Package ‘drape’

November 29, 2024

Title Doubly Robust Average Partial Effects

Version 0.0.2

Description Doubly robust average partial effect estimation. This implementation contains methods for adding additional smoothness to plug-in regression procedures and for estimating score functions using smoothing splines. Details of the method can be found in Harvey Klyne and Rajen D. Shah (2023) <doi:10.48550/arXiv.2308.09207>.

License MIT + file LICENSE

Encoding UTF-8

BugReports https://github.com/harveyklyne/drape/issues

Suggests DoubleML, glmnet, graphics, hdi, knitr, Matrix, mlr3, paradox, partykit, rjson, rmarkdown, testthat (>= 3.0.0), tibble, xgboost

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.1.2

NeedsCompilation no

Author Harvey Klyne [aut, cre, cph]

Maintainer Harvey Klyne <hck33@cantab.ac.uk>

Repository CRAN

Date/Publication 2024-11-28 23:10:13 UTC

Contents

`basis_poly` *Estimate the score function of the d'th covariate using a polynomial basis.*

Description

Computes the score function estimate when rho(X) is assumed to lie within the span of the polynomial basis of X.

Usage

```
basis_poly(X, d, degree = 2, lambda = NULL)
```

Arguments

`X`	matrix of covariates.
`d`	integer index of covariate of interest.
`degree`	maximum degree of polynomial terms.
`lambda`	optional scalar penalty, if "NULL" chosen via cross-validation.

Value

list containing the estimated score function "rho", which takes matrix input and yields a vector of score estimates.

Examples

```
set.seed(0)
X <- matrix(stats::rnorm(200), ncol=4)
bs <- basis_poly(X=X, d=1, degree=2)
bs$rho(X)
```

---

compare *Generate simulation data and evaluate estimators, with sample splitting.*

Description

Generate simulation data and evaluate estimators, with sample splitting.

Usage

```
compare(n, ex_setting, f_setting, nfold = 5)
```

Arguments

n	integer number of samples. For "401k" ex_setting this is ignored and the whole data set is used.
ex_setting	string "normal", "mixture2", "mixture3", "logistic", "t4", "401k".
f_setting	string "plm", "additive", "interaction".
nfold	integer number of cross-validation folds.

Value

list containing estimates, standard error estimates, and sample theta (for debugging).

---

compare_evaluate *Evaluate estimators by training nuisance functions on training set and evaluating them on test set.*

Description

Evaluate estimators by training nuisance functions on training set and evaluating them on test set.

Usage

```
compare_evaluate(train, test, ex_setting, f_setting, regression, sm_bw_out)
```

Arguments

`train`	list containing vector of responses y and matrix of predictors X = (x,z).
`test`	list containing vector of responses y and matrix of predictors X = (x,z).
`ex_setting`	string "normal", "mixture2", "mixture3", "logistic", "t4", "401k".
`f_setting`	string "plm", "additive", "interaction".
`regression`	Optional fitted regression.
`sm_bw_out`	Output of cv_resmooth.

Value

list containing f, df, and score estimates evaluated on the test set.

`compare_lm` *Generate simulation data and evaluate OLS estimator.*

Description

Generate simulation data and evaluate OLS estimator.

Usage

```
compare_lm(n, ex_setting, f_setting)
```

Arguments

`n`	integer number of samples. For "401k" ex_setting this is ignored and the whole data set is used.
`ex_setting`	string "normal", "mixture2", "mixture3", "logistic", "t4", "401k".
`f_setting`	string "plm", "additive", "interaction".

Value

list containing estimate, standard error estimate, and sample theta (for debugging).

compare_partially_linear *Generate simulation data and evaluate partially linear estimator.*

Description

Generate simulation data and evaluate partially linear estimator.

Usage

```
compare_partially_linear(n, ex_setting, f_setting)
```

Arguments

`n`	integer number of samples. For "401k" ex_setting this is ignored and the whole data set is used.
`ex_setting`	string "normal", "mixture2", "mixture3", "logistic", "t4", "401k".
`f_setting`	string "plm", "additive", "interaction".

Value

list containing estimate, standard error estimate, and sample theta (for debugging).

compare_rothenhausler *Generate simulation data and evaluate Rothenhausler estimator.*

Description

Generate simulation data and evaluate Rothenhausler estimator.

Usage

```
compare_rothenhausler(n, ex_setting, f_setting)
```

Arguments

`n`	integer number of samples. For "401k" ex_setting this is ignored and the whole data set is used.
`ex_setting`	string "normal", "mixture2", "mixture3", "logistic", "t4", "401k".
`f_setting`	string "plm", "additive", "interaction".

Value

list containing estimate, standard error estimate, and sample theta (for debugging).

cv_resmooth *K-fold cross-validation for resmoothing bandwidth.*

Description

Picks the largest resmoothing bandwidth achieving a cross-validation score within some specified tolerance of the original regression.

Usage

```
cv_resmooth(
  X,
  y,
  d = 1,
  regression,
  tol = 2,
  prefit = FALSE,
  foldid = NULL,
  bw = exp(seq(-5, 2, 0.2))/(2 * sqrt(3)) * stats::sd(X[, d]),
  nfolds = 5L,
  n_points = 101,
  sd_trim = 5
)
```

Arguments

`X`	matrix of covariates.
`y`	vector of responses.
`d`	integer index of covariate to be smoothed along.
`regression`	If prefit = FALSE this is a function which takes input data of the form (X,y), and returns a prediction function. This prediction function itself accepts matrix input same width as X, and returns a vector of y-predictions, and optionally a vector of derivative predictions. If prefit = TRUE then this is a list of length nfolds with each entry containing a component "fit" consisting of a prediction function taking matrix input and returning a vector.
`tol`	vector of tolerances controlling the degree of permissible cross-validation error increase. Larger values lead to a larger amount of smoothing being selected.
`prefit`	boolean signifying if the regressions are already fit to the training data for each fold.
`foldid`	optional vector with components in 1:nfolds indicating the folds in which each observation fell. Overwrites nfolds.
`bw`	vector of bandwidths for the Gaussian resmoothing kernel.
`nfolds`	integer number of cross-validation folds.
`n_points`	integer number of gridpoints to be used for convolution.
`sd_trim`	float number of standard deviations at which to trim the Gaussian distribution.

Value

list. Vector "bw" of bandwidths used. Vectors "cv" of cross-validation scores and numeric "cv_unsm" for the cross-validation without any smoothing. Vector "bw_opt_inds" for the indices of the selected bandwidths under various tolerances. Vector "bw_opt" for the corresponding bandwidths.

Examples

```
X <- matrix(stats::rnorm(200), ncol=2)
y <- X[,1] + sin(X[,2]) + 0.5 * stats::rnorm(nrow(X))
reg <- function(X,y){
    df <- data.frame(y,X)
    colnames(df) <- c("y", "X1", "X2")
    lm1 <- stats::lm(y~X1+sin(X2), data=df)
    fit <- function(newX){
        newdf = data.frame(newX)
        colnames(newdf) <- c("X1", "X2")
        return(as.vector(stats::predict(lm1, newdata=newdf)))}
    return(list("fit"=fit))
}
cv_resmooth(X=X, y=y, d=2, regression=reg, tol = c(0.5, 1, 2))
```

---

cv_spline_score *K-fold cross-validation for spline_score.*

Description

K-fold cross-validation for spline_score.

Usage

```
cv_spline_score(x, df = 2:15, nfolds = 5L, tol = 0.001, nmax = NULL)
```

Arguments

x	vector of datapoints
df	vector of smoothing parameters for the non-parametric score estimator, corresponding to the effective degrees of freedom for a smoothing spline.
nfolds	integer number of cross-validation folds.
tol	numeric tolerance, minimum distance between neighbouring points, to avoid singularities.
nmax	if specified, overrides tol as maximal number of unique points.

Value

list of 5 elements: df vector, cv vector of corresponding cross-validation scores, se vector of standard error estimates, df_min cross-validation minimiser, df_1se largest smoothing parameter within CV score within one standard error of df_min.

Examples

```
set.seed(0)
x <- stats::rt(100, df=4)
cv_spline_score(x)

x <- stats::rlogis(500)
cvspl <- cv_spline_score(x)
cvspl$df_min
```

---

drape — *Estimate the doubly-robust average partial effect estimate of X on Y, in the presence of Z.*

Description

Estimate the doubly-robust average partial effect estimate of X on Y, in the presence of Z.

Usage

```
drape(
  y,
  x,
  z,
  response_regression,
  predictor_regression,
  resmooth_bw = NULL,
  spline_df = NULL,
  nfolds = 5L,
  foldid = NULL,
  verbose = FALSE
)
```

Arguments

`y` vector of responses.

`x` vector of the predictor of interest.

`z` matrix of additional predictors.

`response_regression`
function which takes input data of the form (X,y), where X=cbind(x,z), and returns a prediction function f:X -> y and optionally a similar derivative estimation function (in this case no resmoothing is done).

`predictor_regression`
function which takes input data of the form (z,x), and returns a prediction function m:z -> x.

`resmooth_bw` optional numeric to be used as resmoothing bandwidth, otherwise chosen via cross-validation. Only used if response_regression doesn't predict derivatives.

spline_df	optional double, a smoothing parameter for the unconditional spline score estimator, corresponding to the effective degrees of freedom for a smoothing spline. If NULL, chosen via cross-validation.
nfolds	integer, number of sample-splits. If set to one, then all data is used for both training and evaluation.
foldid	optional vector with components in 1:nfolds indicating the folds in which each observation fell. Overwrites nfolds.
verbose	boolean controlling level of information outputted.

Value

list containing the average partial effect estimate and the corresponding standard error estimate. If verbose=TRUE, additionally contains variables used in computations.

Examples

```
set.seed(0)
data <- simulate_data(200, "normal", "plm")
response_regression <- function(X,y){
    df <- data.frame(y,X)
    colnames(df) <- c("y", paste0("X", 1:10))
    lm1 <- stats::lm(y~X1+sin(X2), data=df)
    fit <- function(newX){
        newdf <- data.frame(newX)
        colnames(newdf) <- paste0("X", 1:10)
        return(as.vector(stats::predict(lm1, newdata=newdf)))}
    return(list("fit"=fit))
}
predictor_regression <- function(z,x){
    df <- data.frame(x,z)
    colnames(df) <- c("x", paste0("Z", 1:9))
    lm1 <- stats::lm(x~Z1+Z2, data=df)
    fit <- function(newz){
        newdf <- data.frame(newz)
        colnames(newdf) <- paste0("Z", 1:9)
        return(as.vector(stats::predict(lm1, newdata=newdf)))}
    return(list("fit"=fit))
}
drape(data$y, data$x, data$z, response_regression, predictor_regression, nfolds=2)
```

`fit_lasso_poly` *Fit a lasso regression using quadratic polynomial basis, with interactions.*

Description

Compute regression function and derivative estimates based on polynomial basis lasso with penalty parameter chosen by cross validation (CV).

Usage

```
fit_lasso_poly(X, y, degree, lambda = NULL)
```

Arguments

X	matrix of covariates.
y	vector of responses.
degree	maximum degree of polynomial terms.
lambda	optional scalar tuning parameter, if "NULL" chosen via cross-validation.

Value

List containing: A function "fit" which takes matrix input of the same width as X, and returns a vector of y-predictions. A scalar "lambda" the tuning parameter.

---

fit_xgboost *Fit pre-tuned XGBoost regression for use in simulations.*

Description

Fit pre-tuned XGBoost regression for use in simulations.

Usage

```
fit_xgboost(X, y, params, derivative = FALSE)
```

Arguments

X	matrix of covariates.
y	vector of responses.
params	XGBoost hyperparameters.
derivative	logical determining if numerical difference derivative estimate (wrt the first predictor) should also be returned.

Value

list containing a function "fit" which takes matrix input of the same width as X, and returns a vector of predictions. Optionally the list also contains a function "deriv_fit" for numerical difference derivative estimates.

MC_sums — *Compute sums of a Monte Carlo vector for use in resmoothing.*

Description

Compute sums of a Monte Carlo vector for use in resmoothing.

Usage

```
MC_sums(a, n, nMC, nbw)
```

Arguments

`a`	vector of length (n x nMC x nbw).
`n`	integer.
`nMC`	integer.
`nbw`	integer.

Value

list with nbw elements. The j'th element of which is a vector of length n, the i'th element being the sum of the (((j-1)n + (i-1)) x nMC + 1) to (((j-1)n + i) x nMC) elements of a inclusive.

mixture_score — *Population score function for the symmetric mixture two Gaussian random variables.*

Description

Population score function for the symmetric mixture two Gaussian random variables.

Usage

```
mixture_score(x, sd)
```

Arguments

`x`	vector of observations.
`sd`	standard deviation of each Gaussian.

Value

vector of length n

new_X	*Generate a matrix of covariates for use in resmoothing, in which the d'th column of X is translated successively by the Kronecker product of bw and MC_variates.*

Description

Generate a matrix of covariates for use in resmoothing, in which the d'th column of X is translated successively by the Kronecker product of bw and MC_variates.

Usage

```
new_X(X, d, MC_variates, bw)
```

Arguments

X	matrix of covariates.
d	integer index of covariate to be smoothed along.
MC_variates	vector of standard Gaussian rvs.
bw	vector of bandwidths for the Gaussian kernel.

Value

matrix with ncol(X) columns and (nrow(X)length(MC_variates) length(bw)) rows.

ng_pseudo_response	*Generate pseudo responses as in Ng 1994 to enable univariate score estimation by standard smoothing spline regression.*

Description

Pseudo responses should be regarded as a computational tool, not as an estimate of the score itself.

Usage

```
ng_pseudo_response(x, w = rep(1, length(x)))
```

Arguments

x	vector of covariates.
w	vector of weights.

Value

A vector of score estimates.

Examples

```
x <- seq(-3,3, length.out=50)
ng_pseudo_response(x)
```

`partially_linear` *Fit a doubly-robust partially linear regression using the DoubleML package and pre-tuned XGBoost regressions, for use in simulations.*

Description

Fit a doubly-robust partially linear regression using the DoubleML package and pre-tuned XGBoost regressions, for use in simulations.

Usage

```
partially_linear(X, y, g_params, m_params)
```

Arguments

`X`	matrix of covariates.
`y`	vector of responses.
`g_params`	XGBoost hyperparameters for partially linear regression of y on X.
`m_params`	XGBoost hyperparameters for predictor regression of the first column of X on the others.

Value

List containing the linear parameter estimate and the corresponding standard error estimate.

`resmooth` *Resmooth the predictions of a fitted model*

Description

Smooth the predictions of a fitted model by convolving them with a Gaussian kernel along the d'th covariate.

Usage

```
resmooth(
  fit,
  X,
  d = 1,
  bw = exp(seq(-1, 1))/(2 * sqrt(3)) * stats::sd(X[, d]),
  n_points = 101,
  sd_trim = 5
)
```

Arguments

fit	a prediction function taking matrix input and returning a vector.
X	matrix of covariates.
d	integer index of covariate to be smoothed along.
bw	vector of bandwidths for the Gaussian kernel.
n_points	integer number of gridpoints to be used for convolution.
sd_trim	float number of standard deviations at which to trim the Gaussian distribution.

Value

List with the following elements. A list "pred" of the same length as "bw". Each element is a vector of predictions which are smooth with respect to the dth column of X, with smoothness corresponding to the respective element of "bw". A similar list "deriv" of corresponding vectors of first derivatives. Vectors "gridpoints" and "prob_weights" of equally spaced gridpoints and corresponding normal density weights. Vector "bw" of bandwidths used.

Examples

```
# Single bandwidth
X <- matrix(seq(-2,2,by=0.05))
fit <- function(Y){1*(rowMeans(Y)<0)}
sm <- resmooth(fit=fit, X=X, d=1, bw=0.2)
sm$pred[[1]]

# Multiple bandwidths simultaneously
X <- matrix(stats::rnorm(200), ncol=2)
y <- X[,1] + sin(X[,2]) + 0.5 * stats::rnorm(nrow(X))
df <- data.frame(y,X)
lm1 <- stats::lm(y~X1+sin(X2), data=df)
fit <- function(Y){as.vector(stats::predict(lm1, newdata=data.frame(Y)))}
resmooth(fit=fit, X=X, d=2)
```

---

rmixture — *Symmetric mixture two Gaussian random variables.*

Description

The resulting distribution is mean zero, variance one. X ~ N(-sqrt(1-sd^2),sd^2) wp 0.5, N(sqrt(1-sd^2),sd^2) wp 0.5.

Usage

```
rmixture(n, sd)
```

Arguments

n	number of observations.
sd	standard deviation of each Gaussian.

Value

vector of length n

`rothenhausler_basis` *Generate the modified quadratic basis of Rothenhausler and Yu.*

Description

Generate the modified quadratic basis of Rothenhausler and Yu.

Usage

```
rothenhausler_basis(X)
```

Arguments

`X`	matrix of covariates.

Value

List containing the modified basis matrices for regression and derivative estimation.

`rothenhausler_yu` *Estimate the average partial effect of using the debiased lasso method of Rothenhausler and Yu, using pre-tuned lasso penalties, for use in simulations.*

Description

Estimate the average partial effect of using the debiased lasso method of Rothenhausler and Yu, using pre-tuned lasso penalties, for use in simulations.

Usage

```
rothenhausler_yu(X, y, f_lambda, m_lambda)
```

Arguments

`X`	matrix of covariates.
`y`	vector of responses.
`f_lambda`	lasso penalty for regression of y on X.
`m_lambda`	lasso penalty for predictor regression of the first column of X on the others.

Value

List containing the linear parameter estimate and the corresponding standard error estimate.

simulate_data *Generate simulation data.*

Description

If ex_setting = "401k" then 401k data set is used for (X,Z). Otherwise:

$$Z\ N_9(0, \Sigma),$$

where $\Sigma_{jj} = 1$, $\Sigma_{jk} = 0.5$ for all j not equal to k.

$$X = m(Z) + s(Z) * ex,$$

where m and s are step functions of z_1 and z_3 respectively.

$$Y = f(X, Z) + N(0, 1).$$

Usage

```
simulate_data(n, ex_setting, f_setting)
```

Arguments

`n`	integer number of samples. For "401k" ex_setting this is ignored and the whole data set is used.
`ex_setting`	string "normal", "mixture2", "mixture3", "logistic", "t4", "401k".
`f_setting`	string "plm", "additive", "interaction".

Value

list containing y, x, z. Additionally contains the population nuisance parameters evaluated on the data, and the sample version of the average partial effect.

Examples

```
simulate_data(100, "normal", "plm")
```

`sort_bin` *Sort and bin x within a specified tolerance, using hist().*

Description

Sort and bin x within a specified tolerance, using hist().

Usage

```
sort_bin(x, tol = 1e-05, nmax = NULL)
```

Arguments

`x`	vector of covariates.
`tol`	numeric tolerance, minimum distance between neighbouring points, to avoid singularities.
`nmax`	if specified, overrides tol as maximal number of unique points.

Value

list with three elements. x_sort is sorted and binned x, w is a vector of weights corresponding to the frequency of each bin, order is a vector specifying the ordering of x into the binned values sort_x.

`spline_score` *Univariate score estimation via the smoothing spline method of Cox 1985 and Ng 1994.*

Description

Univariate score estimation via the smoothing spline method of Cox 1985 and Ng 1994.

Usage

```
spline_score(x, df = 5, tol = 0.001, nmax = NULL)
```

Arguments

`x`	vector of datapoints
`df`	vector of smoothing parameters for the non-parametric score estimator, corresponding to the effective degrees of freedom for a smoothing spline.
`tol`	numeric tolerance, minimum distance between neighbouring points, to avoid singularities.
`nmax`	if specified, overrides tol as maximal number of unique points.

Value

score function "rho" and derivative "drho", which take vector input and yield a vector of score estimates corresponding to each df (in a list if there are multiple df values). Also output the vector "df".

Examples

```
# Single bandwidth
x <- stats::rlogis(100)
spl <- spline_score(x, df=6)
spl$rho(x)
spl$drho(x)

# Multiple bandwidths simultaneously
x <- stats::rt(n=100, df=4)
spl <- spline_score(x, df=c(2,5,10))
spl$rho(x)
```

---

z_correlated_normal   *Generate n copies of $Z\tilde{}N\_p(0,\Sigma)$, where $\Sigma\_jj = 1$, $\Sigma\_jk = corr$ for all j not equal to k.*

Description

Generate n copies of Z $N_p(0,\Sigma)$, where $\Sigma_{jj} = 1$, $\Sigma_{jk} =$ corr for all j not equal to k.

Usage

```
z_correlated_normal(n, p, corr)
```

Arguments

n	integer number of samples.
p	integer number of dimensions.
corr	float correlation in (-1,1).

Value

n by p matrix.

Index