# Package: MetaHunt (via r-universe)

May 12, 2026

**Title** Privacy-Preserving Meta-Analysis via Low-Rank Basis Hunting

**Version** 0.1.0

**Description** Tools for privacy-preserving meta-analysis of function-valued quantities across heterogeneous studies. Implements the 'MetaHunt' pipeline, including the denoised functional Successive Projection Algorithm (d-fSPA) for basis hunting, constrained weight estimation, Dirichlet regression of weights on study-level covariates, target prediction, and split/cross conformal prediction intervals. Operates on aggregate-level function evaluations, so individual-level data from source studies are not required. Methodology described in Shi, Imai, and Zhang (2026) <doi:10.48550/arXiv.2604.23847>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Depends** R (>= 4.0)

**Imports** quadprog, DirichletReg, stats, graphics, grDevices, withr

**Suggests** grf, knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**URL** https://github.com/WShi18/MetaHunt, https://wshi18.github.io/MetaHunt/, https://arxiv.org/abs/2604.23847

**BugReports** https://github.com/WShi18/MetaHunt/issues

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Wenqi Shi [aut, cre], Kosuke Imai [aut], Yi Zhang [aut]

**Maintainer** Wenqi Shi <wenqishi18@gmail.com>

**Repository** https://cran.r-universe.dev

**Date/Publication** 2026-05-12 21:44:49 UTC

**RemoteUrl** https://github.com/cran/MetaHunt

**RemoteRef** HEAD

**RemoteSha** d0938ee12b9da307dfc623036c9284a7285cc2eb

# Contents



---

`apply_wrapper` *Reduce predicted functions to scalars via a user-supplied wrapper*

---

## Description

Many downstream quantities of interest (average treatment effect, pointwise predictions, or other functionals of $f^{(0)}$) are scalar summaries of the predicted function. `apply_wrapper()` applies any user-supplied reduction to each row of a function matrix, with a default of the weighted mean with respect to `grid_weights` (which coincides with $\int f \, d\mu$ when `grid_weights` represents $\mu$).

## Usage

```
apply_wrapper(F_mat, wrapper = NULL, grid_weights = NULL)
```

### Arguments

| | |
|---|---|
| F_mat | An n-by-G_grid numeric matrix; each row a function on a shared grid (e.g. the output of predict_target()). |
| wrapper | Either NULL (default, weighted mean) or a function that takes a single numeric vector of length G_grid and returns a scalar. Examples: mean, median, max, function(f) f[17] (point evaluation), function(f) sum(f^2). |
| grid_weights | Optional length-G_grid non-negative numeric vector. Used only when wrapper = NULL. Defaults to uniform 1 / G_grid. |

### Value

A length-n numeric vector of scalar summaries.

### Examples

```
F_mat <- matrix(1:12, nrow = 3, byrow = TRUE)  # 3 "functions" on a 4-point grid
apply_wrapper(F_mat)                            # row means (uniform grid weights)
apply_wrapper(F_mat, wrapper = max)             # row maxes
apply_wrapper(F_mat, wrapper = function(f) f[2]) # point evaluation at grid idx 2
```

---

## build_grid *Build a shared evaluation grid from a reference dataset*

---

### Description

Constructs a data frame of grid points suitable for f_hat_from_models() from any reference patient-level dataset. This is convenient when the patient-level covariate space is multidimensional and there is no obvious one-dimensional grid.

### Usage

```
build_grid(reference_data, n_grid = NULL, seed = NULL)
```

### Arguments

| | |
|---|---|
| reference_data | A data frame (or matrix) of reference patient-level covariates. May be a held-out target-population sample, the pooled source covariates, or any plausible reference distribution. |
| n_grid | Optional integer giving the desired grid size. If NULL or >= nrow(reference_data), the full reference data is returned. |
| seed | Optional integer seed for reproducibility (used only when sub-sampling). |

## Details

If `n_grid` is `NULL` or at least `nrow(reference_data)`, the reference data is returned unchanged. Otherwise `n_grid` rows are sampled uniformly at random (without replacement). The reference data should be on the same scale and have the same columns as the data each centre's model was fitted on.

The empirical distribution of the returned grid implicitly defines the $\mu$ measure used downstream. Pass uniform `grid_weights` (the default) to weight each grid point equally; pass non-uniform `grid_weights` to weight by an external reference distribution.

## Value

A data frame of grid points.

## Examples

```
set.seed(1)
ref <- data.frame(age = rnorm(500, 60, 10),
                  bp  = rnorm(500, 130, 15),
                  sex = sample(c("F", "M"), 500, replace = TRUE))
grid <- build_grid(ref, n_grid = 50, seed = 1)
nrow(grid)
head(grid)
```

---

`coef.metahunt_weight_model` *Extract coefficients from a MetaHunt weight model*

---

## Description

Returns the regression coefficients from the underlying weight-model fit. For the default `"dirichlet"` method this delegates to `DirichletReg::DirichReg()`'s `coef()` method.

## Usage

```
## S3 method for class 'metahunt_weight_model'
coef(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | A fitted `"metahunt_weight_model"` from `fit_weight_model()`. |
| `...` | Passed through to `stats::coef()`. |

## Value

The coefficient vector / matrix returned by `DirichletReg::DirichReg()`'s `coef()` method (numeric vector or matrix depending on the parametrisation used by the underlying fit).

## Examples

```
set.seed(1)
m <- 60; K <- 3
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- cbind(0.5 * W$w1, -0.3 * W$w2, rep(0, m))
pi_true <- exp(eta) / rowSums(exp(eta))
pi_hat <- pi_true + matrix(rnorm(m * K, sd = 0.01), m, K)
pi_hat <- pmax(pi_hat, 0); pi_hat <- pi_hat / rowSums(pi_hat)
model <- fit_weight_model(pi_hat, W)
coef(model)
```

---

# conformal_from_fit *Split conformal intervals from a pre-fit MetaHunt pipeline*

---

## Description

Lower-level entry point that builds split conformal intervals for new target covariates using an already-fitted d-fSPA basis decomposition, an already-fitted weight model, and a user-supplied calibration set. Use this when you have independently tuned `K` or want to reuse a pipeline fit; otherwise the high-level `split_conformal()` is usually more convenient.

## Usage

```
conformal_from_fit(
  dfspa_fit,
  weight_model,
  F_cal,
  W_cal,
  W_new,
  alpha = 0.05,
  wrapper = NULL,
  grid_weights = NULL
)
```

## Arguments

| | |
|---|---|
| `dfspa_fit` | A `"dfspa"` object from `dfspa()`. |
| `weight_model` | A `"metahunt_weight_model"` object from `fit_weight_model()`. Must satisfy `weight_model$K == nrow(dfspa_fit$bases)`. |
| `F_cal` | An `n_cal`-by-`G_grid` numeric matrix of observed study-level function evaluations for the calibration set. Calibration studies must be independent of both `dfspa_fit` and `weight_model` for valid coverage. |
| `W_cal` | A matrix or data frame of study-level covariates for the calibration set, with the same columns used to fit `weight_model`. |

| | |
|---|---|
| W_new | A matrix or data frame of study-level covariates for new target studies. |
| alpha | Miscoverage level; default 0.05. |
| wrapper | Optional reduction function (see apply_wrapper()). If NULL, intervals are pointwise at every grid point. |
| grid_weights | Optional length-G_grid non-negative numeric vector used for the default wrapper and for weighted inner products. |

## Value

An object of class "metahunt_conformal"; see split_conformal() for a description of its fields.

## See Also

split_conformal() for the high-level version that splits and fits internally, cross_conformal() for the K-fold variant.

## Examples

```
set.seed(1)
G <- 30; m <- 80
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m))
eta <- cbind(0.8 * W$w1, -0.4 * W$w1, rep(0, m))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

# user-controlled split and fit
tr  <- 1:50; cal <- 51:70; new <- 71:80
fit <- dfspa(F_hat[tr, ], K = 3)
pih <- project_to_simplex(F_hat[tr, ], fit$bases)
wm  <- fit_weight_model(pih, W[tr, , drop = FALSE])

res <- conformal_from_fit(
  fit, wm,
  F_cal = F_hat[cal, ], W_cal = W[cal, , drop = FALSE],
  W_new = W[new, , drop = FALSE],
  wrapper = mean
)
res
```

---

coverage *Empirical coverage of a conformal prediction-interval object*

---

## Description

Computes empirical coverage indicators of a fitted "metahunt_conformal" object against held-out observed study-level functions. The held-out studies must correspond positionally to the targets used to build object (i.e. F_obs[i, ] is the observed function for the same target whose prediction is in object$prediction[i, ] or object$prediction[i]).

## Usage

```
coverage(object, F_obs, grid_weights = NULL)
```

## Arguments

| | |
|---|---|
| object | A "metahunt_conformal" object from split_conformal(), cross_conformal(), or conformal_from_fit(). |
| F_obs | An n_target-by-G_grid numeric matrix of observed study-level functions for the target studies, in the same row order as object$prediction. |
| grid_weights | Optional length-G_grid non-negative numeric vector. Only used in scalar mode (passed to apply_wrapper()). |

## Details

In **pointwise** mode each entry (i, g) of F_obs is compared against [object$lower[i, g], object$upper[i, g]]. In **scalar** mode F_obs is first reduced to a length-n_target vector via apply_wrapper() using object$wrapper and the supplied grid_weights, and then compared against [object$lower, object$upper].

Coverage is a finite-sample diagnostic. Nominal coverage is 1 - object$alpha; empirical coverage will fluctuate around this value due to sampling.

## Value

A list. In **pointwise** mode the list contains

pointwise n_target-by-G_grid logical matrix of coverage indicators.

per_target Length-n_target numeric vector of mean coverage across the grid for each target.

per_grid_point Length-G_grid numeric vector of mean coverage across targets at each grid point.

overall Scalar mean coverage across all entries.

nominal Nominal coverage 1 - object$alpha.

In **scalar** mode the list contains

pointwise Length-n_target logical vector of coverage indicators.

overall Scalar mean coverage.

nominal Nominal coverage 1 - object$alpha.

## Examples

```
set.seed(1)
G <- 25; m <- 80
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m))
eta <- cbind(0.6 * W$w1, -0.3 * W$w1, rep(0, m))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

# held-out test set: same data-generating process, same W
test_idx <- 1:10
train_idx <- setdiff(seq_len(m), test_idx)
res <- split_conformal(F_hat[train_idx, ], W[train_idx, , drop = FALSE],
                       W[test_idx, , drop = FALSE], K = 3,
                       dfspa_args = list(denoise = FALSE), seed = 1)
cov <- coverage(res, F_obs = F_hat[test_idx, , drop = FALSE])
cov$overall
```

---

`cross_conformal` *Cross-conformal prediction intervals (pooled K-fold scores)*

---

## Description

Computes K-fold split conformal intervals in which the calibration scores are pooled across folds, while point predictions for new targets are produced from a final pipeline fit on all studies. Equivalent to running `split_conformal()` `n_folds` times with different calibration sets and pooling all conformity scores into a single empirical distribution.

## Usage

```
cross_conformal(
  F_hat,
  W,
  W_new,
  K,
  alpha = 0.05,
  n_folds = 5L,
  wrapper = NULL,
  grid_weights = NULL,
  dfspa_args = list(),
  weight_model_args = list(),
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| F_hat | An m-by-G_grid numeric matrix of study-level function evaluations. |
| W | An m-by-p matrix or data frame of study-level covariates. |
| W_new | A matrix or data frame of new target covariates. Must contain columns matching W. |
| K | Integer number of basis functions. |
| alpha | Miscoverage level; interval has nominal coverage $1 - \alpha$. Default 0.05. |
| n_folds | Integer number of folds (>= 2). Default 5. |
| wrapper | Optional reduction function (see apply_wrapper()). If NULL, intervals are constructed pointwise at every grid point. |
| grid_weights | Optional length-G_grid non-negative numeric vector used for the $L^2(\mu)$ norm and for the default wrapper. |
| dfspa_args, weight_model_args | Named lists passed to dfspa() and fit_weight_model() respectively. |
| seed | Optional integer seed for reproducible train/calibration splits. |

## Details

For each fold, the MetaHunt pipeline is fit on the out-of-fold studies, and conformity scores are computed on the in-fold studies. After all folds complete, the pooled scores yield a single $(1 - \alpha)$-quantile (or one per grid point). Point predictions for W_new use a pipeline refit on the full dataset. The interval at grid point g for target j is $[\tilde{f}^{(j)}(x_g) - q_g, \tilde{f}^{(j)}(x_g) + q_g]$ (or the scalar analogue when wrapper is supplied).

This differs from Vovk's original cross-conformal predictor for classification. For regression, pooling scores across folds is a common practical extension of split conformal and reduces the variance due to the single calibration split. Exact finite-sample coverage is not guaranteed; see Barber et al. (2021, Jackknife+) for more conservative alternatives.

## Value

An object of class "metahunt_conformal" (see split_conformal() for fields). method is "cross" and n_cal is the number of pooled scores.

## Examples

```
set.seed(1)
G <- 30; m <- 60; K_true <- 3
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m))
eta <- cbind(0.8 * W$w1, -0.3 * W$w1, rep(0, m))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)
W_new <- data.frame(w1 = c(0, 1))

res <- cross_conformal(F_hat, W, W_new, K = 3, n_folds = 4,
                       dfspa_args = list(denoise = FALSE), seed = 1)
```

```
res
```

---

`cv_error_curve` *Cross-validated prediction-error curve for basis-rank selection*

---

## Description

For each candidate `K`, perform k-fold cross-validation at the study level. Within each fold, the full MetaHunt pipeline (d-fSPA + constrained projection + weight model) is refit on the training studies, and the held-out studies' functions are predicted from their study-level covariates. The prediction error is $\|\hat{f}^{(i)} - \tilde{f}^{(i)}\|_{L^2(\mu)}$ averaged over held-out studies and then over folds.

## Usage

```
cv_error_curve(
  F_hat,
  W,
  K_range = NULL,
  n_folds = 5L,
  grid_weights = NULL,
  dfspa_args = list(),
  weight_model_args = list(),
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| `F_hat` | An `m`-by-`G_grid` numeric matrix of study-level function evaluations. |
| `W` | An `m`-by-`p` matrix or data frame of study-level covariates. |
| `K_range` | Integer vector of candidate `K` values. Defaults to `2:min(nrow(F_hat) - 1, 10)`. |
| `n_folds` | Integer number of CV folds (default `5`). |
| `grid_weights` | Optional length-`G_grid` non-negative numeric vector. |
| `dfspa_args` | Named list of extra arguments for `dfspa()`. |
| `weight_model_args` | Named list of extra arguments for `fit_weight_model()`. |
| `seed` | Optional integer seed for reproducible fold assignment; if `NULL` no seeding is performed. |

## Details

This is the supervised rank-selection criterion of Section 3.2 of the paper. Each held-out study is excluded from both basis hunting and weight-model fitting.

## Value

A data frame with columns `K`, `cv_error` (mean over folds), and `cv_se` (standard error across folds). The per-fold error matrix is attached as the attribute `"fold_errors"` (`length(K_range)`-by-`n_folds`). Folds where the pipeline fails contribute `NA` and are summarised in a single warning.

## Examples

```
set.seed(1)
G <- 40; m <- 80; K_true <- 3
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- as.matrix(W) %*% cbind(c(1, -0.8), c(-0.5, 1.2), c(0, 0))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.02), m, G)
cv <- cv_error_curve(F_hat, W, K_range = 2:5, n_folds = 4, seed = 1)
cv
```

---

# dfspa *Denoised functional Successive Projection Algorithm (d-fSPA)*

---

## Description

Recovers a set of `K` latent basis functions from a collection of study-level function estimates under the low-rank cross-study heterogeneity assumption of Shi, Imai, and Zhang. Implements Algorithm 1 of the paper ("The d-fSPA Algorithm for basis hunting").

## Usage

```
dfspa(F_hat, K, grid_weights = NULL, N = NULL, Delta = NULL, denoise = TRUE)
```

## Arguments

| | |
|---|---|
| `F_hat` | An `m`-by-`G` numeric matrix where row `i` is the evaluation of the estimated function $\hat{f}^{(i)}$ at `G` grid points. |
| `K` | Integer number of basis functions to recover. Must satisfy `1 <= K <= m` after denoising. |
| `grid_weights` | Optional length-`G` non-negative numeric vector of grid weights defining the $L^2(\mu)$ inner product. Defaults to uniform weights `1 / G`. |
| `N`, `Delta` | Optional numeric tuning parameters controlling denoising. See Details. |
| `denoise` | Logical; if `FALSE`, the denoising step is skipped and plain fSPA is run. Defaults to `TRUE`. |

## Details

Each study-level function is represented by its evaluations on a shared grid of `G` points. The (weighted) $L^2(\mu)$ inner product is $\langle f, g\rangle = \sum_{j=1}^{G} w_j f(x_j) g(x_j)$, where the `grid_weights` `w_j` are proportional to the measure $\mu$. If not supplied, uniform weights `1 / G` are used.

Denoising follows Jin (2024): for each study `i`, let $B_\Delta(\hat{f}^{(i)}) = \{j : \|\hat{f}^{(j)} - \hat{f}^{(i)}\| \le \Delta\}$. If $|B_\Delta(\hat{f}^{(i)})| < N$, study `i` is discarded; otherwise $\hat{f}^{(i)}$ is replaced by the average of the functions in $B_\Delta$. After denoising, the functional SPA step iteratively selects, at each of the `K` iterations, the remaining function with the largest norm after projecting out the span of previously selected bases.

Default tuning parameters follow the heuristics of the paper: `N = 0.5 * log(m)` and $\Delta = \max_{ij} \|\hat{f}^{(i)} - \hat{f}^{(j)}\|/10$.

## Value

An object of class `"dfspa"`: a list containing

`bases` A `K`-by-`G` matrix whose rows are the recovered basis functions evaluated on the grid (denoised, if applicable).

`selected` Length-`K` integer vector of the selected row indices into the post-denoising function matrix.

`original_indices` Length-`K` integer vector of the selected study indices in the original input `F_hat` (before any rows were dropped by denoising).

`kept` Integer vector of row indices of `F_hat` that survived denoising.

`F_denoised` The post-denoising function matrix (`length(kept)`-by-`G`).

`grid_weights` Grid weights used.

`N`, `Delta` Tuning parameters actually used (or `NA` when `denoise = FALSE`).

`K` Number of bases requested.

`call` The matched call.

## Examples

```
set.seed(1)
G <- 50
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)          # 3 true bases
pi_mat <- rbind(diag(3),                              # 3 pure studies
                c(0.5, 0.3, 0.2),
                c(0.2, 0.5, 0.3),
                c(0.3, 0.3, 0.4))
F_hat <- pi_mat %*% basis                             # m = 6, G = 50
fit <- dfspa(F_hat, K = 3, denoise = FALSE)
fit$original_indices    # should be a permutation of 1, 2, 3
```

f_hat_from_models *Build the* F_hat *matrix from a list of fitted study-level models*

## Description

Many users arrive at MetaHunt with one fitted model per study (e.g. a `ranger::ranger()` random forest or a `grf::causal_forest()`) and a chosen evaluation grid. `f_hat_from_models()` evaluates each model on the shared grid and stacks the predictions into the `m`-by-`G_grid` matrix the rest of the package expects.

## Usage

```
f_hat_from_models(models, grid, predict_fn = NULL)
```

## Arguments

| | |
|---|---|
| `models` | A non-empty list of fitted model objects, one per study. |
| `grid` | A data frame (or matrix) of grid points at which to evaluate each model. Same columns as the data each model was fitted on. |
| `predict_fn` | Optional `function(model, grid)` returning a numeric vector; defaults to the class-aware dispatcher above. |

## Details

By default the function dispatches on the model class:

- `ranger` objects are evaluated as `predict(model, data = grid)$predictions`.
- Objects inheriting from `causal_forest` or `grf` are evaluated as `predict(model, newdata = grid)$predictions`.
- All other classes fall through to `as.numeric(predict(model, newdata = grid))`, which works for `lm`, `glm`, `randomForest`, and most other R model objects.

Override the dispatch with `predict_fn = function(model, grid) ...` if your models need bespoke handling. The function must return a length-`G_grid` numeric vector for each model.

All rows of the returned matrix must have the same length (`G_grid`) and contain no `NA` values; the function errors otherwise.

## Value

An `length(models)`-by-`nrow(grid)` numeric matrix; row `i` is model `i` evaluated at every row of `grid`.

## See Also

`build_grid()` to construct `grid` from a reference dataset.

## Examples

```
# Toy example: each "centre" fits a polynomial regression
set.seed(1)
make_centre_data <- function(slope) {
  x <- runif(60)
  data.frame(x = x, y = slope * x + rnorm(60, sd = 0.1))
}
models <- lapply(c(-1, 0, 1, 0.5, -0.5), function(s)
  stats::lm(y ~ poly(x, 2), data = make_centre_data(s)))

grid  <- data.frame(x = seq(0, 1, length.out = 30))
F_hat <- f_hat_from_models(models, grid)
dim(F_hat)   # 5 x 30
```

---

`fit_weight_model` *Fit a weight model mapping study-level covariates to simplex weights*

---

## Description

Given a matrix of simplex-valued weights $\hat{\pi}_1, \ldots, \hat{\pi}_m$ (e.g. from `project_to_simplex()`) and associated study-level covariates $\mathbf{W}_1, \ldots, \mathbf{W}_m$, fit a model $\widehat{\mathcal{M}} : \mathbf{W} \mapsto \boldsymbol{\pi}$. The default method is Dirichlet regression via the `DirichletReg` package.

## Usage

```
fit_weight_model(
  pi_hat,
  W,
  method = c("dirichlet"),
  boundary_eps = 1e-04,
  formula = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `pi_hat` | An `m`-by-`K` numeric matrix of simplex weights; rows must be non-negative and sum to 1 (up to tolerance `1e-6`). |
| `W` | An `m`-by-`p` matrix or data frame of study-level covariates. |
| `method` | Weight-model method. Currently only `"dirichlet"` is supported. |
| `boundary_eps` | Small positive scalar used to shrink weights away from the simplex boundary before Dirichlet fitting. Defaults to `1e-4`. |
| `formula` | Optional RHS-only formula (e.g. `~ x1 + I(x2^2)`) describing the covariate part of the Dirichlet regression. Defaults to `~ .` (all columns of `W`). |
| `...` | Passed through to `DirichletReg::DirichReg()`. |

## Details

Dirichlet regression cannot handle weights exactly at the simplex boundary (`0` or `1`), which frequently arise after constrained projection. Before fitting, rows of `pi_hat` are shrunk toward the barycenter via $\tilde{\pi} = (\pi + \varepsilon)/(1 + K\varepsilon)$, with $\varepsilon$ set by `boundary_eps`.

## Value

An object of class `"metahunt_weight_model"`: a list with the fitted model, formula, method, `K`, and training covariate names.

## Examples

```
set.seed(1)
m <- 80; K <- 3; p <- 2
W <- matrix(rnorm(m * p), m, p); colnames(W) <- c("w1", "w2")
# generate simplex weights driven by W
eta <- cbind(0.5 * W[, 1], -0.3 * W[, 2], rep(0, m))
pi_true <- exp(eta) / rowSums(exp(eta))
pi_hat <- pi_true + matrix(rnorm(m * K, sd = 0.01), m, K)
pi_hat <- pmax(pi_hat, 0); pi_hat <- pi_hat / rowSums(pi_hat)
model <- fit_weight_model(pi_hat, W)
predict(model, newdata = matrix(c(0, 0), 1, 2, dimnames = list(NULL, c("w1","w2"))))
```

---

# metahunt *Fit the full MetaHunt pipeline*

---

## Description

End-to-end convenience wrapper that runs the three training-time steps of the MetaHunt pipeline in sequence: (1) `dfspa()` for basis hunting, (2) `project_to_simplex()` for per-study weight recovery, and (3) `fit_weight_model()` for modelling the weight-to-covariate map. The result supports `predict.metahunt()` for generating target-function predictions on new study-level covariates.

## Usage

```
metahunt(
  F_hat,
  W,
  K,
  grid_weights = NULL,
  dfspa_args = list(),
  weight_model_args = list()
)
```

## Arguments

| | |
|---|---|
| F_hat | An m-by-G_grid numeric matrix of study-level function evaluations on a shared grid; row i is $\hat{f}^{(i)}$. |
| W | An m-by-p matrix or data frame of study-level covariates. |
| K | Integer number of basis functions. |
| grid_weights | Optional length-G_grid non-negative numeric vector defining the $L^2(\mu)$ inner product; defaults to uniform. |
| dfspa_args | Named list of extra arguments for dfspa(). |
| weight_model_args | Named list of extra arguments for fit_weight_model(). |

## Details

For uncertainty quantification, pair a "metahunt" fit with conformal_from_fit() (requires a separate calibration set). The high-level split_conformal() and cross_conformal() functions perform their own fitting and do not consume a pre-fit "metahunt" object.

## Value

An object of class "metahunt": a list with the dfspa_fit, weight_model, training pi_hat, K, and a stored copy of grid_weights.

## See Also

predict.metahunt(), split_conformal(), cross_conformal(), conformal_from_fit(), reconstruction_error_cur
cv_error_curve().

## Examples

```
set.seed(1)
G <- 40; m <- 80
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- as.matrix(W) %*% cbind(c(1, -0.8), c(-0.5, 1.2), c(0, 0))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

fit <- metahunt(F_hat, W, K = 3)
fit
f_pred <- predict(fit, newdata = W[1:3, ])
dim(f_pred)                                    # 3 x G: predicted functions
predict(fit, newdata = W[1:3, ], wrapper = mean)  # scalar summaries
```

`minmax_regret` *Minimax-regret aggregator for multisite function-valued estimands*

## Description

Implements the minimax-regret estimator of Zhang, Huang, and Imai (arXiv:2412.11136) for aggregating site-level function estimates. Given study-level functions $\hat{f}^{(i)}$, the estimator is

$$\hat{q} = \arg\min_{q \in \Delta_{m-1}} \; q^\top \hat{\Gamma}\, q - \hat{d}^\top q, \qquad \hat{\Gamma}_{ij} = \sum_g w_g\, \hat{f}^{(i)}(x_g)\hat{f}^{(j)}(x_g), \;\; \hat{d}_i = \sum_g w_g\, (\hat{f}^{(i)}(x_g))^2,$$

yielding the predicted target function $\tilde{f}(x) = \sum_{i=1}^m \hat{q}_i\, \hat{f}^{(i)}(x)$. Unlike `metahunt()`, this method does not use study-level covariates; the target is the worst-case-regret aggregator over the convex hull of source functions.

## Usage

```
minmax_regret(F_hat, grid_weights = NULL, ridge = 1e-10, wrapper = NULL)
```

## Arguments

| | |
|---|---|
| `F_hat` | An `m`-by-`G_grid` numeric matrix of source-site function evaluations on a shared grid; row `i` is $\hat{f}^{(i)}$. |
| `grid_weights` | Optional length-`G_grid` non-negative numeric vector defining the target measure used to compute $\hat{\Gamma}$ and $\hat{d}$. Defaults to uniform weights `1 / G_grid`. |
| `ridge` | Non-negative scalar; replaces $\hat{\Gamma}$ with $\hat{\Gamma}+\text{ridge}\cdot I$ for numerical stability. Defaults to `1e-10`. |
| `wrapper` | Optional reduction function (see `apply_wrapper()`). If `NULL`, `prediction` is the length-`G_grid` predicted target function; if a function, `prediction` is the scalar `wrapper(prediction)` (e.g. an ATE when `wrapper = mean` and rows of `F_hat` are CATE evaluations). |

## Details

The simplex-constrained QP is solved with `quadprog::solve.QP()`. A small ridge is added to $\hat{\Gamma}$ for numerical stability when source functions are highly collinear. The resulting `q` is clipped to be non-negative and renormalised to sum to 1 to absorb floating-point drift.

## Value

An object of class `"minmax_regret"`: a list with

`prediction` Predicted target. Length-`G_grid` vector when `wrapper = NULL`; scalar otherwise.

`q` Length-`m` simplex weights from the minimax-regret QP.

`Gamma` The `m`-by-`m` Gram matrix used (post-ridge).

`d` Length-`m` linear coefficient vector.

`grid_weights` Grid weights used.

`ridge` Ridge value used.

`wrapper` Wrapper function or `NULL`.

## References

Zhang, Y., Huang, M., and Imai, K. (2024). Minimax regret estimation for generalizing heterogeneous treatment effects with multisite data. arXiv:2412.11136.

## Examples

```
set.seed(1)
G <- 30; m <- 6
x <- seq(0, 1, length.out = G)
F_hat <- rbind(
  sin(pi * x),
  cos(pi * x),
  x,
  0.5 * sin(pi * x) + 0.5 * x,
  0.3 * cos(pi * x) + 0.7 * x,
  0.4 * sin(pi * x) + 0.4 * cos(pi * x) + 0.2 * x
)

fit <- minmax_regret(F_hat)
fit$q                                    # simplex weights over sources
length(fit$prediction)                   # G: predicted target function

# ATE-style scalar via wrapper
minmax_regret(F_hat, wrapper = mean)$prediction
```

---

`plot.metahunt` *Plot recovered basis functions from a MetaHunt fit*

---

## Description

Plot recovered basis functions from a MetaHunt fit

## Usage

```
## S3 method for class 'metahunt'
plot(x, x_axis = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `"metahunt"` object from `metahunt()`. |
| `x_axis` | Optional numeric vector of length `G_grid` giving the x-axis values. Defaults to `seq_len(G_grid)`. |
| `...` | Passed to `graphics::matplot()`. |

## Value

Invisibly returns `x`.

## Examples

```
set.seed(1)
G <- 25; m <- 40
x_grid <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x_grid), cos(pi * x_grid), x_grid)
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- as.matrix(W) %*% cbind(c(1, -0.8), c(-0.5, 1.2), c(0, 0))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

fit <- metahunt(F_hat, W, K = 3, dfspa_args = list(denoise = FALSE))
plot(fit)
plot(fit, x_axis = x_grid)
```

---

# plot.metahunt_conformal

*Plot a conformal prediction-interval object*

---

## Description

For pointwise objects (no `wrapper` was used), draws the predicted function for one target study together with its pointwise band. For scalar objects, draws point predictions with whisker error bars over all targets.

## Usage

```
## S3 method for class 'metahunt_conformal'
plot(
  x,
  target_idx = 1L,
  x_axis = NULL,
  fill = grDevices::adjustcolor("steelblue", 0.2),
  line_col = "steelblue",
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | A `"metahunt_conformal"` object. |
| `target_idx` | For pointwise objects, the integer index of the target study to plot (default `1`). Ignored for scalar objects. |
| `x_axis` | Optional numeric vector of length `G_grid` giving the x-axis values for pointwise plotting. Defaults to `seq_len(G_grid)`. |

| | |
|---|---|
| fill | Polygon fill colour for the band. Default semi-transparent steel blue. |
| line_col | Line colour for the predicted function (or points in scalar mode). Default steel blue. |
| ... | Additional graphical parameters passed to the underlying plotting calls. |

## Value

Invisibly returns x.

---

predict.metahunt *Predict target functions (or scalar summaries) from a MetaHunt fit*

---

## Description

Predict target functions (or scalar summaries) from a MetaHunt fit

## Usage

```
## S3 method for class 'metahunt'
predict(object, newdata, wrapper = NULL, grid_weights = NULL, ...)
```

## Arguments

| | |
|---|---|
| object | A "metahunt" object from metahunt(). |
| newdata | A matrix or data frame of new study-level covariates. |
| wrapper | Optional reduction function. If NULL, returns the full predicted function matrix (nrow(newdata)-by-G_grid). If a function, applied to each predicted function to return a scalar per new target; see apply_wrapper(). |
| grid_weights | Optional length-G_grid non-negative numeric vector used only when wrapper = NULL with the default weighted-mean path. Defaults to the grid_weights stored in object. |
| ... | Ignored. |

## Value

Either an nrow(newdata)-by-G_grid matrix of predicted functions (when wrapper = NULL) or a length-nrow(newdata) numeric vector of scalar summaries.

## Examples

```
set.seed(1)
G <- 25; m <- 40
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- as.matrix(W) %*% cbind(c(1, -0.8), c(-0.5, 1.2), c(0, 0))
pi_true <- exp(eta) / rowSums(exp(eta))
```

```
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

fit <- metahunt(F_hat, W, K = 3, dfspa_args = list(denoise = FALSE))
f_pred <- predict(fit, newdata = W[1:3, ])
dim(f_pred)                                        # 3 x G
predict(fit, newdata = W[1:3, ], wrapper = mean)  # scalar summaries
```

---

`predict.metahunt_weight_model` *Predict simplex weights for new study-level covariates*

---

## Description

Predict simplex weights for new study-level covariates

## Usage

```
## S3 method for class 'metahunt_weight_model'
predict(object, newdata, ...)
```

## Arguments

| | |
|---|---|
| `object` | A fitted `"metahunt_weight_model"` from `fit_weight_model()`. |
| `newdata` | A matrix or data frame of new study-level covariates with the same columns used at fitting. |
| `...` | Ignored. |

## Value

An `nrow(newdata)`-by-`K` numeric matrix of predicted simplex weights (component means); rows sum to 1.

## Examples

```
set.seed(1)
m <- 40; K <- 3
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- cbind(0.5 * W$w1, -0.3 * W$w2, rep(0, m))
pi_true <- exp(eta) / rowSums(exp(eta))
pi_hat <- pi_true + matrix(rnorm(m * K, sd = 0.01), m, K)
pi_hat <- pmax(pi_hat, 0); pi_hat <- pi_hat / rowSums(pi_hat)
model <- fit_weight_model(pi_hat, W)
predict(model, newdata = data.frame(w1 = c(0, 1), w2 = c(0, -1)))
```

predict_target *Predict the target function for new study-level covariates*

## Description

Given a fitted d-fSPA basis decomposition and a fitted weight model, compute the predicted target function on the shared grid as

$$\tilde{f}^{(0)}(\cdot) = \sum_{k=1}^{\hat{K}} \tilde{\pi}_{0k}\, \hat{g}_k(\cdot), \qquad \tilde{\pi}_0 = \widehat{\mathcal{M}}(\mathbf{W}_0).$$

## Usage

```
predict_target(dfspa_fit, weight_model, W_new)
```

## Arguments

| | |
|---|---|
| dfspa_fit | A "dfspa" object produced by dfspa(). Its bases slot (K-by-G_grid matrix) provides the basis functions on the grid. |
| weight_model | A "metahunt_weight_model" object produced by fit_weight_model(). Must have weight_model$K == nrow(dfspa_fit$bases). |
| W_new | A matrix or data frame of study-level covariates for the new target studies, with columns matching those used to fit weight_model. |

## Value

An nrow(W_new)-by-G_grid numeric matrix; row j is the predicted target function on the grid for the j-th new study.

## See Also

apply_wrapper() to reduce predicted functions to scalars.

## Examples

```
set.seed(1)
G <- 40; m <- 60; K_true <- 3
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)

# generate study-level covariates and softmax weights
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
beta <- cbind(c(1, -0.8), c(-0.5, 1.2), c(0, 0))
eta  <- as.matrix(W) %*% beta
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.02), m, G)

fit    <- dfspa(F_hat, K = K_true)
```

```
pi_hat <- project_to_simplex(F_hat, fit$bases)
wm     <- fit_weight_model(pi_hat, W)

W_new   <- data.frame(w1 = c(0, 1), w2 = c(0, -1))
f_pred  <- predict_target(fit, wm, W_new)
dim(f_pred)   # 2 x G
```

---

`print.metahunt_denoising_search` *Print method for d-fSPA denoising parameter search results*

---

## Description

Print method for d-fSPA denoising parameter search results

## Usage

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

## Arguments

| | |
|---|---|
| `x` | An object of class `metahunt_denoising_search` returned by `select_denoising_params()`. |
| `...` | Unused; present for S3 generic compatibility. |

## Value

Invisibly returns `x`.

---

`print.summary.metahunt` *Print a* `summary.metahunt` *object*

---

## Description

Print a `summary.metahunt` object

## Usage

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

## Arguments

| | |
|---|---|
| `x` | A `"summary.metahunt"` object from `summary.metahunt()`. |
| `...` | Ignored. |

## Value

Invisibly returns `x`.

## Examples

```
set.seed(1)
G <- 25; m <- 40
x_grid <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x_grid), cos(pi * x_grid), x_grid)
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- as.matrix(W) %*% cbind(c(1, -0.8), c(-0.5, 1.2), c(0, 0))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

fit <- metahunt(F_hat, W, K = 3, dfspa_args = list(denoise = FALSE))
print(summary(fit))
```

---

`project_to_simplex` *Project study-level functions onto the simplex spanned by basis functions*

---

## Description

For each study `i`, solves the constrained projection

$$\hat{\pi}_i = \arg\min_{\pi \in \Delta_{K-1}} \left\| \hat{f}^{(i)} - \sum_{k=1}^{K} \pi_k \hat{g}_k \right\|_{L^2(\mu)}$$

where the norm is the weighted $L^2$ norm defined by `grid_weights`. This is Equation (3) of the paper and yields the study-specific weights `hat pi_i` used downstream for weight-model fitting and prediction.

## Usage

```
project_to_simplex(F_hat, bases, grid_weights = NULL, ridge = 1e-10)
```

## Arguments

| | |
|---|---|
| `F_hat` | An `m`-by-`G_grid` numeric matrix; row `i` is the study function $\hat{f}^{(i)}$ evaluated on the shared grid. |
| `bases` | A `K`-by-`G_grid` numeric matrix of basis functions on the same grid, typically the `bases` slot of a `dfspa()` result. |
| `grid_weights` | Optional length-`G_grid` non-negative numeric vector of grid weights defining the $L^2(\mu)$ inner product. Defaults to uniform weights `1 / G_grid`. |
| `ridge` | Small non-negative scalar added to the diagonal of the QP Hessian for numerical stability. Defaults to `1e-10`. |

## Details

The projection reduces to the quadratic program

$$\min_{\pi \in \mathbb{R}^K} \ \pi^\top D\,\pi - 2\,d^\top \pi \quad \text{s.t.} \quad \mathbf{1}^\top \pi = 1,\ \pi \geq 0$$

with $D = GWG^\top$ and $d = GW f^{(i)}$, where $G$ is the K-by-G_grid basis matrix, $W = \mathrm{diag}(\texttt{grid\_weights})$, and $f^{(i)}$ is the i-th row of F_hat. Solved via quadprog::solve.QP(). A tiny ridge is added to D for numerical stability.

## Value

An m-by-K numeric matrix of simplex weights; rows sum to 1 and entries are non-negative.

## Examples

```
set.seed(1)
G <- 40
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
true_pi <- rbind(diag(3), c(0.4, 0.3, 0.3), c(0.1, 0.7, 0.2))
F_hat <- true_pi %*% basis
fit <- dfspa(F_hat, K = 3, denoise = FALSE)
pi_hat <- project_to_simplex(F_hat, fit$bases)
round(pi_hat, 3)
```

---

# reconstruction_error_curve

*Reconstruction-error curve for basis-rank selection*

---

## Description

For each candidate number of bases K, run dfspa() followed by project_to_simplex() and report the average projection residual

$$\mathcal{E}(K) = \frac{1}{m} \sum_{i=1}^{m} \left\| \hat{f}^{(i)} - \sum_{k=1}^{K} \hat{\pi}_{ik} \hat{g}_k \right\|_{L^2(\mu)}.$$

Plotting error against K typically shows an elbow.

## Usage

```
reconstruction_error_curve(
  F_hat,
  K_range = NULL,
  grid_weights = NULL,
  dfspa_args = list()
)
```

### Arguments

| | |
|---|---|
| F_hat | An m-by-G_grid numeric matrix of study-level function evaluations on the shared grid. |
| K_range | Integer vector of candidate K values. Defaults to 2:min(nrow(F_hat) - 1, 10). |
| grid_weights | Optional length-G_grid non-negative numeric vector used for the $L^2(\mu)$ norm; defaults to uniform. |
| dfspa_args | Named list of extra arguments passed to dfspa(), e.g. list(denoise = FALSE, N = 2). |

### Details

This is the unsupervised rank-selection criterion of Section 3.2 of the paper (Equation for $\mathcal{E}(K)$). It does not require study-level covariates.

### Value

A data frame with columns K (integer) and error (numeric). Rows where dfspa() or the projection fails are reported with error = NA and a single warning summarising the failures.

### Examples

```
set.seed(1)
G <- 40
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
m <- 50
pi_mat <- matrix(stats::rgamma(m * 3, shape = 0.5), m, 3)
pi_mat <- pi_mat / rowSums(pi_mat)
F_hat  <- pi_mat %*% basis + matrix(stats::rnorm(m * G, sd = 0.02), m, G)

elbow <- reconstruction_error_curve(F_hat, K_range = 2:6)
elbow
```

---

## select_denoising_params

*Choose d-fSPA denoising parameters by cross-validation*

---

### Description

At a fixed K, performs k-fold cross-validation over a grid of denoising parameter pairs (N, Delta) for dfspa(). For each candidate pair and each fold, the full MetaHunt pipeline is fit on the out-of-fold studies and predicts the held-out studies' functions. The pair with the lowest average prediction error is selected.

## Usage

```
select_denoising_params(
  F_hat,
  W,
  K,
  N_grid = NULL,
  Delta_grid = NULL,
  n_folds = 5L,
  grid_weights = NULL,
  dfspa_args = list(),
  weight_model_args = list(),
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| F_hat | An m-by-G_grid numeric matrix of study-level function evaluations. |
| W | An m-by-p matrix or data frame of study-level covariates. |
| K | Integer number of basis functions (fixed during this search). |
| N_grid | Optional numeric vector of candidate N values. Defaults to c(0.2, 0.5, 1.0) * log(nrow(F_hat)). |
| Delta_grid | Optional numeric vector of candidate Delta values. Defaults to c(0.05, 0.10, 0.20, 0.30) times the maximum pairwise $L^2(\mu)$ distance among studies. |
| n_folds | Integer number of folds (default 5). |
| grid_weights | Optional length-G_grid non-negative numeric vector. |
| dfspa_args | Named list of additional arguments for dfspa() (e.g. list(); do not include N, Delta, or denoise here, they are set by the search). |
| weight_model_args | Named list of additional arguments for fit_weight_model(). |
| seed | Optional integer seed for reproducible fold assignment. |

## Details

This is the cross-validated tuning of the denoising parameters discussed in Section 3.1 of the paper. Joint tuning over (K, N, Delta) is not supported because it scales poorly; if you also want to choose K, do it first via cv_error_curve() and then call this function at the selected K.

Default candidate grids:

- N_grid = m * c(NA, NA, NA) resolved at runtime to c(0.2, 0.5, 1.0) * log(m).
- Delta_grid = max_pairwise_dist * c(0.05, 0.10, 0.20, 0.30).

Pass your own N_grid / Delta_grid (in original units) to override.

## Value

An object of class `metahunt_denoising_search`: a list with

`grid` A data frame with one row per `(N, Delta)` pair, columns `N`, `Delta`, `cv_error`, `cv_se`, `n_folds_ok`.

`best` A list with the `(N, Delta)` minimising `cv_error`.

`K`, `n_folds`, `grid_weights` Inputs echoed back for traceability.

## See Also

`cv_error_curve()` for selecting `K`, `dfspa()` for the underlying basis-hunting algorithm.

## Examples

```
set.seed(1)
G <- 30; m <- 80
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- as.matrix(W) %*% cbind(c(1, -0.5), c(-0.4, 1), c(0, 0))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

tune <- select_denoising_params(F_hat, W, K = 3, n_folds = 4, seed = 1)
tune$grid
tune$best
```

---

# split_conformal *Split conformal prediction intervals for target-function predictions*

---

## Description

Implements Algorithm 2 of the paper (split conformal prediction) over the MetaHunt pipeline. Studies are partitioned into training and calibration sets. The training set is used to fit d-fSPA, the constrained projection, and the weight model; the calibration set supplies conformity scores, which determine the width of the intervals.

## Usage

```
split_conformal(
  F_hat,
  W,
  W_new,
  K,
  alpha = 0.05,
  cal_frac = 0.3,
```

```
  wrapper = NULL,
  grid_weights = NULL,
  calibration_idx = NULL,
  dfspa_args = list(),
  weight_model_args = list(),
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| F_hat | An m-by-G_grid numeric matrix of study-level function evaluations. |
| W | An m-by-p matrix or data frame of study-level covariates. |
| W_new | A matrix or data frame of new target covariates. Must contain columns matching W. |
| K | Integer number of basis functions. |
| alpha | Miscoverage level; interval has nominal coverage $1 - \alpha$. Default 0.05. |
| cal_frac | Numeric in (0, 1) giving the fraction of studies in the calibration set. Default 0.3. Ignored if calibration_idx is supplied. |
| wrapper | Optional reduction function (see apply_wrapper()). If NULL, intervals are constructed pointwise at every grid point. |
| grid_weights | Optional length-G_grid non-negative numeric vector used for the $L^2(\mu)$ norm and for the default wrapper. |
| calibration_idx | Optional integer vector of row indices in F_hat to use as the calibration set. If supplied, cal_frac is ignored. |
| dfspa_args, weight_model_args | Named lists passed to dfspa() and fit_weight_model() respectively. |
| seed | Optional integer seed for reproducible train/calibration splits. |

## Details

Given a target function, one can either construct intervals **pointwise at every grid point** (when wrapper = NULL) or for a **scalar summary** of the target function (when wrapper is a function).

- **Pointwise** (wrapper = NULL): for each grid point g the conformity score is $R_{i,g} = |\hat{f}^{(i)}(x_g) - \tilde{f}^{(i)}(x_g)|$ across calibration studies i. A separate $(1-\alpha)$-quantile $q_g$ is computed per grid point, and the interval at grid point g for target j is $[\tilde{f}^{(j)}(x_g) - q_g,\ \tilde{f}^{(j)}(x_g) + q_g]$.
- **Scalar** (wrapper supplied): conformity scores are $R_i = |s(\hat{f}^{(i)}) - s(\tilde{f}^{(i)})|$ with s = wrapper, and the interval for each target is $[s(\tilde{f}^{(j)}) - q,\ s(\tilde{f}^{(j)}) + q]$ with a single shared quantile $q$.

The finite-sample quantile is $q = R_{(k)}$ with $k = \lceil (1-\alpha)(n_{\text{cal}} + 1) \rceil$; if $k > n_{\text{cal}}$, q = Inf and intervals are $(-\infty, \infty)$.

## Value

An object of class "metahunt_conformal": a list with

prediction Point predictions for W_new. A numeric vector of length nrow(W_new) in the scalar case, or an nrow(W_new)-by-G_grid matrix in the pointwise case.

lower, upper Interval endpoints, same shape as prediction.

alpha Miscoverage level used.

method "split".

n_cal Calibration sample size.

quantile The conformal quantile: a scalar (scalar case) or a length-G_grid vector (pointwise case).

wrapper The wrapper used, or NULL.

## Examples

```
set.seed(1)
G <- 40; m <- 80; K_true <- 3
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- as.matrix(W) %*% cbind(c(1, -0.8), c(-0.5, 1.2), c(0, 0))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

W_new <- data.frame(w1 = c(0, 1), w2 = c(0, -1))
# pointwise intervals at every grid point
pi_grid <- split_conformal(F_hat, W, W_new, K = 3, seed = 1)
dim(pi_grid$lower)  # 2 x 40
# scalar intervals for the grid-weighted mean (ATE-style)
pi_ate  <- split_conformal(F_hat, W, W_new, K = 3, wrapper = mean, seed = 1)
pi_ate$prediction
```

---

summary.metahunt *Summarise a MetaHunt fit*

---

## Description

Produces a compact summary of a "metahunt" object, including study/grid sizes, the weight-model method, per-basis summary statistics for the training simplex weights pi_hat, and denoising bookkeeping from the underlying dfspa() fit.

## Usage

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

### Arguments

| | |
|---|---|
| object | A "metahunt" object from metahunt(). |
| ... | Ignored. |

### Value

An object of class "summary.metahunt": a list with components

m Number of studies.

G_grid Grid size.

K Number of basis functions.

weight_method Method used by the weight model.

predictor_names Character vector of covariate names.

pi_summary A K-by-5 numeric matrix; each row gives min, mean, median, max, and sd of the corresponding column of object$pi_hat.

n_kept Number of studies retained after denoising.

n_dropped Number of studies dropped (m - n_kept).

denoising List with N and Delta from the dfspa fit (both NA when denoise = FALSE).

### Examples

```
set.seed(1)
G <- 25; m <- 40
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m), w2 = rnorm(m))
eta <- as.matrix(W) %*% cbind(c(1, -0.8), c(-0.5, 1.2), c(0, 0))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)

fit <- metahunt(F_hat, W, K = 3, dfspa_args = list(denoise = FALSE))
summary(fit)
```

---

## summary.metahunt_conformal

*Summarise a conformal prediction-interval object*

---

### Description

Produces a small list of descriptive statistics about a "metahunt_conformal" object: interval widths, quantile summaries, and calibration diagnostics. Returns an object of class "summary.metahunt_conformal" with a matching print method.

## Usage

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

## Arguments

| | |
|---|---|
| object | A "metahunt_conformal" object from split_conformal(), cross_conformal(), or conformal_from_fit(). |
| ... | Unused; present for S3 generic consistency. |

## Value

A list of class "summary.metahunt_conformal". In pointwise mode (no wrapper) the list contains n_targets, G_grid, n_cal, alpha, method, mean_interval_width, frac_finite_quantile, quantile_summary, and wrapper. In scalar mode (wrapper supplied) the list contains n_targets, n_cal, alpha, method, mean_interval_width, quantile, quantile_finite, and wrapper.

## Examples

```
set.seed(1)
G <- 25; m <- 60
x <- seq(0, 1, length.out = G)
basis <- rbind(sin(pi * x), cos(pi * x), x)
W <- data.frame(w1 = rnorm(m))
eta <- cbind(0.6 * W$w1, -0.3 * W$w1, rep(0, m))
pi_true <- exp(eta) / rowSums(exp(eta))
F_hat <- pi_true %*% basis + matrix(rnorm(m * G, sd = 0.05), m, G)
W_new <- data.frame(w1 = c(0, 1))
res <- split_conformal(F_hat, W, W_new, K = 3,
                       dfspa_args = list(denoise = FALSE), seed = 1)
summary(res)
```

# Index