Package ‘postDoubleR’

October 7, 2019

Title Post Double Selection with Double Machine Learning

Version 1.4.12

BugReports https://github.com/JSzitas/postDoubleR/issues

Description Implements post double selection using double machine learning,
see Chernozhukov, Chetverikov, Demirer, Duflo,
Hansen, Newey, Robins (2017) <doi:10.1111/ectj.12097>,
for various models, using several back ends. Allows the user flexibility
in specifying their own methods for estimation.

License GPL-3

Suggests testthat (>= 2.1.0), knitr, rmarkdown

Encoding UTF-8

LazyData true

URL https://github.com/JSzitas/postDoubleR

RoxygenNote 6.1.1

Imports glmnet, grf, neuralnet, progress, ggplot2, stats, parallel,
doParallel

NeedsCompilation no

Author Juraj Szitás [aut, cre]

Maintainer Juraj Szitás <szitas.juraj13@gmail.com>

Repository CRAN

Date/Publication 2019-10-07 10:40:03 UTC

R topics documented:

`cross_fit_helper` *Helper function to for cross-fitting*

Description

Fits estimated models to out of fold samples

Usage

```
cross_fit_helper(model_W, model_Y, folds_to_fit, use)
```

Arguments

`model_W`	The model used to produce the estimates of W_hat
`model_Y`	The model used to produce the estimates of Y_hat
`folds_to_fit`	The split folds for which the calculations are being run.
`use`	The method to use when calculating the out of fold prediction (propagates from method).

Value

A list with four elements: The mean estimate of θ, the standard error of the mean estimate, the associated moment conditions, and the estimated heterogenous effects for the single batch of a single run of the simulation.

Examples

```
n = 1000
p = 10
X = matrix(rnorm(n*p),n,p)
W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
Y = pmax(X[,1], 0) * W + X[,2] + pmin(X[,3], 0) + rnorm(n)

fit_on <- sample(1:1000, size = 333)
pred_on_1 <- sample(c(1:1000)[-fit_on], size = 333)
pred_on_2 <- c(1:1000)[-c(fit_on,pred_on_1)]
models <- ols_helper( X = X[fit_on,],
                      Y = Y[fit_on],
                      W = W[fit_on] )
```

```
folds_fit <- list()
folds_fit[[1]] <- data.frame(cbind(pred_on_1, X[pred_on_1,], W[pred_on_1], Y[pred_on_1]))
folds_fit[[2]] <- data.frame(cbind(pred_on_2, X[pred_on_2,], W[pred_on_2], Y[pred_on_2]))

for(i in 1:length(folds_fit)){
 names( folds_fit[[i]] ) <- c("sample_id","Y_t", paste("X_t_", 1:ncol(X), sep = ""), "W_t")
  }

cross_fit_helper( model_W = models[[1]],
                  model_Y = models[[2]],
                  folds_to_fit = folds_fit,
                  use = "ols")
```

`custom_helper` *Custom generator*

Description

Generates custom methods for use in post-double selection based on user specified inputs

Usage

```
custom_helper(X, Y, W, Y.hat.model, W.hat.model)
```

Arguments

`X`	A matrix of covariates
`Y`	A vector of the target variable, of same length as the number of rows of Y
`W`	A vector of the treatment variable, of same length as the number of rows of X
`Y.hat.model`	A function generates a model to predict from. (with a predict method)
`W.hat.model`	A function generates a model to predict from. (with a predict method)

Value

A list with two elements: The fitted W model and the fitted Y model.

Examples

```
 n = 2000; p = 10
 X = matrix(rnorm(n*p), n, p)
 W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
 Y = rbinom(n, 1,( W + 0.1 * (X[,3] > 0) ))
 Y[is.na(Y)] <- 1
```

```
Y_est <- function(X,Y){
Y_model <- glm(Y~.,family = binomial,data = as.data.frame(cbind(X,Y)))
}

W_est <- function(X,W){
W_model <- glm(W~.,family = binomial,data = as.data.frame(cbind(X,W)))
}

 custom_helper(X = X, Y = Y, W = W,
                  Y.hat.model = substitute(Y_est),
                  W.hat.model = substitute(W_est))

 custom_helper( X = X, Y = Y, W = W,
                Y.hat.model =
                             expression(
                           glm(Y~.,family = "binomial",data = as.data.frame(cbind(X,Y)))
                             ),
                W.hat.model =
                             expression(
                           glm(W~.,family = "gaussian",data = as.data.frame(cbind(X,Y)))
                             )
              )
```

`double_ML` *Post double selection.*

Description

Provides a convenient function to calculate the double ML estimated debiased treatment effect θ.

Usage

```
double_ML(X, Y, W, method = c("glmnet", "randomforest", "nn", "ols",
  "custom"), show.progress = TRUE, specify.own = NULL, k.fld = 2,
  simulations = 100, validate.inputs = TRUE, seed.use = 1071, ...)
```

Arguments

`X`	A matrix of covariates (must be all numeric)
`Y`	A vector of the target variable, of same length as the number of rows of Y, must be numeric
`W`	A vector of the treatment variable, of same length as the number of rows of X, must be numeric
`method`	A selection of methods to use when doing post double selection.
`show.progress`	Whether to display the simulation progress, defaults to TRUE.

`specify.own`	Allows the user to supply the method to calculate $\hat{W}$ and $\hat{Y}$, please refer to custom_helper
`k.fld`	How many fold crossfitting to use, defaults to 2.
`simulations`	How many simulations to use for the final result.
`validate.inputs`	A safety measure indicating whether the types of inputs should be checked, defaults to TRUE (disabled for custom methods).
`seed.use`	The seed to use for simulations, defaults to 1071.
`...`	Other arguments to be passed on, see rf_helper, glmnet_helper and ols_helper for details.

Details

Custom functions are currently implemented through a function called custom_generator. For these custom functions, refer to that function and usage examples.

Value

An object of class "ML_Treatment_Effects" that can be further manipulated (ie there is a plot method implemented).

Examples

```
  n = 2000; p = 10
  X = matrix(rnorm(n*p), n, p)
  W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
  Y = pmax(X[,1], 0) * W + X[,2] + pmin(X[,3], 0) + rnorm(n)

double_ML(X, Y, W, method = "glmnet",
              k.fld = 2, simulations = 5,
              lambda.set.Y = 1,
              lambda.set.W = 1,
               Z.trans = F)
```

`fit_cross` *Cross-fitting helper*

Description

Implements k-fold cross-fitting with the supplied method, is a helper function that the user should not need.

Usage

```
fit_cross(Y_def, X_def, W_def, data.size, fun.call, k.folds, method_used)
```

Arguments

`Y_def`	An argument that control the Y input (used for argument passing)
`X_def`	An argument that control the X input (used for argument passing)
`W_def`	An argument that control the W input (used for argument passing)
`data.size`	Parses the length of the dataset (nrow) for splitting.
`fun.call`	Designates the function to use for cross-fitting.
`k.folds`	The number of k-folds for daataset splitting, defaults to 3.
`method_used`	The method to used when applying predict trough a helper function (do not worry about this!).

Details

This only implements the k-fold crossfitting, not the n.repeat simulation - if you intend to use this function, it works as a 'naive' double machine learning.

Value

A list with four elements: The mean estimate of θ, the standard error of the mean estimate, the associated moment conditions, and the estimated heterogenous effects for a combined estimate from a simulation run.

Examples

```


  n = 2000; p = 10
  X = matrix(rnorm(n*p), n, p)
  W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
  Y = pmax(X[,1], 0) * W + X[,2] + pmin(X[,3], 0) + rnorm(n)


fit_cross(Y_def = Y, X_def = X, W_def = W, data.size = 2000, fun.call = glmnet_helper(X,Y,W),
k.folds = 3, method_used = "glmnet")
```

`glmnet_helper` *Helper function(glmnet).*

Description

Helper function that dispatches to glmnet for the post-double estimation.

Usage

```
glmnet_helper(X, Y, W, Z.trans = TRUE, cv.steps = 100,
  parallelize = FALSE, cores.to.use = NULL, lambda.set.Y = NULL,
  lambda.set.W = NULL)
```

Arguments

`X`	A matrix of covariates (must be all numeric)
`Y`	A vector of the target variable, of same length as the number of rows of Y, must be numeric
`W`	A vector of the treatment variable, of same length as the number of rows of X, must be numeric
`Z.trans`	A logical value indicating whether to standardize inputs, defaults to TRUE
`cv.steps`	The number of folds for k-fold cross-validation of the hyperparameter tuning, defaults to 100
`parallelize`	Whether to run the simulations in parallel, using every available core. Defaults to FALSE.
`cores.to.use`	The number of cores to use. If NULL (the default), uses the maximum number of cores detected by detectCores.
`lambda.set.Y`	Allows the user to specify lambda for the Y model, defaults to null.
`lambda.set.W`	Allows the user to specify lambda for the W model, defaults to null.

Details

This function does not support the full range of arguments to glmnet, intentionally. If you need something specific please refer to custom_generator.

Value

A list with two elements: The fitted W model and the fitted Y model.

Examples

```


  n = 2000; p = 10
  X = matrix(rnorm(n*p), n, p)
  W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
  Y = pmax(X[,1], 0) * W + X[,2] + pmin(X[,3], 0) + rnorm(n)

glmnet_helper(X = X, Y = Y, W = W, lambda.set.W = 0.7, lambda.set.Y = 0.5)
```

input_checker *Helper function for input checking*

Description

This functions checks the input to the double_select function for compliance with the default methods used, confirming that the inputs are viable. It is a helper - no interaction with the user should happen unless the user explicitly calls this function.

Usage

```
input_checker(X_matrix, Y_vector, W_vector, specify.custom,
  test.inputs = TRUE)
```

Arguments

`X_matrix`	A matrix of covariates (must be all numeric)
`Y_vector`	A vector of the target variable, of same length as the number of rows of X, must be numeric
`W_vector`	A vector of the treatment variable, of same length as the number of rows of X, must be numeric
`specify.custom`	disables input checking by default, and leaves it to the user.
`test.inputs`	If set to FALSE disables input checking, the default is TRUE.

Details

The function will throw a warning if it is disabled (the input is not checked) or stop the processing if the input is checked, but incorrect. Disables checking for custom functions by default.

Value

A message indicating the status of input checking, or lack thereof.

Examples

```
  n = 2000; p = 10
  X = matrix(rnorm(n*p), n, p)
  W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
  Y = pmax(X[,1], 0) * W + X[,2] + pmin(X[,3], 0) + rnorm(n)

input_checker(X_matrix = X, Y_vector = Y, W_vector = W,
specify.custom = NULL, test.inputs = TRUE)
```

nn_helper	*Helper function for neural networks fitted by neuralnet*

Description

Helper function that dispatches to neuralnet for the double ML estimation (see details).

Usage

```
nn_helper(X, Y, W, neural.net.Y = NULL, neural.net.W = NULL,
  standardize = TRUE, standardization.method = "min-max")
```

Arguments

X	A matrix of covariates (must be all numeric)
Y	A vector of the target variable, of same length as the number of rows of Y, must be numeric
W	A vector of the treatment variable, of same length as the number of rows of X, must be numeric
neural.net.Y	A model specification for Y, see neuralnet
neural.net.W	A model specification for W, see neuralnet
standardize	Whether to standardize the data before starting the computation, defaults to TRUE.
standardization.method	How to standardize data, defaults to min-max, also offers "Z-transform", "Unit-Scale" and "Mean-Scale"

Details

For a more steamlined usage, default arguments as implemented in the neuralnet package are passed to both networks during fitting, unless specified otherwise. Also, any attempt to set the formula or data arguments of neuralnet will be ignored and rewritted with internal structures. The function will print a warning if this happens.

Value

A list with two elements: The fitted W model and the fitted Y model.

Examples

```
  n = 2000; p = 3
  X = matrix(rnorm(n*p), n, p)
  W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
  Y = rbinom(n, 1, 0.2 + 0.2 * (X[,2] > 0) + W * 0.1)

# note that this neural network can fail to converge
```

```
nn_helper( X,
           Y,
           W,
           neural.net.W = list( act.fct = "logistic" ),
           neural.net.Y = list( act.fc = "logistic" ))
```

`ols_helper` *Helper function(least squares).*

Description

Helper function that dispatches to OLS for the post-double estimation.

Usage

```
ols_helper(X, Y, W)
```

Arguments

`X`	A matrix of covariates (must be all numeric)
`Y`	A vector of the target variable, of same length as the number of rows of Y, must be numeric
`W`	A vector of the treatment variable, of same length as the number of rows of X, must be numeric

Details

This function is mostly implemented to provide the option to use ols with cross fitting, though it is assumed that this will be rarely used.

Value

A list with two elements: The fitted W model and the fitted Y model.

Examples

```
  n = 2000; p = 10
  X = matrix(rnorm(n*p), n, p)
  W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
  Y = pmax(X[,1], 0) * W + X[,2] + pmin(X[,3], 0) + rnorm(n)

ols_helper(X = X,Y = Y, W = W)
```

`plot_ML`	*Plot the fitted Debiased ML treatment effects.*

Description

A function to plot the estimated treatment effects.

Usage

```
plot_ML(x, coloured.same = TRUE, palette = NULL, ...)
```

Arguments

`x`	The estimated ML_Treatment_effects object to plot
`coloured.same`	Whether to use the same colour for all plots, defaults to TRUE.
`palette`	An optional custom colour/vector of palette colour codes to be used for plotting.
`...`	Additional arguments (currently ignored).

Details

The plot samples a random colour from a predefined colourblind palette by default. You may use a custom colour/colour palette for plotting by supplying the palette argument. If you supply a palette, the colour will be used in the order in which they are specified. Palette lengths longer than the number of plots will be ignored.

Value

Nothing - as a side effect produces a plot.

Examples

```
n = 2000; p = 10
X = matrix(rnorm(n*p), n, p)
W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
Y = pmax(X[,1], 0) * W + X[,2] + pmin(X[,3], 0) + rnorm(n)

  ols_example <- double_ML(X, Y, W, method = c("ols"),
                           show.progress = FALSE,
                           k.fld = 2,
                           simulations = 10)

 plot_ML(ols_example)
```

rf_helper *Helper function(randomforest).*

Description

Helper function that dispatches to random forest (grf) for the post-double estimation.

Usage

```
rf_helper(X, Y, W, orthog.boost = FALSE, tree.n = 2000, tune = T,
  clustered = NULL)
```

Arguments

`X`	A matrix of covariates (must be all numeric)
`Y`	A vector of the target variable, of same length as the number of rows of Y, must be numeric
`W`	A vector of the treatment variable, of same length as the number of rows of X, must be numeric
`orthog.boost`	Whether to use orthogonal boosting, defaults to FALSE.
`tree.n`	Controls the number of trees grown, defaults to 2000.
`tune`	Whether to use hyperparameter tuning.
`clustered`	Whether to use cluster robust forests, defaults to FALSE.

Details

If you need to use something that is not a default argument here, please refer to custom_generator. Not using honesty is heavily advised, though, as that could lead to a very high splitting, and the honesty is used for essentialy the same reason as crossfitting.

Value

A list with two elements: The fitted W model and the fitted Y model.

Examples

```

  n = 2000; p = 10
  X = matrix(rnorm(n*p), n, p)
  W = rbinom(n, 1, 0.4 + 0.2 * (X[,1] > 0))
  Y = pmax(X[,1], 0) * W + X[,2] + pmin(X[,3], 0) + rnorm(n)



rf_helper( X = X, Y = Y, W = W, tree.n = 10)
```

Index