Package ‘HCPclust’

January 30, 2026

Title Hierarchical Conformal Prediction for Clustered Data with Missing Responses

Version 0.1.1

Description Implements hierarchical conformal prediction for clustered data with missing responses. The method uses repeated cluster-level
splitting and within-cluster subsampling to accommodate dependence, and
inverse-probability weighting to correct distribution shift induced by missingness.
Conditional densities are estimated by inverting fitted conditional quantiles
(linear quantile regression or quantile regression forests), and p-values are
aggregated across resampling and splitting steps using the Cauchy combination test.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

URL `https://github.com/judywangstat/HCP`

BugReports `https://github.com/judywangstat/HCP/issues`

Imports stats, grf, quantreg, xgboost, quantregForest

Suggests foreach, doParallel, doRNG, parallel, testthat (>= 3.0.0), knitr, rmarkdown, FNN, rstudioapi

Config/testthat/edition 3

NeedsCompilation no

Author Menghan Yi [aut, cre],
Judy Wang [aut]

Maintainer Menghan Yi <`menghany@umich.edu`>

Repository CRAN

Date/Publication 2026-01-30 11:10:02 UTC

Contents

`fit_cond_density_quantile` *Estimate conditional density pi(y|x) via quantile process + quotient estimator*

Description

Fits a conditional quantile function $\widehat{Q}_Y(\tau \mid x)$ using pooled observed data (working-independence), and estimates the conditional density through the quotient estimator along the quantile curve:

$$\widehat{\pi}\{\widehat{Q}(\tau \mid x) \mid x\} = \frac{2h(\tau)}{\widehat{Q}(\tau + h(\tau) \mid x) - \widehat{Q}(\tau - h(\tau) \mid x)}.$$

For numerical stability, the quantile curve can be monotone-adjusted (isotonic regression), and tail decay extrapolation can be used before interpolation to $\pi(y \mid x)$.

Usage

```
fit_cond_density_quantile(
  dat,
  y_col = "Y",
  delta_col = "delta",
  x_cols,
  taus = seq(0.05, 0.95, by = 0.01),
  h = NULL,
  method = c("rq", "qrf"),
  enforce_monotone = TRUE,
  tail_decay = TRUE,
  num_extra_points = 10L,
  decay_factor = 0.8,
  dens_floor = 1e-10,
  eps = 1e-08,
  gap_min = 0.01,
  seed = NULL,
  ...
)
```

Arguments

`dat`	data.frame in long format, containing outcome, missingness indicator, and covariates.

`y_col`	name of outcome column (observed Y, may contain NA).
`delta_col`	name of missingness indicator (1 observed, 0 missing).
`x_cols`	character vector of covariate column names (include time if desired).
`taus`	grid of quantile levels in (0,1) at which the quantile process is evaluated.
`h`	Bandwidth(s) for quotient. Either a scalar or a numeric vector of length `length(taus)`. If `NULL`, a tau-specific bandwidth vector $h(\tau)$ is computed via `quantreg::bandwidth.rq`, and automatically shrunk near the boundaries to ensure $\tau \pm h(\tau) \in (0,1)$.
`method`	quantile engine: `"rq"` (linear quantile regression) or `"qrf"` (quantile random forest).
`enforce_monotone`	logical; if TRUE, apply isotonic regression to the predicted quantile curve in τ for each x to reduce quantile crossing.
`tail_decay`	logical; if TRUE, add extra tail points with geometric decay before interpolation.
`num_extra_points`	number of extra tail points on each side when `tail_decay=TRUE`.
`decay_factor`	decay factor in (0,1) for tail densities when `tail_decay=TRUE`.
`dens_floor`	lower bound for density to avoid numerical issues.
`eps`	small stabilizer for denominator `pmax(Qplus-Qminus, eps)`.
`gap_min`	minimum spacing for tail extrapolation points.
`seed`	optional seed.
`...`	extra arguments passed to the underlying quantile engine: `rq` passed to `quantreg::rq.fit`, e.g. `rq_method="br"`. `qrf` passed to `quantregForest::quantregForest`, e.g. `ntree=500`.

Value

A list containing fitted objects and prediction functions:

`predict_Q(x_new, taus_use)` Returns the estimated conditional quantiles

$$\widehat{Q}_Y(\tau \mid x)$$

for $\tau \in (0,1)$ specified by `taus_use`, evaluated at new covariate values `x_new`. The output is a numeric matrix with one row per covariate vector x and one column per quantile level τ.

`predict_density(x_new, y_new)` Returns the estimated conditional density

$$\widehat{\pi}(y \mid x),$$

evaluated at specified (x,y) pairs. The inputs `x_new` and `y_new` are paired row-wise, so that the r-th row of `x_new` is evaluated at `y_new[r]`.

Examples

```
## ------------------------------------------------------------
## Case A: Conditional density evaluated at a single point (x, y)
## ------------------------------------------------------------
## This illustrates the most basic usage: estimating pi(y | x)
## at one covariate value x and one response value y.

dat <- generate_clustered_mar(
  n = 200, m = 4, d = 2,
  target_missing = 0.3, seed = 1
)
fit <- fit_cond_density_quantile(
  dat,
  y_col = "Y", delta_col = "delta",
  x_cols = c("X1", "X2"),
  taus = seq(0.05, 0.95, by = 0.02),
  method = "rq",
  seed = 1
)
## a single covariate value x
x1 <- matrix(c(0.2, -1.0), nrow = 1)
colnames(x1) <- c("X1", "X2")
## estimate pi(y | x) at y = 0.5
fit$predict_density(x1, y_new = 0.5)


## ------------------------------------------------------------
## Case B: Conditional density as a function of y (density curve)
## ------------------------------------------------------------
## Here we fix x and evaluate pi(y | x) over a grid of y values,
## which produces an estimated conditional density curve.

y_grid <- seq(-3, 3, length.out = 201)
## reuse the same x by repeating it to match the y-grid
x_rep <- x1[rep(1, length(y_grid)), , drop = FALSE]
f_grid <- fit$predict_density(x_rep, y_grid)

## ------------------------------------------------------------
## True conditional density under the data generator
## ------------------------------------------------------------
## Data are generated as:
##   Y = X^T beta + b + eps,
##   b ~ N(0, sigma_b^2),  eps ~ N(0, sigma_eps^2)
## Hence the marginal conditional density is:
##   Y | X = x ~ N(x^T beta, sigma_b^2 + sigma_eps^2)

beta_true <- c(0.5, 0.6)
sigma_b_true <- 0.7
sigma_eps_true <- 1.0
mu_true <- drop(x1 %*% beta_true)
sd_true <- sqrt(sigma_b_true^2 + sigma_eps_true^2)
f_true <- stats::dnorm(y_grid, mean = mu_true, sd = sd_true)
```

```
## ------------------------------------------------------------
## Visualization: estimated vs true conditional density
## (use smooth.spline on log-density for a smoother display)
## ------------------------------------------------------------

## smooth the estimated curve for visualization
ok <- is.finite(f_grid) & (f_grid > 0)
sp <- stats::smooth.spline(y_grid[ok], log(f_grid[ok]), spar = 0.85)
f_smooth <- exp(stats::predict(sp, y_grid)$y)

ymax <- max(c(f_smooth, f_true), na.rm = TRUE)
plot(
  y_grid, f_smooth,
  type = "l", lwd = 2,
  xlab = "y",
  ylab = expression(hat(pi)(y ~ "|" ~ x)),
  ylim = c(0, 1.2 * ymax),
  main = "Conditional density at a fixed x: estimated vs true"
)
grid(col = "gray85", lty = 1)
lines(y_grid, f_true, lwd = 2, lty = 2)
legend(
  "topright",
  legend = c("Estimated (smoothed)", "True (generator)"),
  lty = c(1, 2), lwd = c(2, 2), bty = "n"
)
```

fit_missingness_propensity

Fit missingness propensity model P(delta=1 | X) from pooled data

Description

Fits the missingness propensity $\pi(x) = \mathbb{P}(\delta = 1 \mid x)$ under a marginal missingness model using pooled observations. Estimation can be carried out using logistic regression, Generalized Random Forests (GRF), or gradient boosting (xgboost). Both continuous and discrete covariates are supported; categorical variables are automatically expanded into dummy variables via `model.matrix()`.

Usage

```
fit_missingness_propensity(
  dat,
  delta_col = "delta",
  x_cols,
  method = c("logistic", "grf", "boosting"),
  eps = 1e-06,
  ...
)
```

Arguments

`dat`	A `data.frame` containing `delta_col` and `x_cols`. Can be any user-supplied dataset; `generate_clustered_mar()` is used only in examples.
`delta_col`	Name of missingness indicator column (1 observed, 0 missing).
`x_cols`	Character vector of covariate column names used to predict missingness.
`method`	One of `"logistic"`, `"grf"`, `"boosting"`.
`eps`	Clipping level applied to the estimated missingness propensity $\hat{\pi}(x)$, truncating predictions to $[\epsilon, 1-\epsilon]$.
`...`	Extra arguments passed to the learner: `logistic` passed to `stats::glm`. `grf` passed to `grf::probability_forest`. `boosting` passed to `xgboost::xgb.train` via `params=` and `nrounds=`.

Value

A list containing:

`method` The estimation method used.

`fit` The fitted missingness propensity model.

`predict` A function `predict(x_new)` that returns the estimated missingness propensity $\hat{\pi}(x) = \mathbb{P}(\delta = 1 \mid x)$ evaluated at new covariate values `x_new`, with predictions clipped to $[\epsilon, 1-\epsilon]$.

Examples

```
dat <- generate_clustered_mar(
  n = 80, m = 4, d = 2,
  alpha0 = -0.4, alpha = c(-1.0, 0.8),
  target_missing = 0.30,
  seed = 1
)
x_cols <- c("X1", "X2")

## Logistic regression
fit_log <- fit_missingness_propensity(dat, "delta", x_cols, method = "logistic")
p_log <- fit_log$predict(dat[, x_cols, drop = FALSE])
head(p_log)

## Compare with other methods
## True propensity under the generator
s <- attr(dat, "alpha_shift")
eta <- (-0.4) + (-1.0) * dat$X1 + 0.8 * dat$X2
pi_true <- 1 / (1 + exp(-pmin(pmax(eta, -30), 30)))
```

```
fit_grf <- fit_missingness_propensity(
  dat, "delta", x_cols,
  method = "grf", num.trees = 800, num.threads = 1
)
fit_xgb <- fit_missingness_propensity(
  dat, "delta", x_cols,
  method = "boosting",
  nrounds = 300,
  params = list(max_depth = 3, eta = 0.05, subsample = 0.8, colsample_bytree = 0.8),
  nthread = 1
)

p_grf <- fit_grf$predict(dat[, x_cols, drop = FALSE])
p_xgb <- fit_xgb$predict(dat[, x_cols, drop = FALSE])

op <- par(mfrow = c(1, 3))
plot(pi_true, p_log, pch = 16, cex = 0.5,
   xlab = "True pi(x)", ylab = "Estimated pi-hat(x)", main = "Logistic"); abline(0, 1, lwd = 2)
plot(pi_true, p_grf, pch = 16, cex = 0.5,
    xlab = "True pi(x)", ylab = "Estimated pi-hat(x)", main = "GRF"); abline(0, 1, lwd = 2)
plot(pi_true, p_xgb, pch = 16, cex = 0.5,
   xlab = "True pi(x)", ylab = "Estimated pi-hat(x)", main = "Boosting"); abline(0, 1, lwd = 2)
par(op)
```

`generate_clustered_mar` *Simulate clustered continuous outcomes with covariate-dependent MAR missingness*

Description

Simulates clustered data $\{(X_{i,j}, Y_{i,j}, \delta_{i,j})\}$ under a hierarchical subject-level model with covariate-dependent Missing at Random (MAR) missingness: $\delta \perp Y \mid X$. Covariates $X_{i,j}$ are fully observed, while outcomes $Y_{i,j}$ may be missing.

Data are generated according to the following mechanisms:

- **Between-subject level**: subject random intercepts $b_i \sim N(0, \sigma_b^2)$ induce within-cluster dependence, corresponding to latent subject-specific laws P_i.
- **Outcomes**: for each measurement $j = 1, \ldots, m_i$,

  $$Y_{i,j} = X_{i,j}^\top \beta + b_i + \varepsilon_{i,j},$$

  where, for each subject i, the within-cluster errors $\{\varepsilon_{i,j}\}_{j=1}^{m_i}$ are mutually independent with $\varepsilon_{i,j} \sim N(0, \sigma_\varepsilon^2)$ when `rho = 0`. When `rho != 0`, they follow a stationary first-order autoregressive process (AR(1)) within the cluster:

  $$\varepsilon_{i,j} = \rho\, \varepsilon_{i,j-1} + \eta_{i,j}, \quad \eta_{i,j} \sim N\big(0, \sigma_\varepsilon^2(1 - \rho^2)\big),$$

  which implies $\mathrm{Var}(\varepsilon_{i,j}) = \sigma_\varepsilon^2$ and $\mathrm{Cov}(\varepsilon_{i,j}, \varepsilon_{i,j+k}) = \sigma_\varepsilon^2 \rho^{|k|}$ for all k.

- **MAR missingness**: outcomes are observed with probability

$$\Pr(\delta_{i,j} = 1 \mid X_{i,j}) = \text{logit}^{-1}(\alpha_0 + \alpha^\top X_{i,j}),$$

  which depends only on covariates, ensuring $\delta \perp Y \mid X$. If `target_missing` is provided, the intercept α_0 is automatically calibrated (via a deterministic root-finding procedure on the expected missing proportion) so that the *marginal missing proportion* is close to `target_missing`.

Usage

```
generate_clustered_mar(
  n,
  m = 4L,
  d = 2L,
  beta = NULL,
  sigma_b = 0.7,
  sigma_eps = 1,
  rho = 0,
  hetero_gamma = 0,
  x_dist = c("normal", "bernoulli", "uniform"),
  x_params = NULL,
  alpha0 = -0.2,
  alpha = NULL,
  target_missing = NULL,
  seed = NULL
)
```

Arguments

`n`	Number of clusters (subjects).
`m`	Cluster size. Either a single positive integer (common $m_i = m$) or an integer vector of length `n` specifying m_i for each subject.
`d`	Covariate dimension.
`beta`	Population regression coefficients for $Y \mid X$ (length `d`). If `NULL`, defaults to `seq(0.5, 0.5 + 0.1*(d-1), by=0.1)`.
`sigma_b`	SD of subject random intercept b_i.
`sigma_eps`	Marginal SD of within-subject errors $\varepsilon_{i,j}$.
`rho`	AR(1) correlation parameter within cluster for $\varepsilon_{i,j}$.
`hetero_gamma`	Optional heteroskedasticity parameter; a value of 0 yields the standard homoskedastic model, while nonzero values induce covariate-dependent error variance through the first covariate X_1.
`x_dist`	Distribution for covariates: `"normal"`, `"bernoulli"`, or `"uniform"`.
`x_params`	Optional list of distribution parameters for `x_dist`.
`alpha0`	Missingness intercept α_0. If `target_missing` is not `NULL`, the effective intercept becomes $\alpha_0 + s$, where s is a calibrated shift.
`alpha`	Missingness slopes (length `d`). If `NULL`, defaults to zeros.

`target_missing`	Target *marginal missing proportion* defined as the empirical average of the fitted missing probabilities $1 - \pi(X_{i,j})$ over all observations, where $\pi(x) = \Pr(\delta = 1 \mid X = x)$. If `NULL`, no calibration.
`seed`	Optional RNG seed.

Value

A `data.frame` in long format with one row per measurement:

id Cluster index.

j Within-cluster index.

Y Observed outcome; `NA` if missing.

Y_full Latent complete outcome.

delta Observation indicator (1 observed, 0 missing).

X1..Xd Covariates.

Attributes:

`m_i` Integer vector of cluster sizes $(m_1, \ldots, m_n)$.

`target_missing` Target marginal missing proportion used for calibration, defined as the empirical average of missing probabilities over all observations.

`alpha_shift` Calibrated global intercept shift s added to the missingness linear predictor $\alpha_0 + s + \alpha^\top X_{i,j}$ (present only when `target_missing` is provided).

`missing_rate` Sample missing rate $N^{-1} \sum I(\delta_{i,j} = 0)$. This may deviate from `target_missing` due to Bernoulli sampling variability.

Examples

```
dat <- generate_clustered_mar(
  n = 200, m = 5, d = 2,
  alpha0 = -0.2, alpha = c(-1.0, 0.0),
  target_missing = 0.30,
  seed = 1
)
mean(dat$delta == 0)      # ~0.30
attr(dat, "alpha_shift")  # calibrated shift
```

`hcp_conformal_region` *HCP conformal prediction region with repeated subsampling and repeated data splitting*

Description

Constructs a marginal conformal prediction region for a new covariate value x_{n+1} under clustered data with missing outcomes, following the HCP framework:

- **(1) Model fitting.** Fit a pooled conditional density model $\widehat{\pi}(y \mid x)$ using `fit_cond_density_quantile`, together with a marginal missingness propensity model $\widehat{p}(x) = \mathbb{P}(\delta = 1 \mid x)$ using `fit_missingness_propensity`, both estimated on a subject-level training split.
- **(2) Subsampled calibration.** Repeatedly construct calibration sets by randomly drawing one observation per subject from the calibration split.
- **(3) Weighted conformal scoring.** Compute weighted conformal p-values over a candidate grid using the nonconformity score $R(x, y) = -\widehat{\pi}(y \mid x)$ and inverse-propensity weights $w(x) = 1/\widehat{p}(x)$ under a MAR assumption.
- **(4) Aggregation.** Aggregate dependent p-values across subsamples (B) and data splits (S) using either the Cauchy combination test (CCT/ACAT) or the arithmetic mean.

The prediction region is returned as a subset of the supplied grid:

$$\widehat{C}(x_{n+1}; \alpha) = \{y \in \mathcal{Y} : \, p_{\text{final}}(y) > \alpha\}.$$

Usage

```
hcp_conformal_region(
  dat,
  id_col,
  y_col = "Y",
  delta_col = "delta",
  x_cols,
  x_test,
  y_grid,
  alpha = 0.1,
  train_frac = 0.5,
  S = 5,
  B = 5,
  combine_B = c("cct", "mean"),
  combine_S = c("cct", "mean"),
  seed = NULL,
  return_details = FALSE,
  dens_method = c("rq", "qrf"),
  dens_taus = seq(0.05, 0.95, by = 0.02),
  dens_h = NULL,
  enforce_monotone = TRUE,
  tail_decay = TRUE,
  prop_method = c("logistic", "grf", "boosting"),
  prop_eps = 1e-06,
  ...
)
```

Arguments

`dat`	A data.frame containing clustered observations. Must include `id_col`, `y_col`, `delta_col`, and all columns in `x_cols`.
`id_col`	Subject/cluster identifier column name.
`y_col`	Outcome column name.
`delta_col`	Missingness indicator column name (1 observed, 0 missing).
`x_cols`	Covariate column names used for both density estimation and missingness propensity.
`x_test`	New covariate value(s). A numeric vector (treated as one row), or a numeric matrix/data.frame with `nrow(x_test)=K` test points and `ncol(x_test)=length(x_cols)` covariates.
`y_grid`	Numeric vector of candidate y values at which to evaluate conformal p-values.
`alpha`	Miscoverage level in (0,1). Region keeps y with $p(y) > \alpha$.
`train_frac`	Fraction of subjects assigned to training in each split.
`S`	Number of independent subject-level splits.
`B`	Number of subsamples per split (one observation per subject per subsample).
`combine_B`	Combine p-values across B subsamples: `"cct"` (default) or `"mean"`.
`combine_S`	Combine p-values across S splits: `"cct"` (default) or `"mean"`.
`seed`	Optional seed for reproducibility.
`return_details`	Logical; if TRUE, also return split-level p-values and split metadata.
`dens_method`	Density/quantile engine for `fit_cond_density_quantile`: `"rq"` or `"qrf"`.
`dens_taus`	Quantile grid passed to `fit_cond_density_quantile`.
`dens_h`	Bandwidth(s) passed to `fit_cond_density_quantile`.
`enforce_monotone`	Passed to `fit_cond_density_quantile`.
`tail_decay`	Passed to `fit_cond_density_quantile`.
`prop_method`	Missingness propensity method for `fit_missingness_propensity`: `"logistic"`, `"grf"`, or `"boosting"`.
`prop_eps`	Clipping level for propensity predictions used by `fit_missingness_propensity`.
`...`	Extra arguments passed to `fit_missingness_propensity`.

Value

If `return_details=FALSE` (default), a list with:

`region` Length-`K` list; `region[[k]]` is the subset of `y_grid` with `p_final[k, ]` > `alpha`.

`lo_hi` `K` x 2 matrix with columns `c("lo","hi")` giving `min`/`max` of `region[[k]]` (NA if empty).

`p_final` `K` x `length(y_grid)` matrix of final p-values on `y_grid`.

`y_grid` The candidate grid used.

If `return_details=TRUE`, also includes:

`p_split` An array with dimensions `c(S, K, length(y_grid))` of split-level p-values.

`split_meta` Train subject IDs for each split.

Examples

```
dat <- generate_clustered_mar(n = 200, m = 4, d = 2, target_missing = 0.30, seed = 1)
y_grid <- seq(-4, 4, length.out = 200)
x_test <- matrix(c(0.2, -1.0), nrow = 1); colnames(x_test) <- c("X1", "X2")

res <- hcp_conformal_region(
  dat, id_col = "id",
  y_col = "Y", delta_col = "delta",
  x_cols = c("X1", "X2"),
  x_test = x_test,
  y_grid = y_grid,
  alpha = 0.1,
  S = 2, B = 2,
  seed = 1
)

## interval endpoints on the y-grid (outer envelope)
c(lo = min(res$region[[1]]), hi = max(res$region[[1]]))
```

`hcp_predict_targets` *HCP prediction wrapper for multiple measurements with optional per-patient Bonferroni*

Description

Wraps `hcp_conformal_region` to produce conformal prediction regions for a collection of measurements, possibly including multiple measurements per individual.

Based on the structure of the test dataset, the prediction mode is determined automatically as follows, where P denotes the number of patients (clusters) and M denotes the number of measurements per patient:

- $P = 1$, $M = 1$: Predict a single patient with a single measurement.
- $P = 1$, $M > 1$: Predict a single patient with multiple measurements (e.g., repeated or longitudinal measurements for the same patient). If per-patient simultaneous prediction is desired, optional per-patient Bonferroni calibration can be applied.
- $P > 1$, $M = 1$: Predict multiple patients, each with a single measurement. Predictions are performed independently at the nominal level α, without Bonferroni calibration.
- $P > 1$, $M > 1$: Predict multiple patients, each with multiple measurements. When per-patient simultaneous coverage is desired, a Bonferroni correction can be applied by using an effective level α/M_p for each measurement, yielding Bonferroni-adjusted marginal prediction regions for patient p.

Usage

```
hcp_predict_targets(
  dat,
  test,
  pid_col = "pid",
  x_cols,
  y_grid,
  alpha = 0.1,
  bonferroni = FALSE,
  return_region = FALSE,
  id_col = "id",
  y_col = "Y",
  delta_col = "delta",
  ...
)
```

Arguments

`dat`	Training/calibration data passed to `hcp_conformal_region`.
`test`	A data.frame of test measurements, where each row corresponds to a single measurement. The test data must follow one of the four clustered settings $P = 1, M = 1$, $P = 1, M > 1$, $P > 1, M = 1$, or $P > 1, M > 1$, where P is the number of patients (clusters) and M is the number of measurements per patient. The data.frame must include a patient identifier specified by `pid_col` and all covariate columns listed in `x_cols`. Repeated values of `pid_col` indicate multiple measurements (e.g., repeated or longitudinal measurements) for the same patient.
`pid_col`	Column in `test` giving the patient (cluster/subject) identifier. Default `"pid"`.
`x_cols`	Covariate column names (e.g., `c("X1")`).
`y_grid`	Candidate y-grid passed to `hcp_conformal_region`.
`alpha`	Nominal miscoverage level in (0,1) passed to `hcp_conformal_region`.
`bonferroni`	Logical; if TRUE, apply per-patient Bonferroni only when a patient has multiple test measurements (i.e., $M_p > 1$). If FALSE, always use level α.
`return_region`	Logical; if TRUE, return the full region (subset of `y_grid`) for each row.
`id_col, y_col, delta_col`	Column names in `dat` for patient ID, outcome, and missingness indicator.
`...`	Additional arguments forwarded to `hcp_conformal_region` (e.g., `S`, `B`, `combine_B`, `combine_S`, `dens_method`, `prop_method`, `seed`).

Value

A list with:

pred A data.frame in the same row order as `test`. It contains all columns of `test` plus the effective level `alpha_eff` and the prediction-band endpoints `lo` and `hi` for each measurement.

region If `return_region=TRUE`, a list of length `nrow(test)` where each element is the subset of `y_grid` retained in the prediction region for the corresponding test row; otherwise `NULL`.

meta A list with summary information, including the number of patients `P`, the per-patient measurement counts `M_by_pid`, and the settings `alpha` and `bonferroni`.

Note

When per-patient Bonferroni calibration is enabled and a patient has a large number of measurements (e.g., $M_p > 10$), the effective level α/M_p may be very small, which can lead to extremely wide prediction regions (potentially spanning the entire `y_grid`). This behavior is an inherent consequence of Bonferroni adjustment and not a numerical issue.

In longitudinal or panel studies, a cluster corresponds to a single individual (subject), and within-cluster points correspond to multiple time points or repeated measurements on the same individual. In this setting, the time variable `time` can be treated as a generic covariate. In the examples below, time is represented by `X1`.

Examples

```
## ------------------------------------------------------------
## Examples illustrating the four test-data settings:
## (P=1, M=1), (P=1, M>1), (P>1, M=1), and (P>1, M>1)
## ------------------------------------------------------------
set.seed(1)

## training data (fixed across all cases)
dat_train <- generate_clustered_mar(
  n = 200, m = 4, d = 1,
  x_dist = "uniform", x_params = list(min = 0, max = 10),
  target_missing = 0.30,
  seed = 1
)

y_grid <- seq(-6, 6, length.out = 201)

## Case 1: P=1, M=1  (one patient, one measurement)
test_11 <- data.frame(
  pid = 1,
  X1  = 2.5
)
out_11 <- hcp_predict_targets(
  dat = dat_train,
  test = test_11,
  x_cols = "X1",
  y_grid = y_grid,
  alpha = 0.1,
  S = 2, B = 2,
  seed = 1
)
out_11$pred

## Case 2: P=1, M>1  (one patient, multiple measurements)
```

```
test_1M <- data.frame(
  pid = 1,
  X1  = c(1, 3, 7, 9)
)
out_1M <- hcp_predict_targets(
  dat = dat_train,
  test = test_1M,
  x_cols = "X1",
  y_grid = y_grid,
  alpha = 0.1,
  S = 2, B = 2,
  seed = 1
)
out_1M$pred

## Case 3: P>1, M=1  (multiple patients, one measurement each)
test_P1 <- data.frame(
  pid = 1:4,
  X1  = c(2, 4, 6, 8)
)
out_P1 <- hcp_predict_targets(
  dat = dat_train,
  test = test_P1,
  x_cols = "X1",
  y_grid = y_grid,
  alpha = 0.1,
  S = 2, B = 2,
  seed = 1
)
out_P1$pred

## Case 4: P>1, M>1  (multiple patients, multiple measurements per patient)
test_PM <- data.frame(
  pid = c(1,1, 2,2,2, 3,3),
  X1  = c(1,6,  2,5,9,  3,8)
)
out_PM <- hcp_predict_targets(
  dat = dat_train,
  test = test_PM,
  x_cols = "X1",
  y_grid = y_grid,
  alpha = 0.1,
  S = 2, B = 2,
  seed = 1
)
out_PM$pred
```

plot_hcp_intervals *Plot HCP prediction intervals (band vs covariate or intervals by patient)*

Description

Unified plotting function for two common visualizations of HCP prediction intervals:

- `mode="band"`: plot an interval band (lo/hi) versus a 1D covariate (e.g., time X1).
- `mode="pid"`: plot one interval per patient on the x-axis (patients optionally sorted by a covariate).

Usage

```
plot_hcp_intervals(
  df,
  mode = c("band", "pid"),
  lo_col = "lo",
  hi_col = "hi",
  y_true_col = NULL,
  y_true = NULL,
  show_center = TRUE,
  show_true = TRUE,
  x_col = NULL,
  pid_col = "pid",
  x_sort_col = NULL,
  max_patients = NULL,
  ...
)
```

Arguments

`df`	A data.frame containing prediction results. It must include the interval endpoints specified by `lo_col` and `hi_col`, and the covariate columns required by the chosen plotting mode.
`mode`	Plotting mode. Use `"band"` to visualize an interval band as a function of a continuous covariate, or `"pid"` to visualize one prediction interval per patient on the x-axis.
`lo_col`	Name of the column containing the lower endpoint of the prediction interval. Default is `"lo"`.
`hi_col`	Name of the column containing the upper endpoint of the prediction interval. Default is `"hi"`.
`y_true_col`	Optional name of a column in `df` containing the true outcome values. Used for overlaying truth points when `show_true = TRUE`.
`y_true`	Optional numeric vector of true outcome values with length equal to `nrow(df)`. If provided, this overrides `y_true_col`.
`show_center`	Logical; if `TRUE`, draw the midpoint of each interval (as a dashed line in `mode = "band"` or as points in `mode = "pid"`).
`show_true`	Logical; if `TRUE`, overlay true outcome values when available.
`x_col`	(mode = "band") Name of the covariate column used as the x-axis in the interval band plot (e.g., time or another continuous predictor).

`pid_col`	(mode = "pid") Name of the column identifying patients (or clusters). Each patient must appear exactly once in `df`. Default is `"pid"`.
`x_sort_col`	(mode = "pid") Optional covariate column used to order patients along the x-axis (e.g., `"X1"`). If `NULL`, patients are ordered by their IDs.
`max_patients`	(mode = "pid") Optional maximum number of patients to display. If specified, only the first `max_patients` patients after sorting are plotted.
`...`	Additional graphical parameters passed to `plot`, such as `main`, `xlab`, `ylab`, `xlim`, or `ylim`.

Value

Invisibly returns the data.frame used for plotting:

- For `mode = "band"`, the input `df` sorted by `x_col`.
- For `mode = "pid"`, the input `df` sorted by `pid_col` or `x_sort_col`, if provided.

Examples

```
## ------------------------------------------------------------
## Two common plots:
## (A) one patient, multiple measurements  -> interval band vs X1
## (B) multiple patients, one measurement -> intervals by patient (sorted by X1)
## ------------------------------------------------------------
dat_train <- generate_clustered_mar(
  n = 200, m = 20, d = 1,
  x_dist = "uniform", x_params = list(min = 0, max = 10),
  hetero_gamma = 2.5,
  target_missing = 0.30,
  seed = 1
)
y_grid <- seq(-6, 10, length.out = 201)

## test data with latent truth
dat_test <- generate_clustered_mar(
  n = 100, m = 20, d = 1,
  x_dist = "uniform", x_params = list(min = 0, max = 10),
  hetero_gamma = 2.5,
  seed = 999
)

## ---------- Case A: P=1, M>1 (one patient, multiple measurements) ----------
pid <- dat_test$id[1]
idx <- which(dat_test$id == pid)
idx <- idx[order(dat_test$X1[idx])][1:10]
test_1M <- data.frame(pid = pid, X1 = dat_test$X1[idx], y_true = dat_test$Y_full[idx])

out_1M <- hcp_predict_targets(
  dat = dat_train, test = test_1M,
  x_cols = "X1", y_grid = y_grid,
  alpha = 0.1,
  S = 2, B = 2,
  seed = 1
)
plot_hcp_intervals(
  out_1M$pred, mode = "band", x_col = "X1",
  y_true_col = "y_true", show_true = TRUE,
  main = "Case A: one patient, multiple time points (band vs time)"
)

## ---------- Case B: P>1, M=1 (multiple patients, one measurement each) ----------
## take one measurement per patient: j==1 for the first 20 patients
pids <- unique(dat_test$id)[1:20]
test_P1 <- subset(dat_test, id %in% pids & j == 1,
                  select = c(id, X1, Y_full))
names(test_P1) <- c("pid", "X1", "y_true")

out_P1 <- hcp_predict_targets(
  dat = dat_train, test = test_P1,
  x_cols = "X1", y_grid = y_grid,
  alpha = 0.1,
  S = 2, B = 2,
  seed = 1
)
plot_hcp_intervals(
  out_P1$pred, mode = "pid", pid_col = "pid", x_sort_col = "X1",
  y_true_col = "y_true", show_true = TRUE,
  main = "Case B: multiple patients, one time point (by patient)"
)
```

Index