Package 'GPCERF'

July 2, 2022

Title Gaussian Processes for Estimating Causal Exposure Response Curves

Version 0.1.0

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, data.table, xgboost, stats, MASS, spatstat.geom, logger, Rcpp, ggplot2, rlang, Matrix

Encoding UTF-8

RoxygenNote 7.1.2

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>, FASRC),
Boyu Ren [aut] (<https://orcid.org/0000-0002-5300-1184>, McLean Hospital),
Tanujt 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 2022-07-02 09:50:02 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_deriv_weights_gp`

Calculate Derivatives of CERF

Description

Calculates the weights assigned to each observed outcome when deriving the posterior mean of the first derivative of CERF at a given exposure level.

Usage

```
compute_deriv_weights_gp(
  w,
  w_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.
`GPS_m`	A data.table of GPS vectors. Including: • 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)
`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 vector of weights for all samples, based on which the posterior mean of the derivative of CERF at the exposure level of interest is calculated.

Examples

```
set.seed(915)
data <- generate_synthetic_data(sample_size = 150)
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

wi <- 4.8
weights <- compute_deriv_weights_gp(w = wi,
                                    w_obs = data$treat,
                                    GPS_m = GPS_m,
                                    hyperparam = c(1,1,2))
```

`compute_inverse` *Compute Matrix Inverse For a Covariate Matrix*

Description

Computes inverse of a covariate matrix using Choleski decomposition.

Usage

```
compute_inverse(mtrx)
```

Arguments

`mtrx`	An n*n covariate matrix

Value

Inverse matrix

Examples

```
set.seed(934)
A <- runif(10)
B <- runif(10)
C = cbind(A, B)
kernel_fn = function(x) exp(-x^2)
D = kernel_fn(as.matrix(dist(C)))
inv_sigma_obs <- compute_inverse(D)
```

`compute_m_sigma` *Compute mean, credible interval, and covariate balance in Full Gaussian Process (GP)*

Description

Calculates the induced covariate balance associated with one hyper-parameter configuration in full GP.

Usage

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

Arguments

`hyperparam`	A vector of values of hyper-parameters. • First element: alpha • Second element: beta • Third element: g_sigma (gamma/sigma)
`data`	A data.table containing all data including outcome, exposure and covariates. In the following order: • Column 1: Outcome (Y) • Column 2: Exposure or treatment (w) • Column 3~m: Confounders (C)
`w`	A vector of exposure levels at which the CERF is estimated.
`GPS_m`	A data.table of GPS vectors. • Column 1: A vector of estimated GPS evaluated at the observed exposure levels. • Column 2: Estimated conditional means of the exposure given covariates for all samples (e_gps_pred). • Column 3: Estimated conditional standard deviation of the exposure given covariates for all samples (e_gps_std).
`nthread`	An integer value that represents the number of threads to be used by internal packages.
`kernel_fn`	The covariance function of GP.

Value

A list containing two elements: 1) a vector of absolute weighted correlation of each covariate to the exposure, which is the metric for covariate balance and 2) the estimated CERF at `w.all` based on the hyper-parameter values in `param`.

Examples

```
set.seed(912)
data <- generate_synthetic_data(sample_size = 250, gps_spec = 3)

w_all <- seq(0,20,1)

data.table::setDT(data)

#Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

tune_res <- compute_m_sigma(hyperparam = c(0.09, 0.09, 10),
                            data = data,
                            w = w_all,
                            GPS_m = GPS_m,
                            nthread = 1)

gp.cerf <- tune_res$est
```

---

compute_posterior_m_nn

Calculate Posterior Means for nnGP Model

Description

Calculates the posterior mean of a point on the CERF based on the nnGP model. This function also returns the weights assigned to all nearest neighbors when calculating the posterior mean.

Usage

```
compute_posterior_m_nn(
  hyperparam,
  w,
  GPS_w,
  obs_ord,
  y_obs_ord,
  n_neighbor = 10,
  expand = 5,
  block_size = 10000
)
```

Arguments

hyperparam	A set of hyperparameters in the GP model.
w	A scaler representing the exposure level for the point of interest on the CERF.
GPS_w	The GPS for all samples when their exposure levels are set at w.
obs_ord	A matrix of two columns. First column is the observed exposure levels of all samples; second is the GPS at the observed exposure levels. The rows are in ascending order for the first column.
y_obs_ord	A vector of observed outcome values. The vector is ordered as obs_ord.
n_neighbor	The number of nearest neighbors on one side (see also expand).
expand	Scaling factor to determine the total number of nearest neighbors. The total is 2*expand*n_neighbor.
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.

Value

A two-column matrix. The first column is the weights assigned to each nearest neighbor. The second column is the corresponding observed outcome value. The weight in the last row of this matrix is NA and the observed outcome value is the estimated posterior mean of the CERF at point w, which is the weighted sum of all observed outcome values of the neighbors.

Examples

```
set.seed(1029)
data <- generate_synthetic_data(sample_size = 150, gps_spec = 3)

# Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

# Hyperparameter
hyperparam <- c(0.1, 0.2, 1)
n_neighbor <- 10
expand <- 1
block_size <- 10000

# Exposure level
wi <- 0.4

# Estimate GPS for the exposure level
GPS_w <- dnorm(wi,
               mean = GPS_m$e_gps_pred,
               sd = GPS_m$e_gps_std, log = TRUE)

# Order data for easy selection
coord_obs = cbind(data$treat, GPS_m$GPS)
```

```
y_use <- data$Y

obs_ord <- coord_obs[order(coord_obs[,1]),]
y_use_ord <- y_use[order(coord_obs[,1])]

val <- compute_posterior_m_nn(hyperparam = hyperparam,
                              w = wi,
                              GPS_w = GPS_w,
                              obs_ord = obs_ord,
                              y_obs_ord = y_use_ord,
                              n_neighbor = n_neighbor,
                              expand = expand,
                              block_size = block_size)
```

`compute_posterior_sd_nn` *Calculate Posterior Standard Deviations for nnGP Model*

Description

Calculates the posterior standard deviation of a point on the CERF based on the nnGP model.

Usage

```
compute_posterior_sd_nn(
  hyperparam,
  w,
  GPS_w,
  obs_ord,
  sigma2,
  n_neighbor = 10,
  expand = 1
)
```

Arguments

`hyperparam`	The values of hyperparameters in the GP model.
`w`	The exposure level for the point of interest on the CERF.
`GPS_w`	The GPS for all samples when their exposure levels are set at `w`.
`obs_ord`	A matrix of two columns. The first column is the observed exposure levels of all samples; the second is the GPS at the observed exposure levels. The rows are in ascending order for the first column.
`sigma2`	A scaler representing `sigma^2`.
`n_neighbor`	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`.

Value

The posterior standard deviation of the estimated CERF at w.

Examples

```
set.seed(3089)
data <- generate_synthetic_data(sample_size = 150, gps_spec = 3)

# Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

# Hyperparameter
hyperparam <- c(0.1, 0.2, 1)
n_neighbor <- 10
expand <- 1
block_size <- 10000

# Exposure level
wi <- 0.4

# Estimate GPS for the exposure level
GPS_w = dnorm(wi,
              mean = GPS_m$e_gps_pred,
              sd = GPS_m$e_gps_std, log = TRUE)

# Order data for easy selection
coord_obs = cbind(data$treat, GPS_m$GPS)
y_use <- data$Y

obs_ord <- coord_obs[order(coord_obs[,1]),]
y_use_ord <- y_use[order(coord_obs[,1])]

# compute noise
noise <- estimate_noise_nn(hyperparam = hyperparam,
                           w_obs = data$treat,
                           GPS_obs = GPS_m$GPS,
                           y_obs = y_use_ord,
                           n_neighbor = n_neighbor)

# compute posterior standard deviation
pst_sd <- compute_posterior_sd_nn(hyperparam = hyperparam,
                                  w = wi,
                                  GPS_w = GPS_w,
                                  obs_ord = obs_ord,
                                  sigma2 = noise,
                                  n_neighbor = 20,
                                  expand = 1)
```

`compute_rl_deriv_gp` *Change-point Detection 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.table 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 = 200)
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))
```

```
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.table 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 = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

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 = 1000)
```

---

compute_weight_gp *Calculate Weights for Estimation of a Point on CERF*

Description

Calculates the weights of observed outcomes which is then used to estimate the posterior mean of CERF at a given exposure level.

Usage

```
compute_weight_gp(
  w,
  w_obs,
  scaled_obs,
  hyperparam,
  inv_sigma_obs,
```

```
  GPS_m,
  kernel_fn = function(x) exp(-x^2)
)
```

Arguments

w	A scalar of exposure level of interest.
w_obs	A vector of observed exposure levels of all samples.
scaled_obs	A matrix of two columns. • First column is the scaled GPS value of all samples (GPS * 1/sqrt(alpha)) • Second column is the scaled exposure value of all samples (w * 1/sqrt(beta))
hyperparam	A vector of hyper-parameters for the GP. • First element: alpha • Second element: beta • Third element: gamma/sigma
inv_sigma_obs	Inverse of the covariance matrix between observed samples.
GPS_m	A data.table of GPS vectors. • Column 1: A vector of estimated GPS evaluated at the observed exposure levels. • Column 2: Estimated conditional means of the exposure given covariates for all samples (e_gps_pred). • Column 3: Estimated conditional standard deviation of the exposure given covariates for all samples (e_gps_std).
kernel_fn	The covariance function of GP.

Value

A vector of the weights assigned to each sample for the calculate of posterior mean of CERF at w.

Examples

```
set.seed(814)
#Generate synthetic data
data <- generate_synthetic_data(sample_size = 200, gps_spec = 3)
w_obs <- obs_exposure <- data$treat

# Choose an exposure level to compute CERF
w = 1.8

# Define kernel function
kernel_fn <- function(x) exp(-x^2)

# Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))
```

```
GPS <- GPS_m$GPS

# set hyperparameters
hyperparam <- c(0.1, 0.4, 1)
alpha <- hyperparam[1]
beta <- hyperparam[2]
g_sigma <- hyperparam[3]

# Compute scaled observation data and inverse of covariate matrix.
scaled_obs <- cbind(obs_exposure*sqrt(1/alpha), GPS*sqrt(1/beta))
sigma_obs <- g_sigma*kernel_fn(as.matrix(dist(scaled_obs))) + diag(nrow(scaled_obs))
inv_sigma_obs <- compute_inverse(sigma_obs)


weight <- compute_weight_gp(w = w,
                            w_obs = w_obs,
                            scaled_obs = scaled_obs,
                            hyperparam = hyperparam,
                            inv_sigma_obs = inv_sigma_obs,
                            GPS_m = GPS_m,
                            kernel_fn = kernel_fn)
```

`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(data, weights)
```

Arguments

`data`	A data.table of observational data with the following columns: • Column 1: Outcome (Y) • Column 2: Exposure or treatment (w) • Column 3~m: Confounders (C)
`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)
data.table::setDT(mydata)
weights <- runif(nrow(mydata))
compute_w_corr(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.table 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.table 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)
sim.data <- generate_synthetic_data(sample_size = 200, gps_spec = 3)


# Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(sim.data[,-(1:2)]),
                   w_all = as.matrix(sim.data$treat))

# exposure values
w.all = seq(0,20,1)

data.table::setDT(sim.data)

cerf_gp_obj <- estimate_cerf_gp(sim.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, kernel_fn, nthread = 1)
```

Arguments

`data`	A data.table 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.table 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.
`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)
sim.data <- generate_synthetic_data(sample_size = 120, gps_spec = 3)
# Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(sim.data[,-(1:2)]),
                   w_all = as.matrix(sim.data$treat))
# exposure values
w.all <- seq(0,20,2)
data.table::setDT(sim.data)
cerf_nngp_obj <- estimate_cerf_nngp(sim.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),
                                    nthread = 1)
```

`estimate_mean_sd_nn` *Estimate the CERF with the nnGP Model*

Description

Estimates the posterior mean of the conditional exposure response function at specified exposure levels with nnGP.

Usage

```
estimate_mean_sd_nn(
  hyperparam,
  sigma2,
  w_obs,
  w,
  y_obs,
  GPS_m,
  n_neighbor = 50,
  expand = 2,
  block_size = 2000,
  nthread = 1
)
```

Arguments

hyperparam	A set of hyperparameters for the nnGP.
sigma2	A scaler representing sigma^2.
w_obs	A vector of observed exposure levels.
w	A vector of exposure levels at which the CERF is estimated.
y_obs	A vector of observed outcome values.
GPS_m	A data.table 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)
n_neighbor	The number of nearest neighbors on one side (see also expand).
expand	Scaling factor to determine the total number of nearest neighbors. The total is 2*expand*n.neighbour.
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.
nthread	An integer value that represents the number of threads to be used by internal packages.

Value

A list of 2 elements, including:

- the returned value from compute_posterior_m_nn
- the returned value from compute_posterior_sd_nn

Examples

```
set.seed(86)
data <- generate_synthetic_data(sample_size = 200, gps_spec = 3)

# Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

# Hyperparameter
hyperparam <- c(0.1, 0.2, 1)
n_neighbor <- 15
expand <- 1
block_size <- 10000

# compute noise
noise <- estimate_noise_nn(hyperparam = hyperparam,
                           w_obs = data$treat,
                           GPS_obs = GPS_m$GPS,
                           y_obs = data$Y,
```

```
                            n_neighbor = n_neighbor)

# compute posterior mean and standard deviation for vector of w.
w <- seq(0,20,1)
val <- estimate_mean_sd_nn(hyperparam = hyperparam,
                            sigma2 = noise,
                            w_obs = data$treat,
                            w = w,
                            y_obs = data$Y,
                            GPS_m = GPS_m,
                            n_neighbor = n_neighbor,
                            expand = expand,
                            block_size = block_size,
                            nthread = 1)
```

`estimate_noise_gp` *Estimate the Standard Deviation of the Nugget Term in Full Gaussian Process*

Description

Estimates the standard deviations of the nugget term in full GP by calculating the standard deviations of the residuals.

Usage

```
estimate_noise_gp(hyperparam, data, GPS)
```

Arguments

`hyperparam`	A vector of hyper-parameter values for the full GP.
`data`	A data.table of observation data. • Column 1: Outcome (Y) • Column 2: Exposure or treatment (w) • Column 3~m: Confounders (C)
`GPS`	A vector of estimated GPS at the observed exposure levels.

Value

A scalar of estimated standard deviation of the nugget term in full GP.

Examples

```
set.seed(109)
data <- generate_synthetic_data(sample_size = 100, gps_spec = 3)
data.table::setDT(data)

# Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

hyperparam <- c(0.1, 0.2, 1)

noise_est <- estimate_noise_gp(hyperparam, data, GPS_m$GPS)
```

`estimate_noise_nn` *Estimate the Standard Deviation (noise) of the Nugget Term in nnGP*

Description

Estimate the standard deviations of the nugget term (noise) in nnGP by calculating the standard deviations of the residuals.

Usage

```
estimate_noise_nn(hyperparam, w_obs, GPS_obs, y_obs, n_neighbor)
```

Arguments

`hyperparam`	A vector of hyper-parameter values.
`w_obs`	A vector of observed exposure levels.
`GPS_obs`	A vector of estimated GPS evaluated at the observed exposure levels.
`y_obs`	A vector of observed outcomes.
`n_neighbor`	Number of nearest neighbors on one side.

Value

A scalar of estimated standard deviation of the nugget term in nnGP.

Examples

```
set.seed(425)
data <- generate_synthetic_data(sample_size = 200, gps_spec = 3)

# Estimate GPS function
```

```
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

# Hyperparameter
hyperparam <- c(0.1, 0.2, 1)
n_neighbor <- 10
expand <- 1
block_size <- 10000

# Exposure level
wi <- 1.2

# Estimate GPS for the exposure level
GPS_w = dnorm(wi,
              mean = GPS_m$e_gps_pred,
              sd = GPS_m$e_gps_std, log = TRUE)

# Order data for easy selection
coord_obs = cbind(data$treat, GPS_m$GPS)
y_use <- data$Y

obs_ord <- coord_obs[order(coord_obs[,1]),]
y_use_ord <- y_use[order(coord_obs[,1])]

noise <- estimate_noise_nn(hyperparam = hyperparam,
                           w_obs = data$treat,
                           GPS_obs = GPS_m$GPS,
                           y_obs = y_use_ord,
                           n_neighbor = n_neighbor)
```

`find_optimal_nn` *Find the Optimal Hyper-parameter for the Nearest Neighbor Gaussian Process*

Description

Computes covariate balance for each combination of provided hyper-parameters and selects the hyper-parameter values that minimizes the covariate balance.

Usage

```
find_optimal_nn(
  w_obs,
  w,
  y_obs,
  GPS_m,
  design_mt,
 hyperparams = expand.grid(seq(0.5, 4.5, 1), seq(0.5, 4.5, 1), seq(0.5, 4.5, 1)),
  n_neighbor = 50,
```

```
  expand = 2,
  block_size = 2000,
  nthread = 1
)
```

Arguments

w_obs	A vector of the observed exposure levels.
w	A vector of exposure levels at which CERF will be estimated.
y_obs	A vector of observed outcomes
GPS_m	A data.table 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)
design_mt	The covariate matrix of all samples (intercept excluded).
hyperparams	A matrix of candidate values of the hyper-parameters, each row contains a set of values of all hyper-parameters.
n_neighbor	The number of nearest neighbors on one side (see also expand).
expand	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.
nthread	An integer value that represents the number of threads to be used by internal packages.

Value

Estimated covariate balance scores for the grid of hyper-parameter values considered in hyperparams.

Examples

```
set.seed(89)
data <- generate_synthetic_data(sample_size = 200, gps_spec = 3)

# Estimate GPS function
GPS_m <- train_GPS(cov_mt = as.matrix(data[,-(1:2)]),
                   w_all = as.matrix(data$treat))

# Hyperparameter
hyperparam <- c(0.1, 0.2, 1)
n_neighbor <- 10
expand <- 1
block_size <- 10000

# compute posterior mean and standard deviation for vector of w.
```

```
w <- seq(0,20,2)
design_mt <- model.matrix(~.-1, data = data[, 3:ncol(data)])

hyperparam_grid <- expand.grid(seq(0.5,1.0,1),
                               seq(0.4,0.6,0.2),
                               seq(0.5))

optimal_cb <- find_optimal_nn(w_obs = data$treat,
                              w = w,
                              y_obs = data$Y,
                              GPS_m = GPS_m,
                              design_mt = design_mt,
                              hyperparams = hyperparam_grid,
                              n_neighbor = 50, expand = 2, block_size = 2e3,
                              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`	Number of data samples.
`outcome_sd`	Standard deviation used to generate the outcome in the synthetic dataset.
`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)
mydata <- 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: CausalGPS.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 GPS*

Description

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

Usage

```
train_GPS(cov_mt, w_all, dnorm_log = FALSE)
```

Arguments

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

Value

A data.table 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

```
mydata <- generate_synthetic_data()
GPS_m <- train_GPS(as.matrix(mydata[,c("cf1", "cf2", "cf3", "cf4",
                                       "cf5", "cf6")]),
                   as.matrix(mydata$treat))
```

Index