add_moderator("x2", KernelSmooth_cfg$new(neval = 100))
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))

## ------------------------------------------------
## Method `MCATE_cfg$new`
## ------------------------------------------------

MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))

## ------------------------------------------------
## Method `MCATE_cfg$add_moderator`
## ------------------------------------------------

cfg <- MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))
cfg <- cfg$add_moderator("x2", KernelSmooth_cfg$new(neval = 100))
```

Model_cfg *Base Class of Model Configurations*

Description

Model_cfg is the base class from which all other model configurations inherit.

Public fields

model_class The class of the model, required for all classes which inherit from Model_cfg.

Methods

Public methods:

- Model_cfg$new()
- Model_cfg$clone()

Method new(): Create a new Model_cfg object with any necessary parameters.

Usage:

```
Model_cfg$new()
```

Returns: A new Model_cfg object.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Model_data *R6 class to represent data to be used in estimating a model*

Description

R6 class to represent data to be used in estimating a model

R6 class to represent data to be used in estimating a model

Details

This class provides consistent names and interfaces to data which will be used in a supervised regression / classification model.

Public fields

`label` The labels for the eventual model as a vector.

`features` The matrix representation of the data to be used for model fitting. Constructed using `stats::model.matrix`.

`model_frame` The data-frame representation of the data as constructed by `stats::model.frame`.

`split_id` The split identifiers as a vector.

`num_splits` The integer number of splits in the data.

`cluster` A cluster ID as a vector, constructed using the unit identifiers.

`weights` The case-weights as a vector.

Methods

Public methods:

- `Model_data$new()`
- `Model_data$SL_cv_control()`
- `Model_data$clone()`

Method `new()`**:** Creates an R6 object to represent data to be used in a prediction model.

Usage:

```
Model_data$new(data, label_col, ..., .weight_col = NULL)
```

Arguments:

`data` The full dataset to populate the class with.

`label_col` The unquoted name of the column to use as the label in supervised learning models.

`...` The unquoted names of features to use in the model.

`.weight_col` The unquoted name of the column to use as case-weights in subsequent models.

Returns: A `Model_data` object.

Examples:

```
library("dplyr")
df <- dplyr::tibble(
    uid = 1:100,
    x1 = rnorm(100),
    x2 = rnorm(100),
    x3 = sample(4, 100, replace = TRUE)
) %>% dplyr::mutate(
    y = x1 + x2 + x3 + rnorm(100),
    x3 = factor(x3)
)
df <- make_splits(df, uid, .num_splits = 5)
data <- Model_data$new(df, y, x1, x2, x3)
```

Method `SL_cv_control()`**:** A helper function to create the cross-validation options to be used by SuperLearner.

Usage:

```
Model_data$SL_cv_control()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

SuperLearner::SuperLearner.CV.control

Examples

```
## ------------------------------------------------
## Method `Model_data$new`
## ------------------------------------------------

library("dplyr")
df <- dplyr::tibble(
    uid = 1:100,
    x1 = rnorm(100),
    x2 = rnorm(100),
    x3 = sample(4, 100, replace = TRUE)
) %>% dplyr::mutate(
    y = x1 + x2 + x3 + rnorm(100),
    x3 = factor(x3)
)
df <- make_splits(df, uid, .num_splits = 5)
data <- Model_data$new(df, y, x1, x2, x3)
```

---

predict.SL.glmnet.interaction

Prediction for an SL.glmnet object

Description

Prediction for the glmnet wrapper.

Usage

```
## S3 method for class 'SL.glmnet.interaction'
predict(
  object,
  newdata,
  remove_extra_cols = TRUE,
  add_missing_cols = TRUE,
  ...
)
```

Arguments

object	Result object from SL.glmnet
newdata	Dataframe or matrix that will generate predictions.
remove_extra_cols	Remove any extra columns in the new data that were not part of the original model.
add_missing_cols	Add any columns from original data that do not exist in the new data, and set values to 0.
...	Any additional arguments (not used).

See Also

SL.glmnet

---

produce_plugin_estimates

Estimate models of nuisance functions

Description

This takes a dataset with an identified outcome and treatment column along with any number of covariates and appends three columns to the dataset corresponding to an estimate of the conditional expectation of treatment (.pi_hat), along with the conditional expectation of the control and treatment potential outcome surfaces (.mu0_hat and .mu1_hat respectively).

Usage

```
produce_plugin_estimates(data, outcome, treatment, ..., .weights = NULL)
```

Arguments

data	dataframe (already prepared with attach_config and make_splits)
outcome	Unquoted name of the outcome variable.
treatment	Unquoted name of the treatment variable.
...	Unquoted names of covariates to include in the models of the nuisance functions.
.weights	Unquoted name of weights column. If NULL, all analysis will assume weights are all equal to one and sample-based quantities will be returned.

Details

To see an example analysis, read vignette("experimental_analysis") in the context of an experiment, vignette("experimental_analysis") for an observational study, or vignette("methodological_details") for a deeper dive under the hood.

See Also

attach_config(), make_splits(), construct_pseudo_outcomes(), estimate_QoI()

Examples

```
library("dplyr")
if(require("palmerpenguins")) {
data(package = 'palmerpenguins')
penguins$unitid = seq_len(nrow(penguins))
penguins$propensity = rep(0.5, nrow(penguins))
penguins$treatment = rbinom(nrow(penguins), 1, penguins$propensity)
cfg <- basic_config() %>%
add_known_propensity_score("propensity") %>%
add_outcome_model("SL.glm.interaction") %>%
remove_vimp()
attach_config(penguins, cfg) %>%
make_splits(unitid, .num_splits = 4) %>%
produce_plugin_estimates(outcome = body_mass_g, treatment = treatment, species, sex) %>%
construct_pseudo_outcomes(body_mass_g, treatment) %>%
estimate_QoI(species, sex)
}
```

QoI_cfg *Configuration of Quantities of Interest*

Description

QoI_cfg is a configuration class for the Quantities of Interest to be generated by the HTE analysis.

Public fields

mcate A configuration object of type MCATE_cfg of marginal effects to calculate.

pcate A configuration object of type PCATE_cfg of partial effects to calculate.

vimp A configuration object of type VIMP_cfg of variable importance to calculate.

diag A configuration object of type Diagnostics_cfg of model diagnostics to calculate.

ate Logical flag indicating whether an estimate of the ATE should be returned.

predictions Logical flag indicating whether estimates of the CATE for every unit should be returned.

Methods

Public methods:

- QoI_cfg$new()
- QoI_cfg$clone()

Method new(): Create a new QoI_cfg object with specified Quantities of Interest to estimate.

Usage:

```
QoI_cfg$new(
  mcate = NULL,
  pcate = NULL,
  vimp = NULL,
  diag = NULL,
  ate = TRUE,
  predictions = FALSE
)
```

Arguments:

`mcate` A configuration object of type `MCATE_cfg` of marginal effects to calculate.

`pcate` A configuration object of type `PCATE_cfg` of partial effects to calculate.

`vimp` A configuration object of type `VIMP_cfg` of variable importance to calculate.

`diag` A configuration object of type `Diagnostics_cfg` of model diagnostics to calculate.

`ate` A logical flag for whether to calculate the Average Treatment Effect (ATE) or not.

`predictions` A logical flag for whether to return predictions of the CATE for every unit or not.

Returns: A new `Diagnostics_cfg` object.

Examples:

```
mcate_cfg <- MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))
pcate_cfg <- PCATE_cfg$new(
   cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)),
   model_covariates = c("x1", "x2", "x3"),
   num_mc_samples = list(x1 = 100)
)
vimp_cfg <- VIMP_cfg$new()
diag_cfg <- Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE"),
   ps = c("SL_risk", "SL_coefs", "AUC")
)
QoI_cfg$new(
    mcate = mcate_cfg,
    pcate = pcate_cfg,
    vimp = vimp_cfg,
    diag = diag_cfg
)
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
mcate_cfg <- MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))
```

```
pcate_cfg <- PCATE_cfg$new(
   cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)),
   model_covariates = c("x1", "x2", "x3"),
   num_mc_samples = list(x1 = 100)
)
vimp_cfg <- VIMP_cfg$new()
diag_cfg <- Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE"),
   ps = c("SL_risk", "SL_coefs", "AUC")
)
QoI_cfg$new(
    mcate = mcate_cfg,
    pcate = pcate_cfg,
    vimp = vimp_cfg,
    diag = diag_cfg
)

## ------------------------------------------------
## Method `QoI_cfg$new`
## ------------------------------------------------

mcate_cfg <- MCATE_cfg$new(cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)))
pcate_cfg <- PCATE_cfg$new(
   cfgs = list(x1 = KernelSmooth_cfg$new(neval = 100)),
   model_covariates = c("x1", "x2", "x3"),
   num_mc_samples = list(x1 = 100)
)
vimp_cfg <- VIMP_cfg$new()
diag_cfg <- Diagnostics_cfg$new(
   outcome = c("SL_risk", "SL_coefs", "MSE"),
   ps = c("SL_risk", "SL_coefs", "AUC")
)
QoI_cfg$new(
    mcate = mcate_cfg,
    pcate = pcate_cfg,
    vimp = vimp_cfg,
    diag = diag_cfg
)
```

`remove_vimp` *Removes variable importance information*

Description

This removes the variable importance quantity of interest from an `HTE_cfg`.

Usage

```
remove_vimp(hte_cfg)
```

Arguments

hte_cfg	HTE_cfg object to update.

Value

Updated HTE_cfg object

Examples

```
library("dplyr")
basic_config() %>%
   remove_vimp() -> hte_cfg
```

SL.glmnet.interaction *Elastic net regression with pairwise interactions*

Description

Penalized regression using elastic net. Alpha = 0 corresponds to ridge regression and alpha = 1 corresponds to Lasso. Included in the model are pairwise interactions between covariates.

See vignette("glmnet_beta", package = "glmnet") for a nice tutorial on glmnet.

Usage

```
SL.glmnet.interaction(
  Y,
  X,
  newX,
  family,
  obsWeights,
  id,
  alpha = 1,
  nfolds = 10,
  nlambda = 100,
  useMin = TRUE,
  loss = "deviance",
  ...
)
```

Arguments

Y	Outcome variable
X	Covariate dataframe
newX	Dataframe to predict the outcome

family	"gaussian" for regression, "binomial" for binary classification. Untested options: "multinomial" for multiple classification or "mgaussian" for multiple response, "poisson" for non-negative outcome with proportional mean and variance, "cox".
obsWeights	Optional observation-level weights
id	Optional id to group observations from the same unit (not used currently).
alpha	Elastic net mixing parameter, range [0, 1]. 0 = ridge regression and 1 = lasso.
nfolds	Number of folds for internal cross-validation to optimize lambda.
nlambda	Number of lambda values to check, recommended to be 100 or more.
useMin	If TRUE use lambda that minimizes risk, otherwise use 1 standard-error rule which chooses a higher penalty with performance within one standard error of the minimum (see Breiman et al. 1984 on CART for background).
loss	Loss function, can be "deviance", "mse", or "mae". If family = binomial can also be "auc" or "class" (misclassification error).
...	Any additional arguments are passed through to cv.glmnet.

SLEnsemble_cfg *Configuration for a SuperLearner Ensemble*

Description

SLEnsemble_cfg is a configuration class for estimation of a model using an ensemble of models using SuperLearner.

Super class

tidyhte::Model_cfg -> SLEnsemble_cfg

Public fields

cvControl A list of parameters for controlling the cross-validation used in SuperLearner.

SL.library A vector of the names of learners to include in the SuperLearner ensemble.

SL.env An environment containing all of the programmatically generated learners to be included in the SuperLearner ensemble.

family stats::family object to determine how SuperLearner should be fitted.

model_class The class of the model, required for all classes which inherit from Model_cfg.

Methods

Public methods:

- SLEnsemble_cfg$new()
- SLEnsemble_cfg$add_sublearner()
- SLEnsemble_cfg$clone()

Method `new()`**:** Create a new `SLEnsemble_cfg` object with specified settings.

Usage:

```
SLEnsemble_cfg$new(
  cvControl = NULL,
  learner_cfgs = NULL,
  family = stats::gaussian()
)
```

Arguments:

`cvControl` A list of parameters for controlling the cross-validation used in SuperLearner. For more details, see `SuperLearner::SuperLearner.CV.control`.

`learner_cfgs` A list of `SLLearner_cfg` objects.

`family` `stats::family` object to determine how SuperLearner should be fitted.

Returns: A new `SLEnsemble_cfg` object.

Examples:

```
SLEnsemble_cfg$new(
learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)
```

Method `add_sublearner()`**:** Adds a model (or class of models) to the SuperLearner ensemble. If hyperparameter values are specified, this method will add a learner for every element in the cross-product of provided hyperparameter values.

Usage:

```
SLEnsemble_cfg$add_sublearner(learner_name, hps = NULL)
```

Arguments:

`learner_name` Possible values use `SuperLearner` naming conventions. A full list is available with `SuperLearner::listWrappers("SL")`

`hps` A named list of hyper-parameters. Every element of the cross-product of these hyper-parameters will be included in the ensemble. cfg <- SLEnsemble_cfg$new(learner_cfgs = list(SLLearner_cfg$new("SL.glm"))) cfg <- cfg$add_sublearner("SL.gam", list(deg.gam = c(2, 3)))

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
SLEnsemble_cfg$new(
learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)

## ------------------------------------------------
## Method `SLEnsemble_cfg$new`
## ------------------------------------------------

SLEnsemble_cfg$new(
learner_cfgs = list(SLLearner_cfg$new("SL.glm"), SLLearner_cfg$new("SL.gam"))
)
```

SLLearner_cfg *Configuration of SuperLearner Submodel*

Description

SLLearner_cfg is a configuration class for a single sublearner to be included in SuperLearner. By constructing with a named list of hyperparameters, this configuration allows distinct submodels for each unique combination of hyperparameters. To understand what models and hyperparameters are available, examine the methods listed in SuperLearner::listWrappers("SL").

Public fields

model_name The name of the model as passed to SuperLearner through the SL.library parameter.

hyperparameters Named list from hyperparameter name to a vector of values that should be swept over.

Methods

Public methods:

- SLLearner_cfg$new()
- SLLearner_cfg$clone()

Method new(): Create a new SLLearner_cfg object with specified model name and hyperparameters.

Usage:

```
SLLearner_cfg$new(model_name, hp = NULL)
```

Arguments:

model_name The name of the model as passed to SuperLearner through the SL.library parameter.

hp Named list from hyperparameter name to a vector of values that should be swept over. Hyperparameters not included in this list are left at their SuperLearner default values.

Returns: A new `SLLearner_cfg` object.

Examples:

```
SLLearner_cfg$new("SL.glm")
SLLearner_cfg$new("SL.gam", list(deg.gam = c(2, 3)))
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SLLearner_cfg$new`
## ------------------------------------------------

SLLearner_cfg$new("SL.glm")
SLLearner_cfg$new("SL.gam", list(deg.gam = c(2, 3)))
```

`Stratified_cfg` *Configuration for a Stratification Estimator*

Description

`Stratified_cfg` is a configuration class for stratifying a covariate and calculating statistics within each cell.

Super class

`tidyhte::Model_cfg` -> `Stratified_cfg`

Public fields

`model_class` The class of the model, required for all classes which inherit from `Model_cfg`.

`covariate` The name of the column in the dataset which corresponds to the covariate on which to stratify.

Methods

Public methods:

- `Stratified_cfg$new()`
- `Stratified_cfg$clone()`

Method `new()`**:** Create a new `Stratified_cfg` object with specified number of evaluation points.

Usage:

```
Stratified_cfg$new(covariate)
```

Arguments:

`covariate` The name of the column in the dataset which corresponds to the covariate on which to stratify.

Returns: A new `Stratified_cfg` object.

Examples:

```
Stratified_cfg$new(covariate = "test_covariate")
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Stratified_cfg$new`
## ------------------------------------------------

Stratified_cfg$new(covariate = "test_covariate")
```

`VIMP_cfg` *Configuration of Variable Importance*

Description

`VIMP_cfg` is a configuration class for estimating a variable importance measure across all moderators. This provides a meaningful measure of which moderators explain the most of the CATE surface.

Public fields

`estimand` String indicating the estimand to target.

`sample_splitting` Logical indicating whether to use sample splitting in the calculation of variable importance.

`linear` Logical indicating whether the variable importance assuming a linear model should be estimated.

Methods

Public methods:

- VIMP_cfg$new()
- VIMP_cfg$clone()

Method `new()`**:** Create a new `VIMP_cfg` object with specified model configuration.

Usage:

```
VIMP_cfg$new(sample_splitting = TRUE, linear_only = FALSE)
```

Arguments:

`sample_splitting` Logical indicating whether to use sample splitting in the calculation of variable importance. Choosing not to use sample splitting means that inference will only be valid for moderators with non-null importance.

`linear_only` Logical indicating whether the variable importance should use only a single linear-only model. Variable importance measure will only be consistent for the population quantity if the true model of pseudo-outcomes is linear.

Returns: A new `VIMP_cfg` object.

Examples:

```
VIMP_cfg$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

- Williamson, B. D., Gilbert, P. B., Carone, M., & Simon, N. (2021). Nonparametric variable importance assessment using machine learning techniques. Biometrics, 77(1), 9-22.
- Williamson, B. D., Gilbert, P. B., Simon, N. R., & Carone, M. (2021). A general framework for inference on algorithm-agnostic variable importance. Journal of the American Statistical Association, 1-14.

Examples

```
VIMP_cfg$new()

## ------------------------------------------------
## Method `VIMP_cfg$new`
## ------------------------------------------------

VIMP_cfg$new()
```

Index