Package 'GenericML'

July 21, 2025

Title Generic Machine Learning Inference

Version 0.2.2

Author Max Welz [aut, cre] (ORCID: <https://orcid.org/0000-0003-2945-1860>),
Andreas Alfons [aut] (ORCID: <https://orcid.org/0000-0002-2513-3788>),
Mert Demirer [aut],
Victor Chernozhukov [aut]

Maintainer Max Welz <welz@ese.eur.nl>

Description Generic Machine Learning Inference on heterogeneous treatment effects in randomized experiments as proposed in Chernozhukov, Demirer, Duflo and Fernández-Val (2020) <doi:10.48550/arXiv.1712.04802>. This package's workhorse is the 'mlr3' framework of Lang et al. (2019) <doi:10.21105/joss.01903>, which enables the specification of a wide variety of machine learners. The main functionality, GenericML(), runs Algorithm 1 in Chernozhukov, Demirer, Duflo and Fernández-Val (2020) <doi:10.48550/arXiv.1712.04802> for a suite of user-specified machine learners. All steps in the algorithm are customizable via setup functions. Methods for printing and plotting are available for objects returned by GenericML(). Parallel computing is supported.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.0

URL https://github.com/mwelz/GenericML/

BugReports https://github.com/mwelz/GenericML/issues/

Depends ggplot2, mlr3, mlr3learners

Imports sandwich, lmtest, splitstackshape, stats, parallel, abind

Suggests glmnet, ranger, rpart, e1071, xgboost, kknn, DiceKriging, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Repository CRAN

Date/Publication 2022-06-18 08:00:09 UTC

Contents

`BLP` *Performs BLP regression*

Description

Performs the linear regression for the Best Linear Predictor (BLP) procedure.

Usage

```
BLP(
  Y,
  D,
  propensity_scores,
  proxy_BCA,
```

```
  proxy_CATE,
  HT = FALSE,
  X1_control = setup_X1(),
  vcov_control = setup_vcov(),
  significance_level = 0.05
)
```

Arguments

Y	A numeric vector containing the response variable.
D	A binary vector of treatment assignment. Value one denotes assignment to the treatment group and value zero assignment to the control group.
propensity_scores	A numeric vector of propensity scores. We recommend to use the estimates of a "propensity_score" object.
proxy_BCA	A numeric vector of proxy baseline conditional average (BCA) estimates. We recommend to use the estimates of a "proxy_BCA" object.
proxy_CATE	A numeric vector of proxy conditional average treatment effect (CATE) estimates. We recommend to use the estimates of a "proxy_CATE" object.
HT	Logical. If TRUE, a Horvitz-Thompson (HT) transformation is applied (BLP2 in the paper). Default is FALSE.
X1_control	Specifies the design matrix X_1 in the regression. Must be an object of class "setup_X1". See the documentation of setup_X1() for details.
vcov_control	Specifies the covariance matrix estimator. Must be an object of class "setup_vcov". See the documentation of setup_vcov() for details.
significance_level	Significance level. Default is 0.05.

Value

An object of class "BLP", consisting of the following components:

generic_targets A matrix of the inferential results on the BLP generic targets.

coefficients An object of class "coeftest", contains the coefficients of the BLP regression.

lm An object of class "lm" used to fit the linear regression model.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

See Also

setup_X1(), setup_diff(), setup_vcov(), propensity_score(), proxy_BCA(), proxy_CATE()

Examples

```
## generate data
set.seed(1)
n  <- 150                       # number of observations
p  <- 5                         # number of covariates
D  <- rbinom(n, 1, 0.5)         # random treatment assignment
Y  <- runif(n)                  # outcome variable
propensity_scores <- rep(0.5, n) # propensity scores
proxy_BCA         <- runif(n)    # proxy BCA estimates
proxy_CATE        <- runif(n)    # proxy CATE estimates

## perform BLP
BLP(Y, D, propensity_scores, proxy_BCA, proxy_CATE)
```

CLAN *Performs CLAN*

Description

Performs Classification Analysis (CLAN) on all variables in a design matrix.

Usage

```
CLAN(
  Z_CLAN,
  membership,
  equal_variances = FALSE,
  diff = setup_diff(),
  significance_level = 0.05
)
```

Arguments

Z_CLAN	A numeric matrix holding variables on which classification analysis (CLAN) shall be performed. CLAN will be performed on each column of the matrix.
membership	A logical matrix that indicates the group membership of each observation in Z_CLAN. Needs to be of type "quantile_group". Typically, the grouping is based on CATE estimates, which are for instance returned by proxy_CATE.
equal_variances	If TRUE, then all within-group variances of the CLAN groups are assumed to be equal. Default is FALSE. This specification is required for heteroskedasticity-robust variance estimation on the difference of two CLAN generic targets (i.e. variance of the difference of two means). If TRUE (corresponds to homoskedasticity assumption), the pooled variance is used. If FALSE (heteroskedasticity), the variance of Welch's t-test is used.

`diff` Specifies the generic targets of CLAN. Must be an object of class `"setup_diff"`. See the documentation of `setup_diff()` for details.

`significance_level` Significance level. Default is 0.05.

Value

An object of the class `"CLAN"`, consisting of the following components:

`generic_targets` A list of result matrices for each variable in `Z_CLAN`. Each matrix contains inferential results on the CLAN generic targets.

`coefficients` A matrix of point estimates of each CLAN generic target parameter.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

See Also

`quantile_group()`, `setup_diff()`

Examples

```
## generate data
set.seed(1)
n  <- 150                              # number of observations
p  <- 5                                # number of covariates
Z_CLAN <- matrix(runif(n*p), n, p)     # design matrix to perform CLAN on
membership <- quantile_group(rnorm(n)) # group membership

## perform CLAN
CLAN(Z_CLAN, membership)
```

GATES *Performs GATES regression*

Description

Performs the linear regression for the Group Average Treatments Effects (GATES) procedure.

Usage

```
GATES(
  Y,
  D,
  propensity_scores,
  proxy_BCA,
  proxy_CATE,
  membership,
  HT = FALSE,
  X1_control = setup_X1(),
  vcov_control = setup_vcov(),
  diff = setup_diff(),
  significance_level = 0.05
)
```

Arguments

Y	A numeric vector containing the response variable.
D	A binary vector of treatment assignment. Value one denotes assignment to the treatment group and value zero assignment to the control group.
propensity_scores	A numeric vector of propensity scores. We recommend to use the estimates of a "propensity_score" object.
proxy_BCA	A numeric vector of proxy baseline conditional average (BCA) estimates. We recommend to use the estimates of a "proxy_BCA" object.
proxy_CATE	A numeric vector of proxy conditional average treatment effect (CATE) estimates. We recommend to use the estimates of a "proxy_CATE" object.
membership	A logical matrix that indicates the group membership of each observation in Z_CLAN. Needs to be of type "quantile_group". Typically, the grouping is based on CATE estimates, which are for instance returned by proxy_CATE().
HT	Logical. If TRUE, a Horvitz-Thompson (HT) transformation is applied (GATES2 in the paper). Default is FALSE.
X1_control	Specifies the design matrix X_1 in the regression. Must be an object of class "setup_X1". See the documentation of setup_X1() for details.
vcov_control	Specifies the covariance matrix estimator. Must be an object of class "setup_vcov". See the documentation of setup_vcov() for details.
diff	Specifies the generic targets of CLAN. Must be an object of class "setup_diff". See the documentation of setup_diff() for details.
significance_level	Significance level. Default is 0.05.

Value

An object of class "GATES", consisting of the following components:

generic_targets	A matrix of the inferential results on the GATES generic targets.

`coefficients` An object of class "coeftest", contains the coefficients of the GATES regression.

`lm` An object of class "lm" used to fit the linear regression model.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

See Also

setup_X1(), setup_diff(), setup_vcov(), propensity_score(), proxy_BCA(), proxy_CATE()

Examples

```
## generate data
set.seed(1)
n  <- 150                                # number of observations
p  <- 5                                  # number of covariates
D  <- rbinom(n, 1, 0.5)                  # random treatment assignment
Y  <- runif(n)                           # outcome variable
propensity_scores <- rep(0.5, n)         # propensity scores
proxy_BCA         <- runif(n)            # proxy BCA estimates
proxy_CATE        <- runif(n)            # proxy CATE estimates
membership <- quantile_group(proxy_CATE) # group membership

## perform GATES
GATES(Y, D, propensity_scores, proxy_BCA, proxy_CATE, membership)
```

`GenericML` *Generic Machine Learning Inference*

Description

Performs generic machine learning inference on heterogeneous treatment effects as in Chernozhukov, Demirer, Duflo and Fernández-Val (2020) with user-specified machine learning methods. Intended for randomized experiments.

Usage

```
GenericML(
  Z,
  D,
  Y,
  learners_GenericML,
  learner_propensity_score = "constant",
  num_splits = 100,
```

```
  Z_CLAN = NULL,
  HT = FALSE,
  quantile_cutoffs = c(0.25, 0.5, 0.75),
  X1_BLP = setup_X1(),
  X1_GATES = setup_X1(),
  diff_GATES = setup_diff(),
  diff_CLAN = setup_diff(),
  vcov_BLP = setup_vcov(),
  vcov_GATES = setup_vcov(),
  equal_variances_CLAN = FALSE,
  prop_aux = 0.5,
  stratify = setup_stratify(),
  significance_level = 0.05,
  min_variation = 1e-05,
  parallel = FALSE,
  num_cores = parallel::detectCores(),
  seed = NULL,
  store_learners = FALSE,
  store_splits = TRUE
)
```

Arguments

Z	A numeric design matrix that holds the covariates in its columns.
D	A binary vector of treatment assignment. Value one denotes assignment to the treatment group and value zero assignment to the control group.
Y	A numeric vector containing the response variable.
learners_GenericML	A character vector specifying the machine learners to be used for estimating the baseline conditional average (BCA) and conditional average treatment effect (CATE). Either `'lasso'`, `'random_forest'`, `'tree'`, or a custom learner specified with `mlr3` syntax. In the latter case, do *not* specify in the `mlr3` syntax specification if the learner is a regression learner or classification learner. Example: `'mlr3::lrn("ranger", num.trees = 100)'` for a random forest learner with 100 trees. Note that this is a string and the absence of the `classif.` or `regr.` keywords. See https://mlr3learners.mlr-org.com for a list of `mlr3` learners.
learner_propensity_score	The estimator of the propensity scores. Either a numeric vector (which is then taken as estimates of the propensity scores) or a string specifying the estimator. In the latter case, the string must either be equal to `'constant'` (estimates the propensity scores by `mean(D)`), `'lasso'`, `'random_forest'`, `'tree'`, or `mlr3` syntax. Note that in case of `mlr3` syntax, do *not* specify if the learner is a regression learner or classification learner. Example: `'mlr3::lrn("ranger", num.trees = 100)'` for a random forest learner with 100 trees. Note that this is a string and the absence of the `classif.` or `regr.` keywords. See https://mlr3learners.mlr-org.com for a list of `mlr3` learners.

num_splits — Number of sample splits. Default is 100. Must be larger than one. If you want to run GenericML on a single split, please use GenericML_single().

Z_CLAN — A numeric matrix holding variables on which classification analysis (CLAN) shall be performed. CLAN will be performed on each column of the matrix. If NULL (default), then Z_CLAN = Z, i.e. CLAN is performed for all variables in Z.

HT — Logical. If TRUE, a Horvitz-Thompson (HT) transformation is applied in the BLP and GATES regressions. Default is FALSE.

quantile_cutoffs — The cutoff points of the quantiles that shall be used for GATES grouping. Default is c(0.25, 0.5, 0.75), which corresponds to the four quartiles.

X1_BLP — Specifies the design matrix X_1 in the regression. Must be an object of class "setup_X1". See the documentation of setup_X1() for details.

X1_GATES — Same as X1_BLP, just for the GATES regression.

diff_GATES — Specifies the generic targets of GATES. Must be an object of class "setup_diff". See the documentation of setup_diff() for details.

diff_CLAN — Same as diff_GATES, just for the CLAN generic targets.

vcov_BLP — Specifies the covariance matrix estimator in the BLP regression. Must be an object of class "setup_vcov". See the documentation of setup_vcov() for details.

vcov_GATES — Same as vcov_BLP, just for the GATES regression.

equal_variances_CLAN — Logical. If TRUE, then all within-group variances of the CLAN groups are assumed to be equal. Default is FALSE. This specification is required for heteroskedasticity-robust variance estimation on the difference of two CLAN generic targets (i.e. variance of the difference of two means). If TRUE (corresponds to homoskedasticity assumption), the pooled variance is used. If FALSE (heteroskedasticity), the variance of Welch's t-test is used.

prop_aux — Proportion of samples that shall be in the auxiliary set in case of random sample splitting. Default is 0.5. The number of samples in the auxiliary set will be equal to floor(prop_aux * length(Y)). If the data set is large, you can save computing time by choosing prop_aux to be smaller than 0.5. In case of stratified sampling (controlled through the argument stratify via setup_stratify()), prop_aux does not have an effect, and the number of samples in the auxiliary set is specified via setup_stratify().

stratify — A list that specifies whether or not stratified sample splitting shall be performed. It is recommended to use the returned object of setup_stratify() as this list. See the documentation of setup_stratify() for details.

significance_level — Significance level for VEIN. Default is 0.05.

min_variation — Specifies a threshold for the minimum variation of the BCA/CATE predictions. If the variation of a BCA/CATE prediction falls below this threshold, random noise with distribution $N(0, var(Y)/20)$ is added to it. Default is 1e-05.

parallel — Logical. If TRUE, parallel computing will be used. Default is FALSE. On Unix systems, this will be done via forking (shared memory across threads). On non-Unix systems, this will be done through parallel socket clusters.

num_cores	Number of cores to be used in parallelization (if applicable). Default is the number of cores of the user's machine.
seed	Random seed. Default is NULL for no random seeding.
store_learners	Logical. If TRUE, all intermediate results of the learners will be stored. That is, for each learner and each split, all BCA and CATE predictions as well as all BLP, GATES, CLAN, and Λ estimates will be stored. Default is FALSE.
store_splits	Logical. If TRUE (default), the sample splits will be stored.

Details

The specifications "lasso", "random_forest", and "tree" in learners_GenericML and learner_propensity_score correspond to the following mlr3 specifications (we omit the keywords classif. and regr.). "lasso" is a cross-validated Lasso estimator, which corresponds to 'mlr3::lrn("cv_glmnet", s = "lambda.min", alpha = 1)'. "random_forest" is a random forest with 500 trees, which corresponds to 'mlr3::lrn("ranger", num.trees = 500)'. "tree" is a tree learner, which corresponds to 'mlr3::lrn("rpart")'. **Warning:** GenericML() can be quite memory-intensive, in particular when the data set is large. To alleviate memory usage, consider setting store_learners = FALSE, choosing a low number of cores via num_cores (at the expense of longer computing time), setting prop_aux to a value smaller than the default of 0.5, or using GenericML_combine().

Value

An object of class "GenericML". On this object, we recommend to use the accessor functions get_BLP(), get_GATES(), and get_CLAN() to extract the results of the analyses of BLP, GATES, and CLAN, respectively. An object of class "GenericML" contains the following components:

VEIN A list containing two sub-lists called best_learners and all_learners, respectively. Each of these two sub-lists contains the inferential VEIN results on the generic targets of the BLP, GATES, and CLAN analyses. all_learners does this for all learners specified in the argument learners_GenericML, best_learners only for the corresponding best learners. Which learner is best for which analysis is assessed by the Λ criteria discussed in Sections 5.2 and 5.3 of the paper.

best A list containing information on the evaluation of which learner is the best for which analysis. Contains four components. The first three contain the name of the best learner for BLP, GATES, and CLAN, respectively. The fourth component, overview, contains the two Λ criteria used to determine the best learners (discussed in Sections 5.2 and 5.3 of the paper).

propensity_scores The propensity score estimates as well as the "mlr3" objects used to estimate them (if mlr3 was used for estimation).

GenericML_single Only nonempty if store_learners = TRUE. Contains all intermediate results of each learners for each split. That is, for a given learner (first level of the list) and split (second level), objects of classes "BLP", "GATES", "CLAN", "proxy_BCA", "proxy_CATE" as well as the Λ criteria ("best")) are listed, which were computed with the given learner and split.

splits Only nonempty if store_splits = TRUE. Contains a character matrix of dimension length(Y) by num_splits. Contains the group membership (main or auxiliary) of each observation (rows) in each split (columns). "M" denotes the main set, "A" the auxiliary set.

`generic_targets` A list of generic target estimates for each learner. More specifically, each component is a list of the generic target estimates pertaining to the BLP, GATES, and CLAN analyses. Each of those lists contains a three-dimensional array containing the generic targets of a single learner for all sample splits (except CLAN where there is one more layer of lists).

`arguments` A list of arguments used in the function call.

Note

In an earlier development version, Lucas Kitzmueller alerted us to several minor bugs and proposed fixes. Many thanks to him!

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

Lang M., Binder M., Richter J., Schratz P., Pfisterer F., Coors S., Au Q., Casalicchio G., Kotthoff L., Bischl B. (2019). "mlr3: A Modern Object-Oriented Machine Learning Framework in R." *Journal of Open Source Software*, **4**(44), 1903. doi: 10.21105/joss.01903.

See Also

`plot.GenericML()` `print.GenericML()` `get_BLP()`, `get_GATES()`, `get_CLAN()`, `setup_X1()`, `setup_diff()`, `setup_vcov()`, `setup_stratify()`, `GenericML_single()`, `GenericML_combine()`

Examples

```
if (require("glmnet") && require("ranger")) {

## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome

## column names of Z
colnames(Z) <- paste0("V", 1:p)

## specify learners
learners <- c("lasso", "mlr3::lrn('ranger', num.trees = 10)")

## glmnet v4.1.3 isn't supported on Solaris, so skip Lasso in this case
if(Sys.info()["sysname"] == "SunOS") learners <- learners[-1]

## specify quantile cutoffs (the 4 quartile groups here)
quantile_cutoffs <- c(0.25, 0.5, 0.75)
```

```
## specify the differenced generic targets of GATES and CLAN
# use G4-G1, G4-G2, G4-G3 as differenced generic targets in GATES
diff_GATES <- setup_diff(subtract_from = "most",
                         subtracted = c(1,2,3))
# use G1-G3, G1-G2 as differenced generic targets in CLAN
diff_CLAN  <- setup_diff(subtract_from = "least",
                          subtracted = c(3,2))

## perform generic ML inference
# small number of splits to keep computation time low
x <- GenericML(Z, D, Y, learners, num_splits = 2,
               quantile_cutoffs = quantile_cutoffs,
               diff_GATES = diff_GATES,
               diff_CLAN = diff_CLAN,
               parallel = FALSE)

## access BLP generic targets for best learner and make plot
get_BLP(x, plot = TRUE)

## access GATES generic targets for best learner and make plot
get_GATES(x, plot = TRUE)

## access CLAN generic targets for "V1" & best learner and make plot
get_CLAN(x, variable = "V1", plot = TRUE)

}
```

`GenericML_combine` *Combine several GenericML objects*

Description

This function combines multiple "GenericML" objects into one "GenericML" object. Combining several "GenericML" objects can be useful when you cannot run GenericML() for sufficiently many splits due to memory constraints. In this case, you may run GenericML() multiple times with only a small number of sample splits each and combine the returned "GenericML" objects into one GenericML object with this function.

Usage

```
GenericML_combine(x)
```

Arguments

x	A list of "GenericML" objects, as returned by the function GenericML().

Details

To ensure consistency of the estimates, all "GenericML" objects in the list x must have the exact same parameter specifications in their original call to GenericML(), except for the parameters num_splits, parallel, num_cores, seed, and store_learners (i.e. these arguments may vary between the "GenericML" objects in the list x). An error will be thrown if this is not satisfied.

Value

A"GenericML" object as returned by GenericML(). In the arguments component of this object, the objects parallel, num_cores, seed, and store_learners are set to NULL as these might differ between the individual GenericML objects in x. Moreover, the propensity_scores component of the returned object is taken from the first "GenericML" object in x.

See Also

GenericML()

Examples

```
if (require("glmnet") && require("ranger")) {

## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome

## column names of Z
colnames(Z) <- paste0("V", 1:p)

## specify learners
learners <- c("lasso", "mlr3::lrn('ranger', num.trees = 10)")

## glmnet v4.1.3 isn't supported on Solaris, so skip Lasso in this case
if(Sys.info()["sysname"] == "SunOS") learners <- learners[-1]

## call GenericML three times and store the returned objects in a list x
x <- lapply(1:3, function(...) GenericML(Z, D, Y,
                                learners, num_splits = 2,
                                parallel = FALSE))

## combine the objects in x into one GenericML object
genML <- GenericML_combine(x)

## you can use all methods of GenericML objects on the combined object, for instance accessors:
get_BLP(genML, plot = TRUE)
}
```

GenericML_single *Single iteration of the GenericML algorithm*

Description

Performs generic ML inference for a single learning technique and a given split of the data. Can be seen as a single iteration of Algorithm 1 in the paper.

Usage

```
GenericML_single(
  Z,
  D,
  Y,
  learner,
  propensity_scores,
  M_set,
  A_set = setdiff(1:length(Y), M_set),
  Z_CLAN = NULL,
  HT = FALSE,
  quantile_cutoffs = c(0.25, 0.5, 0.75),
  X1_BLP = setup_X1(),
  X1_GATES = setup_X1(),
  diff_GATES = setup_diff(),
  diff_CLAN = setup_diff(),
  vcov_BLP = setup_vcov(),
  vcov_GATES = setup_vcov(),
  equal_variances_CLAN = FALSE,
  significance_level = 0.05,
  min_variation = 1e-05
)
```

Arguments

Z	A numeric design matrix that holds the covariates in its columns.
D	A binary vector of treatment assignment. Value one denotes assignment to the treatment group and value zero assignment to the control group.
Y	A numeric vector containing the response variable.
learner	A character specifying the machine learner to be used for estimating the baseline conditional average (BCA) and conditional average treatment effect (CATE). Either `'lasso'`, `'random_forest'`, `'tree'`, or a custom learner specified with `mlr3` syntax. In the latter case, do *not* specify in the `mlr3` syntax specification if the learner is a regression learner or classification learner. Example: `'mlr3::lrn("ranger", num.trees = 100)'` for a random forest learner with

100 trees. Note that this is a string and the absence of the `classif.` or `regr.` keywords. See https://mlr3learners.mlr-org.com for a list of `mlr3` learners.

`propensity_scores`
: A numeric vector of propensity score estimates.

`M_set`
: A numerical vector of indices of observations in the main sample.

`A_set`
: A numerical vector of indices of observations in the auxiliary sample. Default is complementary set to `M_set`.

`Z_CLAN`
: A numeric matrix holding variables on which classification analysis (CLAN) shall be performed. CLAN will be performed on each column of the matrix. If `NULL` (default), then `Z_CLAN = Z`, i.e. CLAN is performed for all variables in `Z`.

`HT`
: Logical. If `TRUE`, a Horvitz-Thompson (HT) transformation is applied in the BLP and GATES regressions. Default is `FALSE`.

`quantile_cutoffs`
: The cutoff points of the quantiles that shall be used for GATES grouping. Default is `c(0.25, 0.5, 0.75)`, which corresponds to the four quartiles.

`X1_BLP`
: Specifies the design matrix X_1 in the regression. Must be an object of class `"setup_X1"`. See the documentation of `setup_X1()` for details.

`X1_GATES`
: Same as `X1_BLP`, just for the GATES regression.

`diff_GATES`
: Specifies the generic targets of GATES. Must be an object of class `"setup_diff"`. See the documentation of `setup_diff()` for details.

`diff_CLAN`
: Same as `diff_GATES`, just for the CLAN generic targets.

`vcov_BLP`
: Specifies the covariance matrix estimator in the BLP regression. Must be an object of class `"setup_vcov"`. See the documentation of `setup_vcov()` for details.

`vcov_GATES`
: Same as `vcov_BLP`, just for the GATES regression.

`equal_variances_CLAN`
: Logical. If `TRUE`, then all within-group variances of the CLAN groups are assumed to be equal. Default is `FALSE`. This specification is required for heteroskedasticity-robust variance estimation on the difference of two CLAN generic targets (i.e. variance of the difference of two means). If `TRUE` (corresponds to homoskedasticity assumption), the pooled variance is used. If `FALSE` (heteroskedasticity), the variance of Welch's t-test is used.

`significance_level`
: Significance level for VEIN. Default is 0.05.

`min_variation`
: Specifies a threshold for the minimum variation of the BCA/CATE predictions. If the variation of a BCA/CATE prediction falls below this threshold, random noise with distribution $N(0, var(Y)/20)$ is added to it. Default is `1e-05`.

Details

The specifications `"lasso"`, `"random_forest"`, and `"tree"` in `learner` correspond to the following `mlr3` specifications (we omit the keywords `classif.` and `regr.`). `"lasso"` is a cross-validated Lasso estimator, which corresponds to `'mlr3::lrn("cv_glmnet", s = "lambda.min", alpha = 1)'`. `"random_forest"` is a random forest with 500 trees, which corresponds to `'mlr3::lrn("ranger", num.trees = 500)'`. `"tree"` is a tree learner, which corresponds to `'mlr3::lrn("rpart")'`.

Value

A list with the following components:

`BLP` An object of class "`BLP`".

`GATES` An object of class "`GATES`".

`CLAN` An object of class "`CLAN`".

`proxy_BCA` An object of class "`proxy_BCA`".

`proxy_CATE` An object of class "`proxy_CATE`".

`best` Estimates of the Λ parameters for finding the best learner. Returned by `lambda_parameters()`.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

Lang M., Binder M., Richter J., Schratz P., Pfisterer F., Coors S., Au Q., Casalicchio G., Kotthoff L., Bischl B. (2019). "mlr3: A Modern Object-Oriented Machine Learning Framework in R." *Journal of Open Source Software*, **4**(44), 1903. doi: 10.21105/joss.01903.

See Also

`GenericML()`

Examples

```
if(require("ranger")){
## generate data
set.seed(1)
n  <- 150                       # number of observations
p  <- 5                         # number of covariates
Z  <- matrix(runif(n*p), n, p)  # design matrix
D  <- rbinom(n, 1, 0.5)         # random treatment assignment
Y  <- runif(n)                  # outcome variable
propensity_scores <- rep(0.5, n) # propensity scores
M_set <- sample(1:n, size = n/2) # main set

## specify learner
learner <- "mlr3::lrn('ranger', num.trees = 10)"

## run single GenericML iteration
GenericML_single(Z, D, Y, learner, propensity_scores, M_set)
}
```

get_best *Accessor function for the best learner estimates*

Description

The best learner is determined by maximizing the criteria Λ and $\bar{\Lambda}$, see Sections 5.2 and 5.3 of the paper. This function accesses the estimates of these two criteria,

Usage

```
get_best(x)
```

Arguments

x	An object of the class "GenericML", as returned by the function GenericML().

Value

An object of class "best", which consists of the following components:

BLP A string holding the name of the best learner for a BLP analysis.

GATES A string holding the name of the best learner for a GATES analysis.

CLAN A string holding the name of the best learner for a CLAN analysis (same learner as in GATES).

overview A numeric matrix of the estimates of the performance measures Λ and $\bar{\Lambda}$ for each learner.

See Also

GenericML(), get_BLP(), get_GATES(), get_CLAN()

Examples

```
if(require("rpart") && require("ranger")){
## generate data
set.seed(1)
n  <- 150                                # number of observations
p  <- 5                                  # number of covariates
D  <- rbinom(n, 1, 0.5)                  # random treatment assignment
Z  <- matrix(runif(n*p), n, p)           # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                             # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)             # observed outcome

## column names of Z
colnames(Z) <- paste0("V", 1:p)

## specify learners
learners <- c("tree", "mlr3::lrn('ranger', num.trees = 10)")
```

```
## perform generic ML inference
# small number of splits to keep computation time low
x <- GenericML(Z, D, Y, learners, num_splits = 2,
               parallel = FALSE)

## access best learner
get_best(x)

## access BLP generic targets for best learner w/o plot
get_BLP(x, learner = "best", plot = FALSE)

## access BLP generic targets for ranger learner w/o plot
get_BLP(x, learner = "mlr3::lrn('ranger', num.trees = 10)", plot = FALSE)

## access GATES generic targets for best learner w/o plot
get_GATES(x, learner = "best", plot = FALSE)

## access GATES generic targets for ranger learner w/o plot
get_GATES(x, learner = "mlr3::lrn('ranger', num.trees = 10)", plot = FALSE)

## access CLAN generic targets for "V1" & best learner, w/o plot
get_CLAN(x, learner = "best", variable = "V1", plot = FALSE)

## access CLAN generic targets for "V1" & ranger learner, w/o plot
get_CLAN(x, learner = "mlr3::lrn('ranger', num.trees = 10)",
         variable = "V1", plot = FALSE)
}
```

`get_BLP` *Accessor function for the BLP generic target estimates*

Description

Accessor function for the BLP generic target estimates

Usage

```
get_BLP(x, learner = "best", plot = TRUE)
```

Arguments

`x`	An object of the class `"GenericML"`, as returned by the function `GenericML()`.
`learner`	A character string of the learner whose BLP generic target estimates shall be accessed. Default is `"best"` for the best learner for BLP.
`plot`	Logical. If `TRUE` (default), a `"ggplot"` object is computed.

Value

An object of class "BLP_info", which consists of the following components:

estimate A numeric vector of point estimates of the BLP generic targets.

confidence_interval A numeric matrix of the lower and upper confidence bounds for each generic target. The confidence level of the implied confidence interval is equal to 1 - 2 * significance_level.

confidence_level The confidence level of the confidence intervals. Equals 1 - 2 * significance_level.

learner The argument learner.

plot An object of class "ggplot". Only returned if the argument plot = TRUE.

See Also

GenericML(), get_GATES(), get_CLAN(), get_best(), print.BLP_info(), print.GATES_info(), print.CLAN_info()

Examples

```
if(require("rpart") && require("ranger")){
## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome

## column names of Z
colnames(Z) <- paste0("V", 1:p)

## specify learners
learners <- c("tree", "mlr3::lrn('ranger', num.trees = 10)")

## perform generic ML inference
# small number of splits to keep computation time low
x <- GenericML(Z, D, Y, learners, num_splits = 2,
               parallel = FALSE)

## access best learner
get_best(x)

## access BLP generic targets for best learner w/o plot
get_BLP(x, learner = "best", plot = FALSE)

## access BLP generic targets for ranger learner w/o plot
get_BLP(x, learner = "mlr3::lrn('ranger', num.trees = 10)", plot = FALSE)

## access GATES generic targets for best learner w/o plot
```

```
get_GATES(x, learner = "best", plot = FALSE)

## access GATES generic targets for ranger learner w/o plot
get_GATES(x, learner = "mlr3::lrn('ranger', num.trees = 10)", plot = FALSE)

## access CLAN generic targets for "V1" & best learner, w/o plot
get_CLAN(x, learner = "best", variable = "V1", plot = FALSE)

## access CLAN generic targets for "V1" & ranger learner, w/o plot
get_CLAN(x, learner = "mlr3::lrn('ranger', num.trees = 10)",
         variable = "V1", plot = FALSE)
}
```

`get_CLAN` *Accessor function for the CLAN generic target estimates*

Description

Accessor function for the CLAN generic target estimates

Usage

```
get_CLAN(x, variable, learner = "best", plot = TRUE)
```

Arguments

`x`	An object of the class `"GenericML"`, as returned by the function `GenericML()`.
`variable`	The (character) name of a variabe on which CLAN was performed.
`learner`	A character string of the learner whose CLAN generic target estimates shall be accessed. Default is `"best"` for the best learner for CLAN
`plot`	Logical. If `TRUE` (default), a `"ggplot"` object is computed.

Value

An object of class `"CLAN_info"`, which consists of the following components:

`estimate` A numeric vector of point estimates of the CLAN generic targets.

`confidence_interval` A numeric matrix of the lower and upper confidence bounds for each generic target. The confidence level of the implied confidence interval is equal to `1 - 2 * significance_level`.

`confidence_level` The confidence level of the confidence intervals. Equals `1 - 2 * significance_level`.

`learner` The argument `learner`.

`plot` An object of class `"ggplot"`. Only returned if the argument `plot = TRUE`.

`CLAN_variable` The name of the CLAN variable of interest.

See Also

GenericML(), get_BLP(), get_GATES(), get_best(), print.BLP_info(), print.GATES_info(), print.CLAN_info()

Examples

```
if(require("rpart") && require("ranger")){
## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome

## column names of Z
colnames(Z) <- paste0("V", 1:p)

## specify learners
learners <- c("tree", "mlr3::lrn('ranger', num.trees = 10)")

## perform generic ML inference
# small number of splits to keep computation time low
x <- GenericML(Z, D, Y, learners, num_splits = 2,
               parallel = FALSE)

## access best learner
get_best(x)

## access BLP generic targets for best learner w/o plot
get_BLP(x, learner = "best", plot = FALSE)

## access BLP generic targets for ranger learner w/o plot
get_BLP(x, learner = "mlr3::lrn('ranger', num.trees = 10)", plot = FALSE)

## access GATES generic targets for best learner w/o plot
get_GATES(x, learner = "best", plot = FALSE)

## access GATES generic targets for ranger learner w/o plot
get_GATES(x, learner = "mlr3::lrn('ranger', num.trees = 10)", plot = FALSE)

## access CLAN generic targets for "V1" & best learner, w/o plot
get_CLAN(x, learner = "best", variable = "V1", plot = FALSE)

## access CLAN generic targets for "V1" & ranger learner, w/o plot
get_CLAN(x, learner = "mlr3::lrn('ranger', num.trees = 10)",
         variable = "V1", plot = FALSE)
}
```

get_GATES *Accessor function for the GATES generic target estimates*

Description

Accessor function for the GATES generic target estimates

Usage

```
get_GATES(x, learner = "best", plot = TRUE)
```

Arguments

x	An object of the class "GenericML", as returned by the function GenericML().
learner	A character string of the learner whose GATES generic target estimates shall be accessed. Default is "best" for the best learner for GATES.
plot	Logical. If TRUE (default), a "ggplot" object is computed.

Value

An object of class "GATES_info", which consists of the following components:

estimate A numeric vector of point estimates of the GATES generic targets.

confidence_interval A numeric matrix of the lower and upper confidence bounds for each generic target. The confidence level of the implied confidence interval is equal to 1 - 2 * significance_level.

confidence_level The confidence level of the confidence intervals. Equals 1 - 2 * significance_level.

learner The argument learner.

plot An object of class "ggplot". Only returned if the argument plot = TRUE.

See Also

GenericML(), get_BLP(), get_CLAN(), get_best(), print.BLP_info(), print.GATES_info(), print.CLAN_info()

Examples

```
if(require("rpart") && require("ranger")){
## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome
```

```
## column names of Z
colnames(Z) <- paste0("V", 1:p)

## specify learners
learners <- c("tree", "mlr3::lrn('ranger', num.trees = 10)")

## perform generic ML inference
# small number of splits to keep computation time low
x <- GenericML(Z, D, Y, learners, num_splits = 2,
               parallel = FALSE)

## access best learner
get_best(x)

## access BLP generic targets for best learner w/o plot
get_BLP(x, learner = "best", plot = FALSE)

## access BLP generic targets for ranger learner w/o plot
get_BLP(x, learner = "mlr3::lrn('ranger', num.trees = 10)", plot = FALSE)

## access GATES generic targets for best learner w/o plot
get_GATES(x, learner = "best", plot = FALSE)

## access GATES generic targets for ranger learner w/o plot
get_GATES(x, learner = "mlr3::lrn('ranger', num.trees = 10)", plot = FALSE)

## access CLAN generic targets for "V1" & best learner, w/o plot
get_CLAN(x, learner = "best", variable = "V1", plot = FALSE)

## access CLAN generic targets for "V1" & ranger learner, w/o plot
get_CLAN(x, learner = "mlr3::lrn('ranger', num.trees = 10)",
         variable = "V1", plot = FALSE)
}
```

`heterogeneity_CLAN` *Evaluate treatment effect heterogeneity along CLAN variables*

Description

This function tests for statistical significance of all CLAN difference parameters that were specified in the function `setup_diff()`. It reports all CLAN variables along which there are significant difference parameters, which corresponds to evidence for treatment effect heterogeneity along this variable, at the specified significance level.

Usage

```
heterogeneity_CLAN(x, learner = "best", significance_level = 0.05)
```

Arguments

x	An object of class "GenericML", as returned by the function GenericML().
learner	A character string of the learner whose CLAN generic target estimates are of interest. Default is "best" for the best learner for CLAN.
significance_level	Level for the significance tests. Default is 0.05.

Value

An object of class "heterogeneity_CLAN", consisting of the following components:

p_values A matrix of p values of all CLAN difference parameters for all CLAN variables.

significant The names of variables with at least one significant CLAN difference parameter ("variables"), their number "num_variables", and the total number of significant CLAN difference parameters "num_params". All significance tests were performed at level significance_level.

min_pval Information on the smallest p value: Its value ("value"), the variable in which it was estimated ("variable"), the CLAN difference parameter it belongs to ("parameter"), and whether or not it is significant at level significance_level ("significant").

"learner" Name of the learner whose median estimates we used for the listed results.

"significance_level" The level of the significance tests.

See Also

GenericML()

---

lambda_parameters *Estimate the two lambda parameters*

Description

Estimates the lambda parameters Λ and $\bar{\Lambda}$ whose medians are used to find the best ML method.

Usage

```
lambda_parameters(BLP, GATES, proxy_CATE, membership)
```

Arguments

BLP	An object of class "BLP".
GATES	An object of class "GATES".
proxy_CATE	Proxy estimates of the CATE.
membership	A logical matrix that indicates the group membership of each observation in Z_CLAN. Needs to be of type "quantile_group". Typically, the grouping is based on CATE estimates, which are for instance returned by proxy_CATE().

Value

A list containing the estimates of Λ and $\bar{\Lambda}$, denoted `lambda` and `lambda.bar`, respectively.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

Examples

```
## generate data
set.seed(1)
n  <- 200                               # number of observations
p  <- 5                                 # number of covariates
D  <- rbinom(n, 1, 0.5)                 # random treatment assignment
Y  <- runif(n)                          # outcome variable
propensity_scores <- rep(0.5, n)        # propensity scores
proxy_BCA         <- runif(n)           # proxy BCA estimates
proxy_CATE        <- runif(n)           # proxy CATE estimates
membership <- quantile_group(proxy_CATE) # group membership

## perform BLP
BLP <- BLP(Y, D, propensity_scores, proxy_BCA, proxy_CATE)

## perform GATES
GATES <- GATES(Y, D, propensity_scores, proxy_BCA, proxy_CATE, membership)

## get estimates of the lambda parameters
lambda_parameters(BLP, GATES, proxy_CATE, membership)
```

Med *Calculate lower and upper median*

Description

Calculates the lower and and median of a vector as proposed in Comment 4.2 in the paper.

Usage

```
Med(x)
```

Arguments

x	A numeric vector.

Value

A list with the upper and lower median and the Med statistic (which is their mean).

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

Examples

```
set.seed(1)
x <- runif(100)
Med(x)
```

`plot.GenericML` *Plot method for a* `"GenericML"` *object*

Description

Visualizes the estimates of the generic targets of interest: plots the point estimates as well as the corresponding confidence intervals. The generic targets of interest can be (subsets of) the parameters of the BLP, GATES, or CLAN analysis.

Usage

```
## S3 method for class 'GenericML'
plot(
  x,
  type = "GATES",
  learner = "best",
  CLAN_variable = NULL,
  groups = "all",
  ATE = TRUE,
  limits = NULL,
  title = NULL,
  ...
)
```

Arguments

`x`	An object of the class `"GenericML"`, as returned by the function `GenericML()`.
`type`	The analysis whose parameters shall be plotted. Either `"GATES"`, `"BLP"`, or `"CLAN"`. Default is `"GATES"`.
`learner`	The learner whose results are to be returned. Default is `"best"` for the best learner as measured by the Λ parameters.
`CLAN_variable`	Name of the CLAN variable to be plotted. Only applicable if `type = "CLAN"`.

groups	Character vector indicating the per-group parameter estimates that shall be plotted in GATES and CLAN analyses. Default is "all" for all parameters. If there are K groups, this variable is a subset of c("G1", "G2", ..., "GK", "G1-G2", "G1-G2", ..., "G1-GK", "GK-G1", "GK-G2", ...), where Gk denotes the k-th group. Note that this set depends on the choices of the arguments "diff_GATES" and "diff_CLAN" of the "GenericML" object.
ATE	Logical. If TRUE (default), then the BLP estimate of the average treatment effect along with confidence intervals will be added to the plot. Only applicable if type is "CLAN" or "GATES".
limits	A numeric vector of length two holding the limits of the y-axis of the plot.
title	The title of the plot.
...	Additional arguments to be passed down.

Details

If you wish to retrieve the data frame that this plot method visualizes, please use setup_plot().

Value

An object of class "ggplot".

See Also

setup_plot(), GenericML(), get_BLP(), get_GATES(), get_CLAN(), setup_diff()

Examples

```
if(require("ranger")) {

## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome

## name the columns of Z
colnames(Z) <- paste0("V", 1:p)

## specify learners
learners <- c("random_forest")

## specify quantile cutoffs (the 4 quartile groups here)
quantile_cutoffs <- c(0.25, 0.5, 0.75)

## specify the differenced generic targets of GATES and CLAN
diff_GATES <- setup_diff(subtract_from = "most",
```

```
                          subtracted = c(1,2,3))
diff_CLAN  <- setup_diff(subtract_from = "least",
                          subtracted = c(3,2))

## perform generic ML inference
# small number of splits to keep computation time low
x <- GenericML(Z, D, Y, learners, num_splits = 2,
               quantile_cutoffs = quantile_cutoffs,
               diff_GATES = diff_GATES,
               diff_CLAN = diff_CLAN,
               parallel = FALSE)

## plot BLP parameters
plot(x, type = "BLP")

## plot GATES parameters "G1", "G4", "G4-G1"
plot(x, type = "GATES", groups = c("G1", "G4", "G4-G1"))

## plot CLAN parameters "G1", "G2", "G2-G1" of variable "V1":
plot(x, type = "CLAN", CLAN_variable = "V1",
     groups = c("G1", "G2", "G1-G3"))
}
```

`print.BLP_info` *Print method for a* `"BLP_info"` *object*

Description

Print method for a `"BLP_info"` object

Usage

```
## S3 method for class 'BLP_info'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`x`	An object of the class `"BLP_info"`, as returned by the function `get_BLP()`.
`digits`	Number of digits to print.
`...`	Additional arguments to be passed down.

Value

A print to the console.

print.CLAN_info *Print method for a "CLAN_info" object*

Description

Print method for a "CLAN_info" object

Usage

```
## S3 method for class 'CLAN_info'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of the class "CLAN_info", as returned by the function get_CLAN().
digits	Number of digits to print.
...	Additional arguments to be passed down.

Value

A print to the console.

print.GATES_info *Print method for a "GATES_info" object*

Description

Print method for a "GATES_info" object

Usage

```
## S3 method for class 'GATES_info'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of the class "GATES_info", as returned by the function get_GATES().
digits	Number of digits to print.
...	Additional arguments to be passed down.

Value

A print to the console.

print.GenericML *Print method for a* GenericML *object*

Description

Prints key results of the analyses conducted in GenericML().

Usage

```
## S3 method for class 'GenericML'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of the class "GenericML", as returned by the function GenericML().
digits	Number of digits to print.
...	Additional arguments to be passed down.

Value

A print to the console.

Examples

```
if(require("ranger")){

## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome

## specify learners
learners <- c("random_forest")

## perform generic ML inference
# small number of splits to keep computation time low
x <- GenericML(Z, D, Y, learners, num_splits = 2,
               parallel = FALSE)

## print
print(x)
}
```

print.heterogeneity_CLAN *Print method for a "heterogeneity_CLAN" object*

Description

Print method for a "heterogeneity_CLAN" object

Usage

```
## S3 method for class 'heterogeneity_CLAN'
print(x, ...)
```

Arguments

x	An object of class "heterogeneity_CLAN".
...	Additional arguments to be passed down.

Value

A print to the console.

propensity_score *Propensity score estimation*

Description

Estimates the propensity scores $Pr[D = 1|Z]$ for binary treatment assignment D and covariates Z. Either done by taking the empirical mean of D (which should equal roughly 0.5, since we assume a randomized experiment), or by direct machine learning estimation.

Usage

```
propensity_score(Z, D, estimator = "constant")
```

Arguments

Z	A numeric design matrix that holds the covariates in its columns.
D	A binary vector of treatment assignment. Value one denotes assignment to the treatment group and value zero assignment to the control group.
estimator	Character specifying the estimator. Must either be equal to 'constant' (estimates the propensity scores by mean(D)), 'lasso', 'random_forest', 'tree', or mlr3 syntax. Note that in case of mlr3 syntax, do *not* specify if the learner is a regression learner or classification learner. Example: 'mlr3::lrn("ranger", num.trees = 500)' for a random forest learner. Note that this is a string and the absence of the classif. or regr. keywords. See https://mlr3learners.mlr-org.com for a list of mlr3 learners.

Details

The specifications "lasso", "random_forest", and "tree" in estimator correspond to the following mlr3 specifications (we omit the keywords classif. and regr.). "lasso" is a cross-validated Lasso estimator, which corresponds to 'mlr3::lrn("cv_glmnet", s = "lambda.min", alpha = 1)'. "random_forest" is a random forest with 500 trees, which corresponds to 'mlr3::lrn("ranger", num.trees = 500)'. "tree" is a tree learner, which corresponds to 'mlr3::lrn("rpart")'.

Value

An object of class "propensity_score", consisting of the following components:

estimates A numeric vector of propensity score estimates.

mlr3_objects "mlr3" objects used for estimation. Only non-empty if mlr3 was used.

References

Rosenbaum P.R., Rubin D.B. (1983). "The Central Role of the Propensity Score in Observational Studies for Causal Effects." *Biometrika*, **70**(1), 41–55. doi: 10.1093/biomet/70.1.41.

Lang M., Binder M., Richter J., Schratz P., Pfisterer F., Coors S., Au Q., Casalicchio G., Kotthoff L., Bischl B. (2019). "mlr3: A Modern Object-Oriented Machine Learning Framework in R." *Journal of Open Source Software*, **4**(44), 1903. doi: 10.21105/joss.01903.

Examples

```
## generate data
set.seed(1)
n  <- 100                       # number of observations
p  <- 5                         # number of covariates
D  <- rbinom(n, 1, 0.5)         # random treatment assignment
Z  <- matrix(runif(n*p), n, p)  # design matrix

## estimate propensity scores via mean(D)...
propensity_score(Z, D, estimator = "constant")

## ... and via SVM with cache size 40
if(require("e1071")){
  propensity_score(Z, D,
   estimator = 'mlr3::lrn("svm", cachesize = 40)')
}
```

---

proxy_BCA *Baseline Conditional Average*

Description

Proxy estimation of the Baseline Conditional Average (BCA), defined by $E[Y|D = 0, Z]$. Estimation is done on the auxiliary sample, but BCA predictions are made for all observations.

Usage

```
proxy_BCA(Z, D, Y, A_set, learner, min_variation = 1e-05)
```

Arguments

Z	A numeric design matrix that holds the covariates in its columns.
D	A binary vector of treatment assignment. Value one denotes assignment to the treatment group and value zero assignment to the control group.
Y	A numeric vector containing the response variable.
A_set	A numerical vector of the indices of the observations in the auxiliary sample.
learner	A string specifying the machine learner for the estimation. Either 'lasso', 'random_forest', 'tree', or a custom learner specified with mlr3 syntax. In the latter case, do *not* specify in the mlr3 syntax specification if the learner is a regression learner or classification learner. Example: 'mlr3::lrn("ranger", num.trees = 100)' for a random forest learner with 100 trees. Note that this is a string and the absence of the classif. or regr. keywords. See https://mlr3learners.mlr-org.com for a list of mlr3 learners.
min_variation	Specifies a threshold for the minimum variation of the predictions. If the variation of a BCA prediction falls below this threshold, random noise with distribution $N(0, var(Y)/20)$ is added to it. Default is 1e-05.

Details

The specifications "lasso", "random_forest", and "tree" in learner correspond to the following mlr3 specifications (we omit the keywords classif. and regr.). "lasso" is a cross-validated Lasso estimator, which corresponds to 'mlr3::lrn("cv_glmnet", s = "lambda.min", alpha = 1)'. "random_forest" is a random forest with 500 trees, which corresponds to 'mlr3::lrn("ranger", num.trees = 500)'. "tree" is a tree learner, which corresponds to 'mlr3::lrn("rpart")'.

Value

An object of class "proxy_BCA", consisting of the following components:

estimates A numeric vector of BCA estimates of each observation.

mlr3_objects "mlr3" objects used for estimation.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

Lang M., Binder M., Richter J., Schratz P., Pfisterer F., Coors S., Au Q., Casalicchio G., Kotthoff L., Bischl B. (2019). "mlr3: A Modern Object-Oriented Machine Learning Framework in R." *Journal of Open Source Software*, **4**(44), 1903. doi: 10.21105/joss.01903.

See Also

proxy_CATE()

Examples

```
if(require("ranger")){
## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome
A_set <- sample(1:n, size = n/2)           # auxiliary set

## BCA predictions via random forest
proxy_BCA(Z, D, Y, A_set, learner = "mlr3::lrn('ranger', num.trees = 10)")
}
```

`proxy_CATE` *Conditional Average Treatment Effect*

Description

Proxy estimation of the Conditional Average Treatment Effect (CATE), defined by $E[Y|D = 1, Z] - E[Y|D = 0, Z]$. Estimation is done on the auxiliary sample, but CATE predictions are made for all observations.

Usage

```
proxy_CATE(Z, D, Y, A_set, learner, proxy_BCA = NULL, min_variation = 1e-05)
```

Arguments

`Z`	A numeric design matrix that holds the covariates in its columns.
`D`	A binary vector of treatment assignment. Value one denotes assignment to the treatment group and value zero assignment to the control group.
`Y`	A numeric vector containing the response variable.
`A_set`	A numerical vector of the indices of the observations in the auxiliary sample.
`learner`	A string specifying the machine learner for the estimation. Either `'lasso'`, `'random_forest'`, `'tree'`, or a custom learner specified with `mlr3` syntax. In the latter case, do *not* specify in the `mlr3` syntax specification if the learner is a regression learner or classification learner. Example: `'mlr3::lrn("ranger", num.trees = 100)'` for a random forest learner with 100 trees. Note that this is a string and the absence of the `classif.` or `regr.` keywords. See https://mlr3learners.mlr-org.com for a list of `mlr3` learners.
`proxy_BCA`	A vector of proxy estimates of the baseline conditional average, BCA, $E[Y|D = 0, Z]$. If `NULL`, these will be estimated separately.

`min_variation` Minimum variation of the predictions before random noise with distribution $N(0, var(Y)/20)$ is added. Default is `1e-05`.

Details

The specifications `"lasso"`, `"random_forest"`, and `"tree"` in `learner` correspond to the following `mlr3` specifications (we omit the keywords `classif.` and `regr.`). `"lasso"` is a cross-validated Lasso estimator, which corresponds to `'mlr3::lrn("cv_glmnet", s = "lambda.min", alpha = 1)'`. `"random_forest"` is a random forest with 500 trees, which corresponds to `'mlr3::lrn("ranger", num.trees = 500)'`. `"tree"` is a tree learner, which corresponds to `'mlr3::lrn("rpart")'`.

Value

An object of class `"proxy_CATE"`, consisting of the following components:

`estimates` A numeric vector of CATE estimates of each observation.

`mlr3_objects` `"mlr3"` objects used for estimation of $E[Y|D = 1, Z]$ (`Y1_learner`) and $E[Y|D = 0, Z]$ (`Y0_learner`). The latter is not available if `proxy_BCA = NULL`.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

Lang M., Binder M., Richter J., Schratz P., Pfisterer F., Coors S., Au Q., Casalicchio G., Kotthoff L., Bischl B. (2019). "mlr3: A Modern Object-Oriented Machine Learning Framework in R." *Journal of Open Source Software*, **4**(44), 1903. doi: 10.21105/joss.01903.

See Also

`proxy_BCA()`

Examples

```
if(require("ranger")){
## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome
A_set <- sample(1:n, size = n/2)           # auxiliary set

## CATE predictions via random forest
proxy_CATE(Z, D, Y, A_set, learner = "mlr3::lrn('ranger', num.trees = 10)")
}
```

quantile_group *Partition a vector into quantile groups*

Description

Partitions a vector into quantile groups and returns a logical matrix indicating group membership.

Usage

```
quantile_group(x, cutoffs = c(0.25, 0.5, 0.75))
```

Arguments

x	A numeric vector to be partitioned.
cutoffs	A numeric vector denoting the quantile cutoffs for the partition. Default are the quartiles: `c(0.25, 0.5, 0.75)`.

Value

An object of type `"quantile_group"`, which is a logical matrix indicating group membership.

Examples

```
set.seed(1)
x <- runif(100)
cutoffs <- c(0.25, 0.5, 0.75)
quantile_group(x, cutoffs)
```

setup_diff *Setup function for* `diff` *arguments*

Description

This setup function controls how differences of generic target parameters are taken. Returns a list with two components, called `subtract_from` and `subtracted`. The first element (`subtract_from`) denotes what shall be the base group to subtract from in the generic targets of interest (GATES or CLAN); either `"most"` or `"least"`. The second element (`subtracted`) are the groups to be subtracted from `subtract_from`, which is a subset of $1, 2, ..., K$, where K equals the number of groups. The number of groups should be consistent with the number of groups induced by the argument `quantile_cutoffs`, which is the cardinality of `quantile_cutoffs`, plus one.

Usage

```
setup_diff(subtract_from = "most", subtracted = 1)
```

Arguments

subtract_from	String indicating the base group to subtract from, either "most" (default) or "least". The most affected group corresponds to the K-th group in the paper (there are K groups). The least affected group corresponds to the first group.
subtracted	Vector indicating the groups to be subtracted from the group specified in subtract_from. If there are K groups, subtracted should be a subset of $1, 2, ..., K$. Be careful to not specify a zero difference: If subtract_from = "most", subtracting group K results in a zero difference. Same if subtract_from = "least" and we subtract group 1.

Details

The output of this setup function is intended to be used as argument in the functions GenericML() and GenericML_single() (arguments diff_GATES, diff_CLAN), as well as GATES() and CLAN() (argument diff).

Value

An object of class "setup_diff", consisting of the following components:

subtract_from A character equal to "most" or "least".

subtracted A numeric vector of group indices.

See the description above for details.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

See Also

GenericML(), GenericML_single(), CLAN(), GATES(), setup_X1(), setup_vcov()

Examples

```
## specify quantile cutoffs (the 4 quartile groups here)
quantile_cutoffs <- c(0.25, 0.5, 0.75)

## Use group difference GK-G1 as generic targets in GATES and CLAN
## Gx is the x-th group
setup_diff(subtract_from = "most", subtracted = 1)

## Use GK-G1, GK-G2, GK-G3 as differenced generic targets
setup_diff(subtract_from = "most", subtracted = c(1,2,3))

## Use G1-G2, G1-G3 as differenced generic targets
setup_diff(subtract_from = "least", subtracted = c(3,2))
```

setup_plot *Set up information for a* GenericML() *plot*

Description

Extract the relevant information for visualizing the point and interval estimates of the generic targets of interest. The generic targets of interest can be (subsets of) the parameters of the BLP, GATES, or CLAN analysis.

Usage

```
setup_plot(
  x,
  type = "GATES",
  learner = "best",
  CLAN_variable = NULL,
  groups = "all"
)
```

Arguments

x	An object of the class "GenericML", as returned by the function GenericML().
type	The analysis whose parameters shall be plotted. Either "GATES", "BLP", or "CLAN". Default is "GATES".
learner	The learner whose results are to be returned. Default is "best" for the best learner as measured by the Λ parameters.
CLAN_variable	Name of the CLAN variable to be plotted. Only applicable if type = "CLAN".
groups	Character vector indicating the per-group parameter estimates that shall be plotted in GATES and CLAN analyses. Default is "all" for all parameters. If there are K groups, this variable is a subset of c("G1", "G2",...,"GK", "G1-G2", "G1-G2",..., "G1-GK", "GK-G1", "GK-G2",...), where Gk denotes the k-th group. Note that this set depends on the choices of the arguments "diff_GATES" and "diff_CLAN" of the "GenericML" object.

Details

This function is used internally by plot.GenericML(). It may also be useful for users who want to produce a similar plot, but who want more control over what information to display or how to display that information.

Value

An object of class "setup_plot", which is a list with the following elements.

data_plot A data frame containing point and interval estimates of the generic target specified in the argument type.

`data_BLP` A data frame containing point and interval estimates of the BLP analysis.

`confidence_level` The confidence level of the confidence intervals. The confidence level is equal to `1 - 2 * significance_level`, which is the adjustment proposed in the paper.

See Also

`plot.GenericML()`

Examples

```
if(require("ranger") && require("ggplot2")) {

## generate data
set.seed(1)
n  <- 150                                  # number of observations
p  <- 5                                    # number of covariates
D  <- rbinom(n, 1, 0.5)                    # random treatment assignment
Z  <- matrix(runif(n*p), n, p)             # design matrix
Y0 <- as.numeric(Z %*% rexp(p) + rnorm(n)) # potential outcome without treatment
Y1 <- 2 + Y0                               # potential outcome under treatment
Y  <- ifelse(D == 1, Y1, Y0)               # observed outcome

## name the columns of Z
colnames(Z) <- paste0("V", 1:p)

## specify learners
learners <- c("random_forest")

## perform generic ML inference
# small number of splits to keep computation time low
x <- GenericML(Z, D, Y, learners,
               num_splits = 2,
               parallel = FALSE)

## the plot we wish to replicate
plot(x = x, type = "GATES")

## get the data to plot the GATES estimates
data <- setup_plot(x = x, type = "GATES")

## define variables to appease the R CMD check
group <- estimate <- ci_lower <- ci_upper <- NULL

## replicate the plot(x, type = "GATES")
# for simplicity, we skip aligning the colors
ggplot(mapping = aes(x = group,
                     y = estimate), data = data$data_plot) +
  geom_hline(aes(yintercept = 0),
             color = "black", linetype = "dotted") +
  geom_hline(aes(yintercept = data$data_BLP["beta.1", "estimate"],
                 color = "ATE"),
             linetype = "dashed") +
```

```
  geom_hline(aes(yintercept = data$data_BLP["beta.1", "ci_lower"],
                 color = paste0(100*data$confidence_level, "% CI (ATE)")),
             linetype = "dashed")  +
  geom_hline(yintercept = data$data_BLP["beta.1", "ci_upper"],
             linetype = "dashed", color = "red") +
 geom_point(aes(color = paste0("GATES with ",  100*data$confidence_level, "% CI")), size = 3) +
  geom_errorbar(mapping = aes(ymin = ci_lower,
                              ymax = ci_upper))
}
```

`setup_stratify` *Setup function for stratified sampling*

Description

This function controls whether or not stratified sample splitting shall be performed. If no stratified sampling shall be performed, do not pass any arguments to this function (this is the default). If stratified sampling shall be performed, use this function to pass arguments to `stratified()` in the package "splitstackshape". In this case, the specification for `prop_aux` in `GenericML()` does not have an effect because the number of samples in the auxiliary set is specified with the `size` argument in `stratified()`.

Usage

```
setup_stratify(...)
```

Arguments

`...`	Named objects that shall be used as arguments in `stratified()`. If empty (default), ordinary random sampling will be performed.

Details

The output of this setup function is intended to be used as argument `stratify` in the function `GenericML()`. If arguments are passed to `stratified()` via this function, make sure to pass the necessary objects that `stratified()` in the "splitstackshape" package requires. The necessary objects are called `indt`, `group`, and `size` (see the documentation of `stratified()` for details). If either of these objects is missing, an error is thrown.

Value

A list of named objects (possibly empty) specifying the stratified sampling strategy. If empty, no stratified sampling will be performed and instead ordinary random sampling will be performed.

See Also

`stratified()`, `GenericML()`

Examples

```
## sample data of group membership (with two groups)
set.seed(1)
n <- 500
groups <- data.frame(group1 = rbinom(n, 1, 0.2),
                     group2 = rbinom(n, 1, 0.3))

## suppose we want both groups to be present in a strata...
group <- c("group1", "group2")

## ... and that the size of the strata equals half of the observations per group
size <- 0.5

## obtain a list of arguments that will be passed to splitstackshape::stratified()
setup_stratify(indt = groups, group = group, size = size)

## if no stratified sampling shall be used, do not pass anything
setup_stratify()
```

---

setup_vcov *Setup function for* vcov_control *arguments*

Description

Returns a list with two elements called `estimator` and `arguments`. The element `estimator` is a string specifying the covariance matrix estimator to be used in the linear regression regression of interest and needs to be a covariance estimator function in the "sandwich" package. The second element, `arguments`, is a list of arguments that shall be passed to the function specified in the first element, `estimator`.

Usage

```
setup_vcov(estimator = "vcovHC", arguments = list(type = "const"))
```

Arguments

`estimator`	Character specifying a covariance matrix estimator in the "sandwich" package. Default is `"vcovHC"`. Supported estimators are `"vcovBS"`, `"vcovCL"`, `"vcovHAC"`, and `"vcovHC"`.
`arguments`	A list of arguments that are to be passed to the function in the `"sandwich"` package that is specified in `estimator`. Default is `list(type = "const")`, which specifies the homoskedastic ordinary least squares covariance matrix estimator.

Details

The output of this setup function is intended to be used as argument in the functions `GenericML()` and `GenericML_single()` (arguments `vcov_BLP`, `vcov_GATES`), as well as `BLP()` and `GATES()` (argument `vcov_control`).

Value

An object of class "setup_vcov", consisting of the following components:

estimator A character equal to covariance estimation function names in the "sandwich" package.

arguments A list of arguments that shall be passed to the function specified in the estimator argument.

See the description above for details.

References

Zeileis A. (2004). "Econometric Computing with HC and HAC Covariance Matrix Estimators." *Journal of Statistical Software*, **11**(10), 1–17. doi: 10.18637/jss.v011.i10

Zeileis A. (2006). "Object-Oriented Computation of Sandwich Estimators." *Journal of Statistical Software*, **16**(9), 1–16. doi: 10.18637/jss.v016.i09

See Also

GenericML(), GenericML_single(), BLP(), GATES(), setup_X1(), setup_diff()

Examples

```
# use standard homoskedastic OLS covariance matrix estimate
setup_vcov(estimator = "vcovHC", arguments = list(type = "const"))

# use White's heteroskedasticity-robust estimator
setup_vcov(estimator = "vcovHC", arguments = list(type = "HC0"))

if (require("sandwich")){

# use HAC-robust estimator with prewhitening and Andrews' (Econometrica, 1991) weights
# since weightsAndrews() is a function in 'sandwich', require this package
setup_vcov(estimator = "vcovHAC", arguments = list(prewhite = TRUE, weights = weightsAndrews))

}
```

---

setup_X1 *Setup function controlling the matrix X_1 in the BLP or GATES regression*

Description

Returns a list with three elements. The first element of the list, funs_Z, controls which functions of matrix Z are used as regressors in X_1. The second element, covariates, is an optional matrix of custom covariates that shall be included in X_1. The third element, fixed_effects, controls the inclusion of fixed effects.

Usage

```
setup_X1(funs_Z = c("B"), covariates = NULL, fixed_effects = NULL)
```

Arguments

`funs_Z`	Character vector controlling the functions of `Z` to be included in X_1. Subset of `c("S", "B", "p")`, where `"p"` corresponds to the propensity scores, `"B"` to the proxy baseline estimates, and `"S"` to the proxy CATE estimates. Default is `"B"`.
`covariates`	Optional numeric matrix containing additional covariates to be included in X_1. Default is `NULL`.
`fixed_effects`	Numeric vector of integers that indicates cluster membership of the observations: For each cluster, a fixed effect will be added. Default is `NULL` for no fixed effects.

Details

The output of this setup function is intended to be used as argument in the functions `GenericML()` and `GenericML_single()` (arguments `X1_BLP`, `X1_GATES`), as well as `BLP()` and `GATES()` (argument `X1_control`).

Value

An object of class `"setup_X1"`, consisting of the following components:

`funs_Z` A character vector, being a subset of `c("S", "B", "p")`.

`covariates` Either `NULL` or a numeric matrix.

`fixed_effects` Either `NULL` or an integer vector indicating cluster membership.

See the description above for details.

References

Chernozhukov V., Demirer M., Duflo E., Fernández-Val I. (2020). "Generic Machine Learning Inference on Heterogenous Treatment Effects in Randomized Experiments." *arXiv preprint arXiv:1712.04802*. URL: https://arxiv.org/abs/1712.04802.

See Also

`GenericML()`, `GenericML_single()`, `BLP()`, `GATES()`, `setup_vcov()`, `setup_diff()`

Examples

```
set.seed(1)
n <- 100 # sample size
p <- 5   # number of covariates
covariates <- matrix(runif(n*p), n, p) # sample matrix of covariates

# let there be three clusters; assign membership randomly
fixed_effects <- sample(c(1,2,3), size = n, replace = TRUE)
```

```
# use BCA estimates in matrix X1
setup_X1(funs_Z = "B", covariates = NULL, fixed_effects = NULL)

# use BCA and propensity score estimates in matrix X1
# uses uniform covariates and fixed effects
setup_X1(funs_Z = c("B", "p"), covariates = covariates, fixed_effects = NULL)
```

`TrueIfUnix` *Check if user's OS is a Unix system*

Description

Check if user's OS is a Unix system

Usage

```
TrueIfUnix()
```

Value

A Boolean that is `TRUE` if the user's operating system is a Unix system and `FALSE` otherwise.

Index