Package ‘GPCERF’

January 22, 2023

Title Gaussian Processes for Estimating Causal Exposure Response
Curves

Version 0.2.0

Maintainer Naeem Khoshnevis <nkhoshnevis@g.harvard.edu>

Description Provides a non-parametric Bayesian framework based on Gaussian process priors for estimating causal effects of a continuous exposure and detecting change points in the causal exposure response curves using observational data. Ren, B., Wu, X., Braun, D., Pillai, N., & Dominici, F.(2021). ``Bayesian modeling for exposure response curve via gaussian processes: Causal effects of exposure to air pollution on health outcomes.'' arXiv preprint <arXiv:2105.03454>.

License GPL (>= 3)

Language en-US

URL `https://github.com/NSAPH-Software/GPCERF`

BugReports `https://github.com/NSAPH-Software/GPCERF/issues`

Copyright Harvard University

Imports parallel, xgboost, stats, MASS, spatstat.geom, logger, Rcpp,
RcppArmadillo, ggplot2, rlang, Rfast, SuperLearner

Encoding UTF-8

RoxygenNote 7.2.1

Depends R (>= 3.5.0)

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

LinkingTo RcppArmadillo, Rcpp

NeedsCompilation yes

Author Naeem Khoshnevis [aut, cre] (<`https://orcid.org/0000-0003-4315-1426`>, HUIT),
Boyu Ren [aut] (<`https://orcid.org/0000-0002-5300-1184`>, McLean Hospital),
Tanujit Dey [ctb] (<`https://orcid.org/0000-0001-5559-211X`>, HMS),
Danielle Braun [aut] (<`https://orcid.org/0000-0002-5177-8598`>, HSPH)

Repository CRAN

Date/Publication 2023-01-22 08:30:02 UTC

R topics documented:

`GPCERF-package` *The 'GPCERF' package.*

Description

Provides a non-parametric Bayesian framework based on Gaussian process priors for estimating causal effects of a continuous exposure and detecting change points in the causal exposure response curves using observational data.

Author(s)

Naeem Khoshnevis

Boyu Ren

Danielle Braun

References

Ren, B., Wu, X., Braun, D., Pillai, N. and Dominici, F., 2021. Bayesian modeling for exposure response curve via gaussian processes: Causal effects of exposure to air pollution on health outcomes. arXiv preprint arXiv:2105.03454.

`compute_rl_deriv_gp` *Detect change-point in full GP*

Description

Calculates the posterior mean of the difference between left- and right-derivatives at an exposure level for the detection of change points.

Usage

```
compute_rl_deriv_gp(
  w,
  w_obs,
  y_obs,
  GPS_m,
  hyperparam,
  kernel_fn = function(x) exp(-x),
  kernel_deriv_fn = function(x) -exp(-x)
)
```

Arguments

`w`	A scalar of exposure level of interest.
`w_obs`	A vector of observed exposure levels of all samples.
`y_obs`	A vector of observed outcome values of all samples.
`GPS_m`	A data.frame of GPS vectors. • Column 1: GPS • Column 2: Prediction of exposure for covariate of each data sample (e_gps_pred). • Column 3: Standard deviation of e_gps (e_gps_std)
`hyperparam`	A vector of hyper-parameters in the GP model.
`kernel_fn`	The covariance function.
`kernel_deriv_fn`	The partial derivative of the covariance function.

Value

A numeric value of the posterior mean of the difference between two one-sided derivatives.

Examples

```
set.seed(847)
data <- generate_synthetic_data(sample_size = 100)
GPS_m <- train_gps(cov_mt = data[,-(1:2)],
                   w_all = data$treat,
                   sl_lib = c("SL.xgboost"),
                    dnorm_log = FALSE)

wi <- 8.6

val <- compute_rl_deriv_gp(w = wi,
                           w_obs = data$treat,
                           y_obs = data$Y,
                           GPS_m = GPS_m,
                           hyperparam = c(1,1,2))
```

`compute_rl_deriv_nn` *Calculate right minus left derivatives for change-point detection in nnGP*

Description

Calculates the posterior mean of the difference between left- and right-derivatives at an exposure level for the detection of change points. nnGP approximation is used.

Usage

```
compute_rl_deriv_nn(
  w,
  w_obs,
  GPS_m,
  y_obs,
  hyperparam,
  n_neighbor,
  expand,
  block_size,
  kernel_fn = function(x) exp(-x),
  kernel_deriv_fn = function(x) -exp(-x)
)
```

Arguments

`w`	A scalar of exposure level of interest.
`w_obs`	A vector of observed exposure levels of all samples.
`GPS_m`	A data.frame of GPS vectors. • Column 1: GPS values. • Column 2: Prediction of exposure for covariate of each data sample (e_gps_pred). • Column 3: Standard deviation of e_gps (e_gps_std).
`y_obs`	A vector of observed outcome values.
`hyperparam`	A vector of hyper-parameters in the GP model.
`n_neighbor`	The number of nearest neighbors on one side (see also `expand`).

`expand`	A scaling factor to determine the total number of nearest neighbors. The total is `2*expand*n_neighbor`.
`block_size`	The number of samples included in a computation block. Mainly used to balance the speed and memory requirement. Larger `block_size` is faster, but requires more memory.
`kernel_fn`	The covariance function. The input is the square of Euclidean distance.
`kernel_deriv_fn`	The partial derivative of the covariance function. The input is the square of Euclidean distance.

Value

A numeric value of the posterior mean of the difference between two one-sided derivatives.

Examples

```
set.seed(325)
data <- generate_synthetic_data(sample_size = 200)
GPS_m <- train_gps(cov_mt = data[,-(1:2)],
                   w_all = data$treat,
                   sl_lib = c("SL.xgboost"),
                   dnorm_log = FALSE)

wi <- 12.2

deriv_val <- compute_rl_deriv_nn(w = wi,
                                 w_obs = data$treat,
                                 GPS_m = GPS_m,
                                 y_obs = data$Y,
                                 hyperparam = c(0.2,0.4,1.2),
                                 n_neighbor = 20,
                                 expand = 1,
                                 block_size = 10)
```

`compute_w_corr` *Compute weighted correlation*

Description

Computes weighted correlation of the observational data based on weights achieved by Gaussian Process.

Usage

```
compute_w_corr(w, confounders, weights)
```

Arguments

`w`	A vector of exposure values for the observed data.
`confounders`	A data.frame of observational confounders.
`weights`	A vector of weights for each observation data.

Value

A vector of covariate balance.

Examples

```
set.seed(124)
mydata <- generate_synthetic_data(sample_size = 200)
weights <- runif(nrow(mydata))
compute_w_corr(mydata$treat,
               mydata[, 3:ncol(mydata)],
               weights)
```

`estimate_cerf_gp` *Estimate the conditional exposure response function using Gaussian process*

Description

Estimates the conditional exposure response function (cerf) using Gaussian Process (gp). The function tune the best match (the lowest covariate balance) for the provided set of hyperparameters.

Usage

```
estimate_cerf_gp(
  data,
  w,
  GPS_m,
  params,
  nthread = 1,
  kernel_fn = function(x) exp(-x^2)
)
```

Arguments

`data` A data.frame of observation data.

- Column 1: Outcome (Y)
- Column 2: Exposure or treatment (w)
- Column 3~m: Confounders (C)

`w` A vector of exposure level to compute CERF.

`GPS_m` A data.frame of GPS vectors.

- Column 1: GPS
- Column 2: Prediction of exposure for covariate of each data sample (e_gps_pred).
- Column 3: Standard deviation of e_gps (e_gps_std)

`params` A list of parameters that is required to run the process. These parameters include:

- alpha: A scaling factor for the GPS value.
- beta: A scaling factor for the exposure value.
- g_sigma: A scaling factor for kernel function (gamma/sigma).
- tune_app: A tuning approach. Available approaches:
  - all: try all combinations of hyperparameters. alpha, beta, and g_sigma can be a vector of parameters.

`nthread` An integer value that represents the number of threads to be used by internal packages.

`kernel_fn` A kernel function. A default value is a Gaussian Kernel.

Value

A cerf_gp object that includes the following values:

- w, the vector of exposure levels.
- pst_mean, Computed mean for the w vector.
- pst_sd, Computed credible interval for the w vector.

Examples

```
set.seed(129)
data <- generate_synthetic_data(sample_size = 100, gps_spec = 3)


# Estimate GPS function
GPS_m <- train_gps(cov_mt = data[,-(1:2)],
                   w_all = data$treat,
                   sl_lib = c("SL.xgboost"),
                   dnorm_log = FALSE)

# exposure values
w_all <- seq(0,10,1)


cerf_gp_obj <- estimate_cerf_gp(data,
                                w_all,
                                GPS_m,
                                params = list(alpha = c(0.1),
                                              beta=0.2,
                                              g_sigma = 1,
                                tune_app = "all"),
                      nthread = 1)
```

`estimate_cerf_nngp` *Estimate the conditional exposure response function using nearest neighbor Gaussian process*

Description

Estimates the conditional exposure response function (cerf) using the nearest neighbor (nn) Gaussian Process (gp). The function tune the best match (the lowest covariate balance) for the provided set of hyperparameters.

Usage

```
estimate_cerf_nngp(
  data,
  w,
  GPS_m,
  params,
  formula,
  kernel_fn = function(x) exp(-x^2),
  nthread = 1
)
```

Arguments

`data` A data.frame of observation data.

- Column 1: Outcome (Y)
- Column 2: Exposure or treatment (w)
- Column 3~m: Confounders (C)

`w` A vector of exposure level to compute CERF.

`GPS_m` A data.frame of GPS vectors.

- Column 1: GPS
- Column 2: Prediction of exposure for covariate of each data sample (e_gps_pred).
- Column 3: Standard deviation of e_gps (e_gps_std)

`params` A list of parameters that is required to run the process. These parameters include:

- alpha: A scaling factor for the GPS value.
- beta: A scaling factor for the exposure value.
- g_sigma: A scaling factor for kernel function (gamma/sigma).
- tune_app: A tuning approach. Available approaches:

    - all: try all combinations of hyperparameters.
- expand: Scaling factor to determine the total number of nearest neighbors. The total is `2 * expand * n_neighbour`.
- n_neighbor: Number of nearest neighbors on one side.
- block_size: Number of samples included in a computation block. Mainly used to balance the speed and memory requirement. Larger `block_size` is faster, but requires more memory. alpha, beta, and g_sigma can be a vector of parameters.

`formula` A formula to indicate the design matrix of the model for GPS.

`kernel_fn` A kernel function. A default value is a Gaussian Kernel.

`nthread` An integer value that represents the number of threads to be used by internal packages.

Value

A cerf_nngp object that includes the following values:

- w, the vector of exposure levels.
- pst_mean, the computed mean for the w vector.
- pst_sd, the computed credible interval for the w vector.

Examples

```
set.seed(19)
data <- generate_synthetic_data(sample_size = 120, gps_spec = 3)
# Estimate GPS function
GPS_m <- train_gps(cov_mt = data[,-(1:2)],
                   w_all = data$treat,
                   sl_lib = c("SL.xgboost"),
                   dnorm_log = FALSE)
# exposure values
w.all <- seq(0,20,2)
cerf_nngp_obj <- estimate_cerf_nngp(data,
                                    w.all,
                                    GPS_m,
                                    params = list(alpha = c(0.1),
                                                  beta = 0.2,
                                                  g_sigma = 1,
                                                  tune_app = "all",
                                                  n_neighbor = 20,
                                                  expand = 1,
                                                  block_size = 1e4),
                                    formula = ~ . - 1 - Y - treat,
                                    nthread = 1)
```

`generate_synthetic_data` *Generate synthetic data for GPCERF package*

Description

Generates synthetic data set based on different GPS models and covariates.

Usage

```
generate_synthetic_data(
  sample_size = 1000,
  outcome_sd = 10,
  gps_spec = 1,
  cova_spec = 1
)
```

Arguments

`sample_size`	A number of data samples.
`outcome_sd`	Standard deviation used to generate the outcome in the synthetic data set.
`gps_spec`	A numeric value (1-6) that indicates the GPS model used to generate the continuous exposure.
`cova_spec`	A numeric value (1-2) to modify the covariates.

Value

A data frame of the synthetic data. Outcome is labeled as Y, exposure as w, and covariates cf1-6.

Examples

```
set.seed(351)
data <- generate_synthetic_data(sample_size = 200)
```

`get_logger` *Get logger settings*

Description

Returns current logger settings.

Usage

```
get_logger()
```

Value

Returns a list that includes **logger_file_path** and **logger_level**.

Examples

```
set_logger("mylogger.log", "INFO")
log_meta <- get_logger()
```

---

plot.cerf_gp *Extend generic plot functions for cerf_gp class*

Description

A wrapper function to extend generic plot functions for cerf_gp class.

Usage

```
## S3 method for class 'cerf_gp'
plot(x, ...)
```

Arguments

x	A cerf_gp object.
...	Additional arguments passed to customize the plot.

Value

Returns a ggplot2 object, invisibly. This function is called for side effects.

---

plot.cerf_nngp *Extend generic plot functions for cerf_nngp class*

Description

A wrapper function to extend generic plot functions for cerf_nngp class.

Usage

```
## S3 method for class 'cerf_nngp'
plot(x, ...)
```

Arguments

`x`	A cerf_nngp object.
`...`	Additional arguments passed to customize the plot.

Value

Returns a ggplot2 object, invisibly. This function is called for side effects.

`print.cerf_gp` *Extend print function for cerf_gp object*

Description

Extend print function for cerf_gp object

Usage

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

Arguments

`x`	A cerf_gp object.
`...`	Additional arguments passed to customize the results.

Value

No return value. This function is called for side effects.

`print.cerf_nngp` *Extend print function for cerf_nngp object*

Description

Extend print function for cerf_nngp object

Usage

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

Arguments

`x`	A cerf_nngp object.
`...`	Additional arguments passed to customize the results.

Value

No return value. This function is called for side effects.

`set_logger` *Set logger settings*

Description

Updates logger settings, including log level and location of the file.

Usage

```
set_logger(logger_file_path = "GPCERF.log", logger_level = "INFO")
```

Arguments

`logger_file_path`	A path (including file name) to log the messages. (Default: GPCERF.log)
`logger_level`	The log level. Available levels include: • TRACE • DEBUG • INFO (Default) • SUCCESS • WARN • ERROR • FATAL

Value

No return value. This function is called for side effects.

Examples

```
set_logger("mylogger.log", "INFO")
```

summary.cerf_gp *print summary of cerf_gp object*

Description

print summary of cerf_gp object

Usage

```
## S3 method for class 'cerf_gp'
summary(object, ...)
```

Arguments

`object`	A cerf_gp object.
`...`	Additional arguments passed to customize the results.

Value

Returns summary of data

summary.cerf_nngp *print summary of cerf_nngp object*

Description

print summary of cerf_nngp object

Usage

```
## S3 method for class 'cerf_nngp'
summary(object, ...)
```

Arguments

`object`	A cerf_nngp object.
`...`	Additional arguments passed to customize the results.

Value

Returns summary of data.

train_gps *Train a model for generalized propensity score*

Description

Estimates the conditional mean and sd of exposure level as a function of covariates.

Usage

```
train_gps(cov_mt, w_all, sl_lib, dnorm_log)
```

Arguments

`cov_mt`	A covariate matrix containing all covariates. Each row is a data sample and each column is a covariate.
`w_all`	A vector of observed exposure levels.
`sl_lib`	A vector of SuperLearner's package libraries.
`dnorm_log`	Logical, if TRUE, probabilities p are given as log(p).

Value

A data.frame that includes:

- a vector of estimated GPS at the observed exposure levels;
- a vector of estimated conditional means of exposure levels when the covariates are fixed at the observed values;
- estimated standard deviation of exposure levels

Examples

```
data <- generate_synthetic_data(sample_size = 200)
GPS_m <- train_gps(cov_mt = data[,-(1:2)],
                   w_all = data$treat,
                   sl_lib = c("SL.xgboost"),
                   dnorm_log = FALSE)
```

Index