Package: kernR (via r-universe)

June 3, 2026

Title Kernel-Based Causal Distributional Testing

Version 0.7.0.9000

Description Kernel-based hypothesis tests for causal inference and distributional treatment effects. Implements backdoor-adjusted HSIC (bd-HSIC) for testing causal association, and doubly robust kernel statistics (DR-DATE, DR-DETT) for testing distributional treatment effects beyond mean shifts. Supports binary, continuous, and mixed treatments, hierarchical/nested data structures, and scales to large datasets via Nystrom approximation and random Fourier features. This package depends on 'PESTO' (GPL (>= 3)); kernR's own sources are released under the MIT licence, but the installed combination with 'PESTO' is a combined work subject to the terms of the GPL (>= 3).

License MIT + file LICENSE

URL https://github.com/max578/kernR, https://max578.github.io/kernR

BugReports https://github.com/max578/kernR/issues

Depends R (>= 4.1.0)

Imports data.table, methods, Rcpp, stats, PESTO (>= 0.6.0)

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, proxymix (>= 0.3.0), ranger, rmarkdown, testthat (>= 3.0.0), vdiffr, xgboost

Remotes github::max578/PESTO

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository https://max578.r-universe.dev

Date/Publication 2026-06-03 01:02:57 UTC

RemoteUrl https://github.com/max578/kernR

RemoteRef main

RemoteSha 0b7ac899935e81f89518c0522cf5439b3de8a0c2

Contents

`aggregate_downscale` *Aggregate-Likelihood Downscaling*

Description

Inverts a known aggregation operator `Y = T(X) + eps` to recover the fine-scale latent `X` from coarse / aggregate observations `Y`, using a Gaussian-mixture prior on the latent. Implements the aggregate-likelihood / kernel-downsizing framework (Sejdinovic et al.) as a kernR-side method consuming an optional `proxymix::fit_proxymix()` latent prior.

Usage

```
aggregate_downscale(
  y,
  aggregator,
  latent_prior,
  sigma_y = 0.1,
  n_samples_per_component = 200L,
  min_ess_fraction = 0.1,
  seed = NULL
)
```

Arguments

`y`	Numeric vector of length `dim_y`, or `1 x dim_y` matrix. The observed aggregate.
`aggregator`	Either a numeric `dim_y x dim_x` matrix (treated as a linear aggregator and dispatched to the closed-form path) or a function `function(x) -> y_matrix` that maps an `n x dim_x` matrix of latent samples to an `n x dim_y` matrix of aggregates (non-linear IS path).
`latent_prior`	Either (a) a list with elements `means` (`N x dim_x` matrix or list of `dim_x`-vectors), `covariances` (list of `N` `dim_x x dim_x` matrices), `weights` (length-`N` numeric, summing to 1) – or (b) a `proxymix::gmm_fit` (any object exposing `@means`, `@covariances`, `@weights` slots).
`sigma_y`	Numeric. Observation noise standard deviation (scalar; `eps ~ N(0, sigma_y^2 I)`). Default `0.1`.
`n_samples_per_component`	Integer. Importance-sampling sample count per prior component (non-linear path only). Default `200L`.
`min_ess_fraction`	Numeric in `(0, 1]`. ESS-floor reliability gate for the IS path: when per-component ESS drops below `min_ess_fraction * n_samples_per_component`, a warning is emitted. Default `0.1`. Set `0` to disable.

seed Integer or NULL. Random seed (non-linear path).

Details

This is the **third** downscaling method in kernR, structurally different from the existing pair:

- kernel_downscale() (CME, Park-Muandet-Fukumizu-Sejdinovic 2013) – paired (coarse, fine) training data; supervised regression.
- dist_regression() (Szabo-Sriperumbudur-Poczos-Gretton 2016) – distribution-to-distribution mapping from a bag-of-points design.
- aggregate_downscale() (this function) – only aggregate observations + known aggregator + parametric latent prior. Used when no paired training data exists (only coarse-grid observations) but the aggregation operator is known (spatial averaging, temporal averaging, linear or non-linear projection).

Two computational paths, selected on aggregator's class:

- **Linear-Gaussian closed form** (when aggregator is a matrix A): each prior component's posterior is a Kalman update; K_k = Sigma_k A^T (A Sigma_k A^T + sigma_y^2 I)^{-1}; mu_k|y = mu_k + K_k (y - A mu_k), Sigma_k|y = (I - K_k A) Sigma_k; posterior mixture weights reweight by per-component evidence N(y | A mu_k, A Sigma_k A^T + sigma_y^2 I).
- **Non-linear importance sampling** (when aggregator is a function): draw n_samples_per_component samples from each prior component, evaluate T(.), weight by Gaussian likelihood N(y | T(x), sigma_y^2 I), recover posterior moments + reweighted mixture weights from the importance-weighted samples. Reports per-component effective sample size; warns when below a stated floor.

The aggregate-likelihood / GMM-proxy direction is shared with the companion proxymix Tier-2 stub proxymix::from_aggregate_likelihood(), which targets the same problem from the prior-fitting side; this function targets it from the consumption side (inversion given a fitted prior).

Value

An object of class "aggregate_downscale" with components:

- posterior_mean – length-dim_x numeric, E[X | y].
- posterior_cov – dim_x x dim_x matrix, Cov[X | y] (law-of-total-covariance over mixture components).
- posterior_weights – length-N numeric, posterior mixture weights (sum to 1).
- posterior_components_means – list of N length-dim_x posterior component means.
- posterior_components_covariances – list of N posterior component covariances.
- aggregator_type – "linear" or "nonlinear".
- method – "linear_closed_form" or "nonlinear_is".
- ess_per_component – length-N per-component ESS (IS path only; NA for closed form).
- ess_warning – TRUE if any per-component ESS fell below the floor (IS path only).
- n_components, sigma_y, n_samples_per_component, call.

See Also

kernel_downscale(), dist_regression().

Other downscaling and embeddings: dist_regression(), fit_cme(), kernel_downscale(), posterior_sample_aggregate()

Examples

```
set.seed(1L)
# Linear-Gaussian: spatial averaging of two adjacent cells.
A <- matrix(c(0.5, 0.5), nrow = 1L)
prior <- list(
  means = list(c(0, 0), c(2, 2)),
  covariances = list(diag(2L), diag(2L)),
  weights = c(0.5, 0.5)
)
fit <- aggregate_downscale(y = 1.0, aggregator = A,
                           latent_prior = prior, sigma_y = 0.2)
fit$posterior_mean

# Non-linear: aggregator is sin of the sum.
agg_fn <- function(x) matrix(sin(rowSums(x)), ncol = 1L)
fit2 <- aggregate_downscale(y = 0.5, aggregator = agg_fn,
                            latent_prior = prior, sigma_y = 0.1,
                            n_samples_per_component = 300L, seed = 1L)
fit2$posterior_mean
```

---

assess_overlap *Assess Propensity Score Overlap*

Description

Diagnoses overlap (positivity) between treated and control groups by summarising the propensity score distributions.

Usage

```
assess_overlap(propensity, treatment = NULL)
```

Arguments

propensity	A propensity_fit object or a numeric vector of scores.
treatment	Binary treatment vector. Required if propensity is a numeric vector.

Value

A list of class "overlap_diagnostic" with:

treated Summary statistics of propensity scores for treated.

control Summary statistics for controls.

overlap_warning Logical. TRUE if overlap is poor.

See Also

Other density ratio and propensity: effective_sample_size(), estimate_density_ratio(), estimate_propensity(), fit_density_ratio(), plot_weights(), predict_density_ratio()

Examples

```
set.seed(1L)
n <- 200L
treatment <- rbinom(n, 1L, 0.5)
scores <- plogis(rnorm(n) + 0.6 * treatment)
assess_overlap(scores, treatment)
```

---

bd_hsic_test *Backdoor-HSIC Test for Causal Association*

Description

Tests the do-null hypothesis H_0: p(y | do(x)) = p*(y) using a kernel-based test with backdoor adjustment via density ratio estimation. Detects causal associations including non-linear effects that standard linear methods miss.

Usage

```
bd_hsic_test(
  x,
  y,
  z,
  kernel_x = kernel_spec(),
  kernel_y = kernel_spec(),
  density_ratio = c("logistic", "ranger", "xgboost", "proxymix", "rulsif"),
  n_permutations = 500L,
  n_clusters = "auto",
  split_ratio = 0.5,
  alpha = 0.05,
  seed = NULL,
  verbose = FALSE,
  cluster_id = NULL,
  permutation = c("auto", "within_cluster", "naive"),
  min_ess_fraction = 0.1
)
```

Arguments

x	Numeric vector or matrix. Treatment variable.
y	Numeric vector or matrix. Outcome variable.
z	Numeric matrix, data.frame, or data.table. Confounders.

`kernel_x` Kernel specification for treatment space. Default is RBF with median heuristic.

`kernel_y` Kernel specification for outcome space. Default is RBF with median heuristic.

`density_ratio` Character. Method for density ratio estimation: `"logistic"` (default), `"ranger"`, `"xgboost"`, `"proxymix"`, or `"rulsif"`. The `"proxymix"` backend fits Gaussian-mixture proxies to the joint and product-of-marginals sample clouds via classical EM (Hoek & Elliott, 2024), giving a parametric alternative to NCE-based classifiers; useful for multimodal densities or when classifier calibration is unreliable. Requires the `proxymix` package (`>= 0.3.0`).

`n_permutations` Integer. Number of permutations for the null distribution. Default is 500.

`n_clusters` Integer or `"auto"`. Number of *propensity* clusters for valid permutation when `cluster_id = NULL`. Default is `"auto"`.

`split_ratio` Numeric in (0, 1). Proportion of data for training the density ratio estimator. Default is 0.5.

`alpha` Numeric. Significance level. Default is 0.05.

`seed` Integer or `NULL`. Random seed for reproducibility.

`verbose` Logical. Print progress. Default is `FALSE`.

`cluster_id` Optional vector of length `nrow(x)` identifying external clusters (e.g. site, season, paddock, farm). When supplied, the permutation null is built by within-cluster reshuffling of `y`, which preserves cluster-level effects in the null. Coerced to factor; the test split inherits the cluster assignment. The result then carries a per-cluster stratified bd-HSIC alongside the pooled statistic.

`permutation` Character. Permutation scheme:

- `"auto"` (default) – when `cluster_id` is supplied, equivalent to `"within_cluster"`; otherwise falls back to k-means clustering on propensity weights (the original Hu/Sejdinovic/Evans scheme).
- `"within_cluster"` – requires `cluster_id`. Permutes `y` indices only within clusters; preserves cluster-level effects.
- `"naive"` – unrestricted permutation across all observations. Use only when independence within clusters is plausible (rarely true in ag-systems data).

`min_ess_fraction` Numeric in `(0, 1)` or `0` / non-finite to disable. ESS-floor reliability gate (added 0.0.0.9013): if the weighted-HSIC effective sample size is below `min_ess_fraction * n_test`, a warning is emitted and `result$ess_warning` is `TRUE`. The default `0.1` (10%) is a conservative floor; tighten it for studies with strict reliability requirements.

Details

The bd-HSIC test (Hu, Sejdinovic & Evans, 2024) tests whether treatment X has a causal effect on outcome Y after adjusting for confounders Z via the backdoor criterion.

The test works by:

1. Estimating density ratios w(x, z) = p*(x) / p(x|z) to reweight observational samples to the interventional distribution.
2. Computing a weighted HSIC statistic between X and Y.

3. Obtaining p-values via permutation of Y within exchangeability clusters – propensity-similarity clusters by default, or external design clusters (site / season / paddock) when `cluster_id` is supplied.

Hierarchical extension. When the design is naturally clustered (multi-site agricultural trials, paddock x season factorial designs, patient x hospital data), supplying `cluster_id` activates within-cluster permutation: indices of `y` are reshuffled only within each cluster, preserving cluster-level effects in the null. This is the safer default for clustered data; naive permutation across clusters can inflate Type I error when cluster effects exist.

Unlike PDS or Double ML, bd-HSIC can detect non-linear causal effects (e.g., U-shaped relationships) where the treatment affects higher moments of the outcome but not necessarily the mean.

Value

An object of class `"kernel_test_result"`. When `cluster_id` is supplied, the result additionally carries:

permutation_scheme Character: which scheme was used.

cluster_id Integer cluster assignment on the test split.

cluster_levels Character cluster labels.

per_cluster_statistic Per-cluster weighted HSIC (stratified contributions); `NA` for clusters with `< 2` test observations.

Train/test split (0.0.0.9014)

The density-ratio estimator is now **fit on the train split and predicted on the held-out test split** via `fit_density_ratio()` + `predict_density_ratio()`. The documented `split_ratio` is honoured end-to-end for all four classifier / proxymix backends. The 0.0.0.9013 sample-split leak warning is therefore retired. RuLSIF, the kernel-based closed-form backend, still uses `estimate_rulsif()` on the train/test split natively.

The fitted density-ratio model is preserved on `result$density_ratio_fit` for callers that want backend diagnostics (see `?fit_density_ratio` Value; proxymix exposes BIC, AIC, log-likelihood, convergence per GMM).

References

Hu, R., Sejdinovic, D., & Evans, R. J. (2024). A kernel test for causal association via noise contrastive backdoor adjustment. *JMLR*, 25(160), 1-56.

See Also

Other causal association tests: `hierarchical_test()`, `kernel_causal_test()`

Examples

```
set.seed(42)
n <- 300
z <- matrix(rnorm(n * 2), n, 2)
x <- z[, 1] + rnorm(n)
```

```
y <- 0.5 * x + z[, 2] + rnorm(n, sd = 0.5)

result <- bd_hsic_test(x, y, z, n_permutations = 200, seed = 1)
print(result)
```

---

concordance_test *Kernel k-sample Concordance Test*

Description

Tests whether two or more samples come from a common distribution, using the summed pairwise Maximum Mean Discrepancy with a joint-permutation null. The samples are typically posterior draws from different inference engines, or scenario ensembles from different simulators; the test asks whether they are mutually concordant. Unlike repeated two-sample `mmd_test()` calls, the null is a single shared relabeling of the pooled sample, so the family-wise error is controlled and the overall verdict is one calibrated p-value.

Usage

```
concordance_test(
  x,
  kernel = kernel_spec(),
  n_permutations = 500L,
  alpha = 0.05,
  seed = NULL,
  n_exact_max = 5000L
)
```

Arguments

`x`	A list of two or more samples to compare. Each element is a numeric vector, matrix, or data.frame with `n_k` observations (rows) over a shared `d` columns. A named list labels the sources in the output; an unnamed list is labelled `Source 1`, `Source 2`, and so on. Each sample needs at least five rows.
`kernel`	Kernel specification. Default is RBF with the median heuristic over the pooled sample.
`n_permutations`	Integer. Number of joint permutations for the null. Default `500`.
`alpha`	Numeric in `(0, 1)`. Significance level for the verdict. Default `0.05`.
`seed`	Integer or `NULL`. Random seed for reproducibility.
`n_exact_max`	Integer or `Inf`. Pooled-sample-size ceiling for the exact `O(n^2)` test. Above it, the call is delegated to `concordance_test_nystrom()` (with a message; the verdict object records `approximation = "nystrom"`). `Inf` forces the exact test at any size. Default `5000L`.

Details

The returned object carries the full pairwise MMD discrepancy matrix, so a rejection can be read down to the offending pair: convergence across sources is corroborating evidence, and divergence localises which source departs and on which margin. This is the cross-engine concordance role – a sample-based complement to the score-based `ksd_test()`.

The exact test materialises the pooled `n x n` kernel matrix (`O(n^2)`). To keep large ensembles tractable without a silent loss of exactness, a pooled sample with more than `n_exact_max` rows is delegated to `concordance_test_nystrom()` – a low-rank approximation that is *announced* by a message and *recorded* in the returned object's `approximation` and `m` fields. Set `n_exact_max = Inf` to force the exact test, or call `concordance_test_nystrom()` directly to control the approximation rank `m`.

Value

An object of class `c("concordance_test", "kernel_test_result")` carrying the standard `kernel_test_result` fields plus:

statistic Summed pairwise unbiased MMD-squared.

p_value Upper-tail joint-permutation p-value (with +1 correction).

n_groups Number of sources compared.

group_sizes Named integer vector of per-source sample sizes.

pairwise Symmetric `K x K` matrix of pairwise unbiased MMD-squared, row/column-named by source.

alpha, reject Verdict level and `p_value <= alpha`.

Author(s)

Max Moldovan, <max.moldovan@adelaide.edu.au>

References

Gretton, A., Borgwardt, K. M., Rasch, M. J., Scholkopf, B., & Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, 13, 723-773.

See Also

`mmd_test()`, `ksd_test()`, `mmd_ppc()`

Other goodness-of-fit tests: `concordance_test_nystrom()`, `coverage_test()`, `gaussian_score()`, `ksd_test()`, `ksd_test_nystrom()`, `numeric_score()`

Examples

```
set.seed(1)

# Three concordant sources (same distribution): not rejected
draws <- list(
  engine_a = matrix(stats::rnorm(400L), ncol = 2L),
  engine_b = matrix(stats::rnorm(400L), ncol = 2L),
```

```
  engine_c = matrix(stats::rnorm(400L), ncol = 2L)
)
fit_ok <- concordance_test(draws, n_permutations = 199L, seed = 1L)
fit_ok

# One source departs (mean-shifted): rejected, and the pairwise matrix
# localises engine_c
draws$engine_c <- draws$engine_c + 1
fit_bad <- concordance_test(draws, n_permutations = 199L, seed = 1L)
fit_bad$pairwise
```

`concordance_test_nystrom` *Accelerated Kernel k-sample Concordance Test (Nystrom / RFF)*

Description

Low-rank counterpart to `concordance_test()` for large ensembles. The pooled sample is factorised once – by the Nystrom method (default) or random Fourier features – into an `n x m` factor `F` with $FF^\top \approx K$; the summed pairwise unbiased MMD-squared and its joint-permutation null are then computed from `F` in `O(n m)` per permutation rather than `O(n^2)`, with `m << n` controlling the speed / accuracy trade-off. The verdict object and its interpretation are identical to `concordance_test()`; only the cost scales differently.

Usage

```
concordance_test_nystrom(
  x,
  kernel = kernel_spec(),
  method = c("nystrom", "rff"),
  m = 100L,
  n_permutations = 500L,
  alpha = 0.05,
  seed = NULL,
  regularise = 1e-06
)
```

Arguments

`x`	A list of two or more samples to compare. Each element is a numeric vector, matrix, or data.frame with `n_k` observations (rows) over a shared `d` columns. A named list labels the sources in the output; an unnamed list is labelled `Source 1`, `Source 2`, and so on. Each sample needs at least five rows.
`kernel`	Kernel specification. Default is RBF with the median heuristic over the pooled sample.
`method`	Character. `"nystrom"` (default) or `"rff"`. RFF requires an RBF `kernel`.

m	Integer. Rank of the approximation: the number of Nystrom landmarks or RFF features. Larger m improves accuracy at higher cost. Default 100L.
n_permutations	Integer. Number of joint permutations for the null. Default 500.
alpha	Numeric in (0, 1). Significance level for the verdict. Default 0.05.
seed	Integer or NULL. Random seed for reproducibility.
regularise	Small positive numeric. Ridge added before the Nystrom Cholesky for numerical stability; ignored under method = "rff". Default 1e-6.

Details

The factorisation of the pooled sample preserves the per-source mean embeddings (per-source column sums of F), so the pairwise discrepancy matrix still localises which source departs. The joint-permutation null is built by relabelling the rows of F – the low-rank analogue of permuting the pooled-sample labels in concordance_test().

Use concordance_test() for exact results at moderate n; reach for this function when the pooled sample is large enough that the O(n^2) kernel matrix is the bottleneck. RFF (method = "rff") requires an RBF kernel; Nystrom supports any kernel_spec().

Value

An object of class c("concordance_test", "kernel_test_result") carrying the same fields as concordance_test() plus:

approximation "nystrom" or "rff".

m Effective rank used for the factorisation.

Author(s)

Max Moldovan, <max.moldovan@adelaide.edu.au>

References

Gretton, A., Borgwardt, K. M., Rasch, M. J., Scholkopf, B., & Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, 13, 723-773.

Williams, C. K. I., & Seeger, M. (2001). Using the Nystrom method to speed up kernel machines. *NeurIPS*, 13.

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *NeurIPS*, 20.

See Also

concordance_test(), nystrom_factor(), rff_features(), hsic_test_nystrom()

Other goodness-of-fit tests: concordance_test(), coverage_test(), gaussian_score(), ksd_test(), ksd_test_nystrom(), numeric_score()

Other low-rank acceleration: hsic_test_nystrom(), ksd_test_nystrom(), nystrom_factor(), rff_features()

Examples

```
set.seed(1)
big <- list(
  engine_a = matrix(stats::rnorm(4000L), ncol = 2L),
  engine_b = matrix(stats::rnorm(4000L), ncol = 2L),
  engine_c = matrix(stats::rnorm(4000L), ncol = 2L) + 0.4
)
fit <- concordance_test_nystrom(big, m = 80L,
                                n_permutations = 199L, seed = 1L)
fit
fit$pairwise
```

`coverage_test` *Coverage / Calibration Diagnostic for a Predictive Ensemble*

Description

Quantifies *how* a predictive ensemble is calibrated against held-out observations, rather than only testing whether it differs from them. Each observation is mapped to its probability integral transform (PIT) within the ensemble; calibrated draws give uniform PITs. The result reports empirical coverage at nominal interval levels, a signed dispersion ratio, a bias indicator, and a rank-histogram uniformity test, and classifies the ensemble as calibrated, under-dispersed (over-confident), over-dispersed, or biased.

Usage

```
coverage_test(x, ...)

## Default S3 method:
coverage_test(
  x,
  observed,
  levels = c(0.5, 0.8, 0.9),
  n_bins = 10L,
  alpha = 0.05,
  ...
)

## S3 method for class 'pesto_ensemble'
coverage_test(x, observed = NULL, ...)

## S3 method for class 'pesto_ensemble_manifest'
coverage_test(x, observed, outputs = NULL, ...)
```

Arguments

x	Numeric matrix n_draws x d of predictive draws, a pesto_ensemble (see pesto_ensemble()), or a pesto_ensemble_manifest.
...	Additional arguments passed between methods (currently unused).
observed	Numeric matrix n_obs x d of held-out observations. When x is a pesto_ensemble carrying an observed slot, may be NULL.
levels	Numeric vector in (0, 1). Central predictive-interval levels at which to report empirical coverage. Default c(0.5, 0.8, 0.9).
n_bins	Integer. Number of equal-width bins for the rank-histogram uniformity test. Default 10.
alpha	Numeric in (0, 1). Significance level for the calibration verdict. Default 0.05.
outputs	Optional character vector of manifest output columns to test (the real_name column is always excluded). Defaults to all numeric output columns. Used only for the pesto_ensemble_manifest method.

Details

This is the graded complement to the binary kernel verdicts mmd_ppc() and ksd_test(). Its motivating use is ensemble over-confidence: an under-dispersed ensemble (for example a collapsed iterative-ensemble-smoother posterior) gives a U-shaped rank histogram, a dispersion ratio above one, and empirical coverage below nominal – all of which this function names and measures.

Calibration is assessed **per output dimension and pooled across dimensions** (marginal calibration); it does not test the joint dependence structure, for which the two-sample mmd_ppc() is the right tool. With few held-out observations the uniformity test has low power; read the coverage and dispersion summaries (which are informative at any sample size) alongside it.

Value

An object of class "coverage_test" with components:

coverage Data frame of nominal vs pooled empirical coverage at each requested level.

coverage_by_dim Matrix of empirical coverage per dimension x level.

dispersion_ratio Var(PIT) / (1/12): above one under-dispersed, below one over-dispersed.

mean_pit Pooled mean PIT (0.5 under calibration; a bias indicator).

calibration List with the rank-histogram chi-squared statistic, p_value, and n_bins.

verdict Character: the calibration classification.

reject Logical: calibration$p_value <= alpha.

pit The n_obs x d matrix of PIT values.

n_draws, n_obs, dimension, levels, alpha Inputs / sizes.

pesto_metadata Provenance carried from a pesto_ensemble_manifest (including fidelity), or NULL.

Author(s)

Max Moldovan, <max.moldovan@adelaide.edu.au>

References

Gneiting, T., Balabdaoui, F., & Raftery, A. E. (2007). Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society B*, 69(2), 243-268.

Hamill, T. M. (2001). Interpretation of rank histograms for verifying ensemble forecasts. *Monthly Weather Review*, 129(3), 550-560.

See Also

`mmd_ppc()`, `ksd_test()`, `pesto_ensemble()`

Other goodness-of-fit tests: `concordance_test()`, `concordance_test_nystrom()`, `gaussian_score()`, `ksd_test()`, `ksd_test_nystrom()`, `numeric_score()`

Examples

```
set.seed(1)
obs <- matrix(stats::rnorm(120L), ncol = 2L)

# Calibrated: predictive ensemble from the same law
ens_ok <- matrix(stats::rnorm(2000L), ncol = 2L)
coverage_test(ens_ok, obs)

# Under-dispersed (over-confident): ensemble too narrow
ens_tight <- matrix(stats::rnorm(2000L, sd = 0.4), ncol = 2L)
coverage_test(ens_tight, obs)
```

---

dist_regression *Kernel Distribution Regression*

Description

Regression where each input is a *bag* of points (a sample from a distribution) rather than a single feature vector. Each bag is implicitly mapped to its empirical mean embedding in the RKHS of an inner kernel; the outer (between-bag) kernel acts on those embeddings, and kernel ridge regression predicts a scalar or multivariate output. This is the Szabó-Sriperumbudur-Póczos-Gretton (2016) "learning theory for distribution regression" setup; kernel-mean-embedding background is from Muandet et al. (2017).

Usage

```
dist_regression(
  bags,
  y,
  inner_kernel = kernel_spec(),
  outer = c("linear", "rbf"),
  outer_bandwidth = "median",
  lambda = "cv"
)
```

Arguments

bags	A list of length M of numeric matrices. Each element is a bag (n_i x d points). All bags must have the same number of columns; bag sizes n_i may differ.
y	Numeric vector of length M, or numeric matrix M x d_y, of training targets.
inner_kernel	A kernel_spec() applied between points within and across bags. Median bandwidth heuristic resolved on the pooled training points. Default RBF with median heuristic.
outer	Character. Outer-kernel form: "linear" (default) or "rbf".
outer_bandwidth	Outer-kernel bandwidth (RBF only). "median" (default) resolves to the median embedding-space pairwise distance on the training Gram; a positive numeric overrides.
lambda	Ridge regularisation for kernel ridge regression. If "cv" (default), selected by leave-one-out CV over 10^seq(-6, 1, length.out = 15).

Details

Outer kernels supported.

- "linear": $K(P_i, P_j) = \langle \hat{\mu}_{P_i}, \hat{\mu}_{P_j} \rangle = \frac{1}{n_i n_j} \sum_{k,l} k(x_i^{(k)}, x_j^{(l)})$.
- "rbf": $K(P_i, P_j) = \exp(-\|\hat{\mu}_{P_i} - \hat{\mu}_{P_j}\|^2/(2\sigma^2))$, where the embedding-space distance is recovered from the inner Gram via $\|\hat{\mu}_i - \hat{\mu}_j\|^2 = G_{ii} - 2G_{ij} + G_{jj}$.

Typical ag-systems use: each paddock contributes a *bag* of soil-core measurements (variable depth, multiple cores per paddock); the regression predicts paddock-level yield from the distributional shape of the soil profile. Distinct from kernel_downscale() in that the input is itself a distribution, not a fixed-length vector.

Value

An object of class "dist_regression" with components:

alpha Ridge weights (length M or M x d_y).

G_train Training bag-inner-mean Gram (M x M).

K_outer_train Outer kernel matrix (M x M) actually used.

bags_train Bags used (kept for prediction).

y_train Targets used.

inner_kernel Resolved inner kernel.

outer, outer_bandwidth Outer kernel choice and resolved bandwidth (NA for linear outer).

lambda Ridge parameter used.

M, d_y, call Metadata.

References

Szabó, Z., Sriperumbudur, B. K., Póczos, B., & Gretton, A. (2016). Learning theory for distribution regression. *Journal of Machine Learning Research*, 17(152), 1-40.

Muandet, K., Fukumizu, K., Sriperumbudur, B., & Scholkopf, B. (2017). *Kernel mean embedding of distributions: A review and beyond.* Foundations and Trends in Machine Learning, 10(1-2).

See Also

`predict.dist_regression()`, `kernel_downscale()`

Other downscaling and embeddings: `aggregate_downscale()`, `fit_cme()`, `kernel_downscale()`, `posterior_sample_aggregate()`

Examples

```
set.seed(1)
# 30 bags, each a sample from N(mu_i, 1); predict mu_i
M <- 30L
mu <- stats::rnorm(M)
bags <- lapply(mu, function(m)
  matrix(stats::rnorm(40, mean = m), ncol = 1L))
fit <- dist_regression(bags, y = mu, outer = "linear")
fit

# Predict at new bags
new_mu <- stats::rnorm(5L)
new_bags <- lapply(new_mu, function(m)
  matrix(stats::rnorm(40, mean = m), ncol = 1L))
predict(fit, new_bags)
```

---

dr_date_scenario *DR-DATE for Two PESTO Ensemble Scenarios*

Description

Tests whether the *distribution* of simulated outputs differs between two APSIM (or other) scenario ensembles – e.g. baseline management vs an intervention such as stubble retention – by running the doubly robust DR-DATE statistic of Fawkes, Hu, Evans & Sejdinovic (2024) over the pooled ensembles.

Usage

```
dr_date_scenario(
  baseline,
  intervention,
  output = NULL,
```

```
  propensity_model = c("logistic", "ranger", "xgboost"),
  outcome_model = c("krr", "zero"),
  n_permutations = 500L,
  n_bins = 10L,
  regularisation = "cv",
  alpha = 0.05,
  seed = NULL,
  verbose = FALSE,
  strict_fidelity = FALSE,
  ...
)
```

Arguments

`baseline`	A `pesto_ensemble_manifest` (S7) – the reference scenario.
`intervention`	A `pesto_ensemble_manifest` (S7) – the alternative scenario. Must share `pesto_version` (major.minor) plus parameter and observation schemas with `baseline`.
`output`	Optional character vector of observation column names to test against. Defaults to all numeric output columns shared by the two manifests (the `real_name` column is excluded). Pass a subset to focus the test on specific outputs (e.g. end-of-season yield only).
`propensity_model`	Forwarded to `dr_date_test()`. Default `"logistic"`. In the scenario context the true propensity is 50/50 by design; logistic recovers that and absorbs any sampling imbalance.
`outcome_model`	Forwarded to `dr_date_test()`. Default `"krr"`.
`n_permutations`	Forwarded to `dr_date_test()`. Default 500.
`n_bins`	Forwarded to `dr_date_test()`. Default 10.
`regularisation`	Forwarded to `dr_date_test()`. Default `"cv"`.
`alpha`	Forwarded to `dr_date_test()`. Default 0.05.
`seed`	Integer or `NULL`. Random seed.
`verbose`	Logical. Default `FALSE`.
`strict_fidelity`	Logical. If `FALSE` (default) a mismatch in the two manifests' multi-fidelity provenance raises a `warning`; if `TRUE` it raises an error. See the *Fidelity provenance* section.
`...`	Reserved.

Details

This is a thin scenario-facing wrapper around `dr_date_test()`: parameters are treated as covariates (so the test adjusts for any systematic difference in the parameter posteriors that came from the two PESTO runs), outputs are the outcome, and the scenario label is the binary treatment. Sensitive to distributional differences (variance, shape, tails), not just mean shifts.

The PESTO 0.3.0 `PESTO::pesto_ensemble_manifest` S7 contract is the supported input shape; the per-realisation file-I/O for ingestion is handled by `PESTO::read_manifest()` upstream of this call.

Value

An object of class `c("dr_date_scenario", "kernel_test_result")` with the standard `kernel_test_result` fields plus:

baseline_run_id Run id from the baseline manifest.

intervention_run_id Run id from the intervention manifest.

n_baseline Realisations in baseline ensemble.

n_intervention Realisations in intervention ensemble.

outputs_tested Character vector of output columns used.

pesto_versions Named character – baseline / intervention.

fidelity List with `baseline` / `intervention` fidelity provenance from the PESTO manifests (a `list(type, schedule, final_level, n_levels, costs)` for a multi-fidelity run, or `NULL` for a single-fidelity run).

Fidelity provenance

A PESTO multi-fidelity run records, in the manifest `fidelity` slot, which fidelity levels produced the ensemble. Comparing a baseline and an intervention that were calibrated at different fidelities risks confounding the distributional contrast with fidelity bias. This wrapper therefore surfaces a provenance mismatch – one scenario single-fidelity and the other multi-fidelity, or two multi-fidelity runs with different stack shapes / final levels – as a `warning` (default) or, with `strict_fidelity = TRUE`, a hard error. Matched or both-single-fidelity provenance passes silently. The check is forward-compatible: manifests from PESTO versions that do not populate the slot read as `NULL` and pass.

References

Fawkes, J., Hu, R., Evans, R. J., & Sejdinovic, D. (2024). Doubly robust kernel statistics for testing distributional treatment effects. *Transactions on Machine Learning Research*.

See Also

`dr_date_test()` for the underlying observational-causal test; `PESTO::pesto_ensemble_manifest` for the input contract; `PESTO::pesto_ies_callback()` for producing the ensembles upstream.

Other distributional treatment effects: `dr_date_test()`, `dr_dett_test()`

Examples

```
# Requires PESTO (>= 0.4.1) -- wired through Imports.
library(PESTO)
npar <- 2L; nobs <- 4L; nreal <- 60L
G  <- matrix(stats::rnorm(nobs * npar), nobs, npar)
y0 <- as.numeric(G %*% c(1.0, -0.5)) + stats::rnorm(nobs, sd = 0.05)
y1 <- y0 + c(0.6, 0.6, 0.6, 0.6)   # intervention shifts outputs
names(y0) <- names(y1) <- paste0("o", seq_len(nobs))

prior <- matrix(stats::rnorm(nreal * npar), nreal, npar,
                dimnames = list(NULL, c("p1", "p2")))
```

```
fit0 <- pesto_ies_callback(function(t) t %*% t(G), prior, y0, 0.05,
                           noptmax = 3, verbose = FALSE)
fit1 <- pesto_ies_callback(function(t) t %*% t(G), prior, y1, 0.05,
                           noptmax = 3, verbose = FALSE)
m_base <- as_manifest(fit0, run_id = "baseline")
m_intv <- as_manifest(fit1, run_id = "intervention")
res <- dr_date_scenario(m_base, m_intv,
                        n_permutations = 200L, seed = 1L)
print(res)
```

`dr_date_test` *Doubly Robust Distributional Average Treatment Effect Test (DR-DATE)*

Description

Tests whether the distributions of potential outcomes Y(1) and Y(0) differ using a doubly robust kernel MMD statistic. Detects distributional effects (variance, shape) that mean-based tests miss.

Usage

```
dr_date_test(
  y,
  treatment,
  covariates,
  kernel_y = kernel_spec(),
  propensity_model = c("logistic", "ranger", "xgboost"),
  outcome_model = c("krr", "zero"),
  cross_fit = TRUE,
  n_folds = 5L,
  n_permutations = 500L,
  n_bins = 10L,
  regularisation = "cv",
  min_ess_fraction = 0.1,
  alpha = 0.05,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`y`	Numeric vector or matrix. Outcome variable.
`treatment`	Binary vector (0/1). Treatment indicator.
`covariates`	Numeric matrix, data.frame, or data.table. Confounders.
`kernel_y`	Kernel specification for outcome space. Default is RBF.

`propensity_model`
: Character. Method for propensity estimation: `"logistic"` (default), `"ranger"`, or `"xgboost"`.

`outcome_model`
: Character. `"krr"` (kernel ridge regression, default) fits a conditional mean embedding for each arm and forms the doubly robust statistic; `"zero"` drops the outcome model and returns the inverse-probability-weighted (singly robust) statistic.

`cross_fit`
: Logical. If `TRUE` (default), both nuisances – the propensity score and the conditional mean embedding – are estimated by `n_folds`-fold cross-fitting and evaluated out-of-fold, as the doubly robust theory requires under flexible nuisance estimators. If `FALSE`, both are fit in-sample (faster, but the test can be anti-conservative).

`n_folds`
: Integer. Number of cross-fitting folds. Default is 5.

`n_permutations`
: Integer. Number of permutations. Default is 500.

`n_bins`
: Integer. Propensity score bins for permutation. Default is 10.

`regularisation`
: Numeric or `"cv"`. Ridge parameter for the CME. Default is `"cv"`.

`min_ess_fraction`
: Numeric in (0, 1). If the effective sample size of either arm's inverse-probability weights falls below this fraction of `n`, a reliability `warning()` is emitted. Default 0.1.

`alpha`
: Numeric. Significance level. Default is 0.05.

`seed`
: Integer or `NULL`. Random seed. Permutations are drawn through R's RNG, so a fixed `seed` makes the test fully reproducible.

`verbose`
: Logical. Print progress. Default is `FALSE`.

Details

The DR-DATE test (Fawkes, Hu, Evans & Sejdinovic, 2024) constructs doubly robust (augmented inverse-probability-weighted) estimators for the counterfactual mean embeddings of Y(1) and Y(0) in a reproducing kernel Hilbert space. For arm a the augmented embedding is

$$\hat{\mu}_a = \frac{1}{n}\sum_i \tilde{w}_{a,i}\big(k(y_i,\cdot) - \hat{m}_a(x_i)\big) + \hat{m}_a(x_i),$$

where $\hat{m}_a$ is the conditional mean embedding fitted on arm a and $\tilde{w}_{a,i}$ are stabilised inverse-probability weights. The statistic is $\|\hat{\mu}_1 - \hat{\mu}_0\|^2$ in the RKHS. Setting `outcome_model = "zero"` sets $\hat{m}_a \equiv 0$ and recovers the inverse-probability-weighted statistic.

Double robustness: the test is consistent if *either* the propensity model *or* the outcome (CME) model is correctly specified. Cross-fitting (`cross_fit = TRUE`) makes this hold under flexible machine-learning nuisances by removing own-observation overfitting bias (Chernozhukov et al., 2018).

Permutation null: the reference distribution permutes treatment labels *within* propensity-score bins, holding the fitted nuisances fixed. This is valid under within-bin exchangeability of treatment given the (binned) propensity score; calibration degrades as the bins coarsen relative to the propensity variation inside them.

Key advantage: unlike DML or TMLE which test only for mean shifts, DR-DATE detects *any* distributional difference including changes in variance, skewness, or shape.

Value

An object of class "kernel_test_result". The ess element holds the smaller of the two per-arm effective sample sizes and ess_warning records whether the reliability floor was hit.

References

Fawkes, J., Hu, R., Evans, R. J., & Sejdinovic, D. (2024). Doubly robust kernel statistics for testing distributional treatment effects. *Transactions on Machine Learning Research*.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., & Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1), C1-C68.

See Also

Other distributional treatment effects: dr_date_scenario(), dr_dett_test()

Examples

```
set.seed(42)
n <- 300
x <- matrix(rnorm(n * 2), n, 2)
logit_p <- 0.5 * x[, 1]
t <- rbinom(n, 1, plogis(logit_p))
y <- t * 1.0 + x[, 1] + rnorm(n, sd = 0.5)

result <- dr_date_test(y, t, x, n_permutations = 200, seed = 1)
print(result)
```

---

dr_dett_test *Doubly Robust Distributional Effect on the Treated Test (DR-DETT)*

Description

Tests whether the distribution of the treated potential outcome Y(1) differs from the control potential outcome Y(0) among the treated subpopulation. Requires only one-sided overlap.

Usage

```
dr_dett_test(
  y,
  treatment,
  covariates,
  kernel_y = kernel_spec(),
  propensity_model = c("logistic", "ranger", "xgboost"),
  outcome_model = c("krr", "zero"),
  cross_fit = TRUE,
```

```
  n_folds = 5L,
  n_permutations = 500L,
  n_bins = 10L,
  regularisation = "cv",
  min_ess_fraction = 0.1,
  alpha = 0.05,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

y	Numeric vector or matrix. Outcome variable.
treatment	Binary vector (0/1). Treatment indicator.
covariates	Numeric matrix, data.frame, or data.table. Confounders.
kernel_y	Kernel specification for outcome space. Default is RBF.
propensity_model	Character. Method for propensity estimation: "logistic" (default), "ranger", or "xgboost".
outcome_model	Character. "krr" (kernel ridge regression, default) fits a conditional mean embedding for each arm and forms the doubly robust statistic; "zero" drops the outcome model and returns the inverse-probability-weighted (singly robust) statistic.
cross_fit	Logical. If TRUE (default), both nuisances – the propensity score and the conditional mean embedding – are estimated by n_folds-fold cross-fitting and evaluated out-of-fold, as the doubly robust theory requires under flexible nuisance estimators. If FALSE, both are fit in-sample (faster, but the test can be anti-conservative).
n_folds	Integer. Number of cross-fitting folds. Default is 5.
n_permutations	Integer. Number of permutations. Default is 500.
n_bins	Integer. Propensity score bins for permutation. Default is 10.
regularisation	Numeric or "cv". Ridge parameter for the CME. Default is "cv".
min_ess_fraction	Numeric in (0, 1). If the effective sample size of either arm's inverse-probability weights falls below this fraction of n, a reliability warning() is emitted. Default 0.1.
alpha	Numeric. Significance level. Default is 0.05.
seed	Integer or NULL. Random seed. Permutations are drawn through R's RNG, so a fixed seed makes the test fully reproducible.
verbose	Logical. Print progress. Default is FALSE.

Details

DR-DETT is analogous to DR-DATE but focuses on the **effect on the treated** (ETT) rather than the average treatment effect. The treated counterfactual Y(1) | T = 1 is observed directly, so only

the *control* arm needs an outcome model. The control counterfactual Y(0) | T = 1 is reconstructed by augmented inverse-probability weighting, reweighting controls by the treatment odds $e(x)/(1-e(x))$ so that their covariate distribution matches the treated. With `outcome_model = "krr"` the control conditional mean embedding $\hat{m}_0$ supplies the doubly robust augmentation; `outcome_model = "zero"` returns the inverse-probability-weighted statistic. It requires only one-sided overlap: P(T = 1 | X) bounded away from 0 (not necessarily from 1), so it applies where positivity fails for the control group.

Value

An object of class `"kernel_test_result"`. The `ess` element holds the effective sample size of the control reconstruction weights and `ess_warning` records whether the reliability floor was hit.

References

Fawkes, J., Hu, R., Evans, R. J., & Sejdinovic, D. (2024). Doubly robust kernel statistics for testing distributional treatment effects. *Transactions on Machine Learning Research*.

See Also

Other distributional treatment effects: `dr_date_scenario()`, `dr_date_test()`

Examples

```
set.seed(42)
n <- 300
x <- matrix(rnorm(n * 2), n, 2)
t <- rbinom(n, 1, plogis(0.5 * x[, 1]))
y <- t * rnorm(n, sd = 2) + (1 - t) * rnorm(n, sd = 1) + x[, 1]

result <- dr_dett_test(y, t, x, n_permutations = 200, seed = 1)
print(result)
```

`effective_sample_size` *Compute Effective Sample Size*

Description

Computes ESS from importance weights: ESS = (sum(w))^2 / sum(w^2).

Usage

```
effective_sample_size(w)
```

Arguments

`w`	Numeric vector of weights.

Value

Scalar effective sample size.

See Also

Other density ratio and propensity: assess_overlap(), estimate_density_ratio(), estimate_propensity(), fit_density_ratio(), plot_weights(), predict_density_ratio()

Examples

```
w <- runif(100, 0.5, 2)
effective_sample_size(w)
```

---

estimate_density_ratio

Estimate Density Ratios (backwards-compatible wrapper)

Description

Wraps fit_density_ratio() + predict_density_ratio() on the same data. Preserved for backwards compatibility with kernR 0.0.0.901x callers; new code should prefer the explicit fit/predict pair so train/test splits are honoured.

Usage

```
estimate_density_ratio(
  x,
  z,
  method = c("logistic", "ranger", "xgboost", "proxymix"),
  n_noise = 1L,
  proxymix_components = 2L,
  seed = NULL
)
```

Arguments

x	Numeric vector or matrix. Treatment variable (training).
z	Numeric matrix or data.frame. Confounders (training).
method	Character. Backend: "logistic" (default), "ranger", "xgboost", or "proxymix".
n_noise	Integer. Noise samples per real sample for classifier backends. Default 1L.
proxymix_components	Integer. Mixture components per density when method = "proxymix". Default 2L.
seed	Integer or NULL. Random seed.

Details

The return shape (`weights`, `ratios`, `ess`, `method`, `n`) is unchanged from previous versions. Internally, ratios are now computed in log-space (which fixes pathological tail behaviour that the classifier-based 0.0.0.9012 implementation occasionally showed under extreme imbalance).

Value

A list of class `density_ratio_fit_estimate` with components `weights`, `ratios`, `ess`, `method`, `n`, and `fit` (the underlying `density_ratio_fit` for callers that want diagnostics).

See Also

`fit_density_ratio()` for the fit/predict surface.

Other density ratio and propensity: `assess_overlap()`, `effective_sample_size()`, `estimate_propensity()`, `fit_density_ratio()`, `plot_weights()`, `predict_density_ratio()`

Examples

```
set.seed(42)
n <- 200
z <- matrix(rnorm(n * 2), n, 2)
x <- z[, 1] + rnorm(n)
dr <- estimate_density_ratio(x, z)
dr$ess
```

---

estimate_propensity *Estimate Propensity Scores*

Description

Estimates P(T = 1 | X) using the specified model, with built-in cross-fitting support.

Usage

```
estimate_propensity(
  treatment,
  covariates,
  method = c("logistic", "ranger", "xgboost"),
  cross_fit = TRUE,
  n_folds = 5L,
  trim = 0.01,
  seed = NULL
)
```

Arguments

`treatment`	Binary vector (0/1). Treatment indicator.
`covariates`	Numeric matrix or data.frame. Confounders.
`method`	Character. `"logistic"` (default), `"ranger"`, or `"xgboost"`.
`cross_fit`	Logical. If `TRUE`, uses 5-fold cross-fitting to produce out-of-sample propensity estimates. Default is `TRUE`.
`n_folds`	Integer. Number of cross-fitting folds. Default is 5.
`trim`	Numeric. Trim extreme propensity scores to `[trim, 1-trim]`. Default is 0.01.
`seed`	Integer or `NULL`. Random seed for the cross-fitting fold assignment, so a fixed `seed` makes cross-fitted scores reproducible.

Value

A list of class `"propensity_fit"` with components:

scores Estimated propensity scores P(T=1|X).

method Method used.

trim Trimming threshold applied.

n_trimmed Number of scores that were trimmed.

See Also

Other density ratio and propensity: `assess_overlap()`, `effective_sample_size()`, `estimate_density_ratio()`, `fit_density_ratio()`, `plot_weights()`, `predict_density_ratio()`

Examples

```
set.seed(42)
n <- 300
x <- matrix(rnorm(n * 3), n, 3)
logit_p <- 0.5 * x[, 1] - 0.3 * x[, 2]
t <- rbinom(n, 1, plogis(logit_p))
ps <- estimate_propensity(t, x)
summary(ps$scores)
```

`fit_cme` *Estimate Conditional Mean Embedding via Kernel Ridge Regression*

Description

Estimates the conditional mean embedding $\mu_{Y|X=x}$ in the RKHS using kernel ridge regression (Park, Muandet, Fukumizu & Sejdinovic, 2013; Muandet et al., 2017).

Usage

```
fit_cme(
  x,
  y,
  kernel_x = kernel_spec(),
  kernel_y = kernel_spec(),
  lambda = "cv"
)
```

Arguments

x	Numeric matrix of conditioning variables (n x d_x).
y	Numeric matrix of target variables (n x d_y).
kernel_x	Kernel specification for x.
kernel_y	Kernel specification for y.
lambda	Ridge regularisation parameter. If "cv", selected by leave-one-out cross-validation. Default is "cv".

Details

This is the lower-level building block used by kernel_downscale(). Most users should call kernel_downscale(); use fit_cme() directly when you need access to the trained operator (the weight matrix W) for custom downstream computations – e.g. constructing a kernel Bayes' rule update, plugging into a manuscript figure pipeline, or composing with other RKHS operators.

Value

A list of class "cme_fit" with components:

W Operator matrix (K_x + n lambda I)^{-1} (n x n).

Ky Kernel matrix of y.

x_train Training x data.

kernel_x, kernel_y Resolved kernel specifications.

lambda Regularisation parameter used.

References

Park, J., Muandet, K., Fukumizu, K., & Sejdinovic, D. (2013). *Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models.* IEEE Signal Processing Magazine.

Muandet, K., Fukumizu, K., Sriperumbudur, B., & Scholkopf, B. (2017). *Kernel mean embedding of distributions: A review and beyond.* Foundations and Trends in Machine Learning, 10(1-2).

See Also

predict.cme_fit(), kernel_downscale()

Other downscaling and embeddings: aggregate_downscale(), dist_regression(), kernel_downscale(), posterior_sample_aggregate()

Examples

```
set.seed(1L)
x <- matrix(rnorm(60L), ncol = 2L)
y <- matrix(x[, 1L] + rnorm(30L, sd = 0.2), ncol = 1L)
fit <- fit_cme(x, y, lambda = 1e-2)
dim(fit$W)
```

`fit_density_ratio` *Fit a Density-Ratio Model*

Description

Trains a density-ratio estimator for the do-null reweighting `w(x, z) = p*(x) / p(x | z)` used by `bd_hsic_test()`. The fitted model is decoupled from evaluation: `predict_density_ratio()` applies it to held-out rows, so train/test splits are honoured cleanly.

Usage

```
fit_density_ratio(
  x,
  z,
  method = c("logistic", "ranger", "xgboost", "proxymix"),
  n_noise = 1L,
  proxymix_components = 2L,
  seed = NULL
)
```

Arguments

`x`	Numeric vector or matrix. Treatment variable (training).
`z`	Numeric matrix or data.frame. Confounders (training).
`method`	Character. Backend: `"logistic"` (default), `"ranger"`, `"xgboost"`, or `"proxymix"`.
`n_noise`	Integer. Noise samples per real sample for classifier backends. Default `1L`.
`proxymix_components`	Integer. Mixture components per density when `method = "proxymix"`. Default `2L`.
`seed`	Integer or `NULL`. Random seed.

Details

Four backends are supported (the `method` argument):

- `"logistic"` (default), `"ranger"`, `"xgboost"` – classifier-based noise-contrastive estimation. The classifier is trained to distinguish joint samples `(x, z)` from product-of-marginals samples `(x_perm, z)`; the density ratio is recovered from the calibrated class probabilities. Log-ratios are stored internally for numerical stability.

- "proxymix" – Gaussian-mixture density-ratio. Fits one GMM to the joint sample cloud (x, z) and one to a permuted product-of-marginals cloud via proxymix::fit_proxymix(regime = "sample"); ratios are evaluated in log-space from proxymix::dgmm(). Per-GMM convergence diagnostics (BIC, AIC, final log-likelihood, iteration count) are surfaced on the returned fit; query them via fit$diagnostics.

Introduced in kernR 0.0.0.9014 to close the documented-but- unimplemented sample-split gap in bd_hsic_test() (see NEWS). estimate_density_ratio() is now a thin backwards-compatible wrapper that fits and predicts on the same data.

Value

An object of class density_ratio_fit (plus density_ratio_fit_<method> as the dispatch class). Carries: method, the backend-specific fit (model for classifiers; fit_joint + fit_marg for proxymix), diagnostics, n_train, ncol_x, ncol_z, seed.

See Also

predict_density_ratio(), estimate_density_ratio(), bd_hsic_test().

Other density ratio and propensity: assess_overlap(), effective_sample_size(), estimate_density_ratio(), estimate_propensity(), plot_weights(), predict_density_ratio()

Examples

```
set.seed(1L)
n <- 200L
z <- matrix(rnorm(n * 2L), n, 2L)
x <- z[, 1L] + rnorm(n)
fit <- fit_density_ratio(x, z, method = "logistic", seed = 1L)
fit$diagnostics
```

---

gaussian_score *Score function for a multivariate normal target*

Description

Builds a score function – the gradient of the log density, $\nabla_x \log p(x) = -\Sigma^{-1}(x - \mu)$ – for a multivariate normal target, suitable for the score argument of ksd_test().

Usage

```
gaussian_score(mean = NULL, sigma = NULL)
```

Arguments

mean	Numeric vector of length d, the target mean. NULL (default) uses the zero vector of the sample dimension.
sigma	Numeric d x d covariance matrix. NULL (default) uses the identity matrix of the sample dimension. Must be symmetric and invertible.

Details

The returned closure accepts the sample matrix and returns the score evaluated row-wise. Leaving `mean` or `sigma` at `NULL` defaults them to the zero vector and the identity matrix of the dimension seen at call time, so `gaussian_score()` with no arguments is the standard-normal score in any dimension.

Value

A function of one argument (an `n x d` numeric matrix) returning the `n x d` matrix of scores. The mean and covariance are captured by the closure.

Author(s)

Max Moldovan, <max.moldovan@adelaide.edu.au>

References

Liu, Q., Lee, J. D., & Jordan, M. I. (2016). A kernelized Stein discrepancy for goodness-of-fit tests. *Proceedings of the 33rd International Conference on Machine Learning*, PMLR 48, 276-284.

See Also

`ksd_test()`

Other goodness-of-fit tests: `concordance_test()`, `concordance_test_nystrom()`, `coverage_test()`, `ksd_test()`, `ksd_test_nystrom()`, `numeric_score()`

Examples

```
set.seed(1)
x <- matrix(stats::rnorm(200L), ncol = 2L)

# Standard-normal target in two dimensions
s0 <- gaussian_score()
str(s0(x))

# Correlated-normal target
sig <- matrix(c(1, 0.5, 0.5, 1), nrow = 2L)
s1 <- gaussian_score(mean = c(0, 0), sigma = sig)
```

---

hierarchical_test *Hierarchical Kernel Causal Test*

Description

Extends bd-HSIC and DR-DATE/DR-DETT to hierarchical (nested/clustered) data by decomposing the test statistic into within-cluster and between-cluster components.

Usage

```
hierarchical_test(
  y,
  treatment,
  covariates,
  cluster_id,
  method = c("dr-date", "dr-dett", "bd-hsic"),
  kernel_y = kernel_spec(),
  n_permutations = 500L,
  weight_method = c("equal", "icc", "within_only"),
  seed = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

y	Numeric vector or matrix. Outcome.
treatment	Treatment variable (binary for DR tests, any for bd-HSIC).
covariates	Numeric matrix of confounders.
cluster_id	Factor or integer vector identifying clusters.
method	Character. "dr-date" (default), "dr-dett", or "bd-hsic".
kernel_y	Kernel specification for outcomes.
n_permutations	Integer. Number of permutations. Default is 500.
weight_method	Character. How to weight within/between components: "equal" (default), "icc" (variance decomposition), or "within_only".
seed	Integer or NULL.
verbose	Logical.
...	Additional arguments passed to the underlying test. Do not pass cross_fit or n_folds: within-cluster sub-tests are always fit in-sample, as the top-level within-cluster permutation supplies the calibration.

Details

For clustered data (e.g., patients within hospitals, plots within farms), standard kernel tests may have inflated type I error because observations within the same cluster are not independent.

This function decomposes the test into:

- **Within-cluster**: Average of within-cluster test statistics (tests for treatment effects within each cluster).
- **Between-cluster**: Test on cluster-level mean embeddings (tests for treatment effects across clusters).

The combined statistic is a weighted sum, with weights determined by weight_method. Permutation is performed within clusters to preserve the hierarchical structure.

Value

An object of class "kernel_test_result" with additional hierarchical component containing within/between statistics.

See Also

Other causal association tests: bd_hsic_test(), kernel_causal_test()

Examples

```
set.seed(42)
n_clusters <- 20
n_per <- 30
n <- n_clusters * n_per
cluster_id <- rep(1:n_clusters, each = n_per)

# Cluster-level random effects
cluster_effect <- rnorm(n_clusters, sd = 1)[cluster_id]
x <- matrix(rnorm(n * 2), n, 2)
t <- rbinom(n, 1, plogis(0.3 * x[, 1]))
y <- 0.5 * t + cluster_effect + x[, 1] + rnorm(n)

result <- hierarchical_test(y, t, x, cluster_id,
  method = "dr-date",
  n_permutations = 100,
  seed = 1
)
print(result)
```

---

hsic_identifiability *HSIC-Based Identifiability Diagnostic*

Description

Pre-PESTO (or pre-IES) screening: for each parameter theta[, j] and each output y[, k], computes an HSIC permutation test of independence and flags parameters with no detectable association to any output as unidentifiable. Useful for trimming the parameter space before ensemble-smoother calibration of mechanistic ag-system models such as APSIM.

Usage

```
hsic_identifiability(
  theta,
  y,
  alpha = 0.05,
  p_adjust = c("BH", "holm", "hochberg", "bonferroni", "BY", "fdr", "none"),
```

```
  n_permutations = 500L,
  kernel_theta = kernel_spec(),
  kernel_y = kernel_spec(),
  seed = NULL
)
```

Arguments

theta	Numeric matrix n x p of parameter design points (one row per simulator run, one column per parameter). Vectors are coerced via as.matrix().
y	Numeric matrix n x q of simulator outputs. Vectors are coerced via as.matrix().
alpha	Numeric in (0, 1). Identifiability threshold on the adjusted minimum p-value. Default 0.05.
p_adjust	Character. Across-grid p-value adjustment method passed to stats::p.adjust(). Default "BH". Use "none" to disable.
n_permutations	Integer. Permutations per HSIC test. Default 500.
kernel_theta	A kernel_spec() for parameter columns. Default RBF with per-column median heuristic.
kernel_y	A kernel_spec() for output columns. Default RBF with per-column median heuristic.
seed	Integer or NULL. Random seed for reproducibility.

Details

Kernel matrices are computed once per parameter and once per output, so the total cost is O((p + q) n^2) (kernel construction) plus O(p q n_permutations n^2) (permutation null), where p is the number of parameters, q the number of outputs and n the design size.

A parameter is **identifiable** at level alpha when its smallest (optionally adjusted) p-value across outputs satisfies min_p <= alpha. Across-grid p-value adjustment defaults to Benjamini-Hochberg, which is the natural FDR control for screening applications.

Value

An object of class "hsic_identifiability" with components:

statistic p x q matrix of HSIC statistics.

p_value p x q matrix of raw permutation p-values.

p_value_adjusted p x q matrix of adjusted p-values (same as p_value when p_adjust = "none").

max_statistic Length-p vector: per-parameter maximum HSIC across outputs.

min_p_value Length-p vector: per-parameter minimum adjusted p-value across outputs.

identifiable Length-p logical: min_p_value <= alpha.

rank Parameter indices ordered by descending max_statistic.

alpha, p_adjust, n, n_permutations Inputs / metadata.

param_names, output_names Character vectors.

call The matched call.

References

Gretton, A., Fukumizu, K., Teo, C. H., Song, L., Scholkopf, B., & Smola, A. J. (2008). A kernel statistical test of independence. *NeurIPS*, 20.

Da Veiga, S. (2015). Global sensitivity analysis with dependence measures. *Journal of Statistical Computation and Simulation*, 85(7), 1283-1305.

See Also

lhs_design(), hsic_test()

Other sensitivity and identifiability: hsic_sensitivity(), lhs_design()

Examples

```
set.seed(1)
n <- 60
# 3 active parameters + 1 inert
theta <- matrix(stats::runif(n * 4), nrow = n,
                dimnames = list(NULL, paste0("p", 1:4)))
y1 <- theta[, 1] + 0.5 * theta[, 2]^2 + stats::rnorm(n, sd = 0.1)
y2 <- sin(2 * pi * theta[, 3]) + stats::rnorm(n, sd = 0.1)
y <- cbind(yield = y1, biomass = y2)
fit <- hsic_identifiability(theta, y, n_permutations = 199, seed = 1)
print(fit)
```

---

hsic_sensitivity *HSIC-Based Distributional Sensitivity Index*

Description

Computes HSIC-based sensitivity indices (Da Veiga, 2015) for each input parameter against each output. By default returns first-order indices only; `total_order = TRUE` adds the complementary total-order decomposition.

Usage

```
hsic_sensitivity(
  theta,
  y,
  total_order = FALSE,
  p_value = TRUE,
  n_permutations = 500L,
  total_order_ci = FALSE,
  n_bootstrap = 200L,
  ci_level = 0.95,
  total_order_test = c("none", "cond_perm"),
```

```
  n_clusters_cp = "auto",
  p_adjust = c("BH", "holm", "hochberg", "bonferroni", "BY", "fdr", "none"),
  kernel_theta = kernel_spec(),
  kernel_y = kernel_spec(),
  seed = NULL,
  total_order_p_value = NULL
)
```

Arguments

theta	Numeric matrix `n x p` of input draws (one row per simulator run, one column per parameter). Vectors are coerced via `as.matrix()`.
y	Numeric matrix `n x q` of simulator outputs. Vectors are coerced via `as.matrix()`.
total_order	Logical. If `TRUE`, additionally compute the total-order HSIC-Sensitivity Index via Da Veiga's complement formulation. Requires `p >= 2`. Default `FALSE` (backwards- compatible).
p_value	Logical. If `TRUE` (default), compute permutation p-values and BH-adjusted p-values for *first-order* indices. Set `FALSE` to skip the permutation null. Has no effect on total-order indices (see Details).
n_permutations	Integer. Permutations per HSIC test. Default `500`.
total_order_ci	Logical. If `TRUE`, compute pair-bootstrap percentile CIs for the total-order indices. Requires `total_order = TRUE`. Default `FALSE` (backwards-compatible). Replaces the misleading 0.0.0.9012 `total_order_p_value` argument (removed in 0.0.0.9013; see Details).
n_bootstrap	Integer. Pair-bootstrap resamples used for the total-order CI. Default `200`. Only consulted when `total_order_ci = TRUE`.
ci_level	Numeric in `(0, 1)`. Two-sided percentile CI level for the total-order bootstrap. Default `0.95`. Only consulted when `total_order_ci = TRUE`.
total_order_test	Character. `"none"` (default; backwards- compatible) or `"cond_perm"` (since 0.0.0.9014). The latter activates a conditional-permutation test for `H_0: X_j _||_ Y | X_{~j}` and populates `p_value_total_order` and `p_value_total_order_adjusted`. Requires `total_order = TRUE`. Different from (and replaces) the retracted 0.0.0.9012 pair-bootstrap method.
n_clusters_cp	Integer or `"auto"`. Number of conditioning bins for the conditional-permutation test (k-means on `X_{~j}`). `"auto"` chooses `min(floor(n / 5), 20)`. Only consulted when `total_order_test = "cond_perm"`.
p_adjust	Character. Across-grid p-value adjustment method passed to `stats::p.adjust()`. Default `"BH"`.
kernel_theta	A `kernel_spec()` for parameter columns. Default RBF with median heuristic. Used for both first-order (per-column) and total-order (per `X_{~j}` subset).
kernel_y	A `kernel_spec()` for output columns. Default RBF with per-column median heuristic.
seed	Integer or `NULL`. Random seed for reproducibility.
total_order_p_value	Defunct as of 0.0.0.9013. Passing any non-`NULL` value errors with a pointer to `total_order_test` and `total_order_ci`.

Details

First-order index (always computed):

$$S_j^{HSIC} = \frac{HSIC(X_j, Y)}{\sqrt{HSIC(X_j, X_j) \cdot HSIC(Y, Y)}}$$

is the direct contribution of `X_j` to `Y` – analogous to (but not equal to) the Sobol first-order index.

Total-order index (when `total_order = TRUE`):

$$T_j^{HSIC} = 1 - \frac{HSIC(X_{\sim j}, Y)}{\sqrt{HSIC(X_{\sim j}, X_{\sim j}) \cdot HSIC(Y, Y)}}$$

where `X_{~j}` is the parameter design with column `j` removed. By construction the difference `T_j - S_j` captures the contribution of `X_j` *through interactions* with other parameters. For purely additive models `T_j = S_j`; in the presence of interaction `T_j > S_j`.

Unlike variance-based Sobol indices, both versions of the HSIC-Sensitivity Index capture non-linear and distributional effects: a parameter that affects the variance, skewness, or tail of `Y` without shifting its mean is invisible to Sobol but visible to HSIC. The normalisation bounds each index to `[0, 1]`.

Optional permutation p-values are computed for first-order indices (Benjamini-Hochberg-adjusted across the grid by default).

Total-order uncertainty quantification (`total_order_ci = TRUE`, since 0.0.0.9013). A pair-bootstrap percentile CI on the index `T_j` itself: resample `(theta, y)` pairs with replacement `n_bootstrap` times, recompute `T_j` on each resample, report a `ci_level` (default 95%) two-sided percentile interval. This is uncertainty quantification, not a hypothesis test.

Total-order conditional-permutation significance test (`total_order_test = "cond_perm"`, since 0.0.0.9014). Tests `H_0: X_j _||_ Y | X_{~j}` – under the null, `X_j` adds nothing beyond what is already in `X_{~j}` and `T_j` is concentrated at zero. The null is generated by **conditional permutation**: k-means-cluster the design points by `X_{~j}` similarity, then within each cluster permute `Y`; recompute `T_j` on each permuted design. p-value = `(1 + #{T_perm >= T_obs}) / (1 + B)`. The `p_adjust` method applies grid-wide.

The 0.0.0.9012 `total_order_p_value` pair-bootstrap implementation was **not** this – it sampled the empirical joint, not a null-of-no-effect, and was retracted in 0.0.0.9013. The 0.0.0.9014 conditional-permutation test repopulates `p_value_total_order` with a properly-calibrated value; the `total_order_test` flag on the result distinguishes the new mode from the retracted one.

Power caveat (empirical calibration 2026-05-16). Repeated-seed calibration (`orchestra_calibration_20260516.R`, B = 100, N = 80) confirms the cond_perm test is null-calibrated (per-parameter type-I rates 0.01-0.04 at nominal alpha = 0.05) but **conservative on additive designs** (rejection rate ~3% on a strong-additive design where the first-order test rejects above 90%). The reason is structural: on additive `Y = sum_j f_j(X_j)`, the conditional permutation of `Y` within `X_{~j}` bins preserves all of `X_{~j}`'s contribution and randomises only the `X_j` component (which is independent of `X_{~j}`), so the permuted `HSIC(X_{~j}, Y_perm)` is statistically close to the observed `HSIC(X_{~j}, Y)`. Use cond_perm for **interaction detection**; use the first-order permutation `p_value` (always computed when `p_value = TRUE`) for **additive contributions**. The two are complementary, not interchangeable.

Total-order CIs cost `B * p * q` kernel re-evaluations on resampled designs; set `n_bootstrap` accordingly. Use the CI for honest uncertainty bars on the index magnitude; do not interpret it as a significance verdict.

Value

An object of class "hsic_sensitivity" with components:

index p x q matrix of first-order HSIC-Sensitivity Indices in [0, 1].

statistic p x q matrix of raw first-order HSIC statistics.

p_value p x q matrix of raw permutation p-values (first-order), or NULL if p_value = FALSE.

p_value_adjusted p x q matrix of adjusted p-values (first-order), or NULL.

index_total_order p x q matrix of total-order indices, or NULL when total_order = FALSE.

statistic_total_order p x q matrix of HSIC(X_{~j}, Y_k) statistics, or NULL.

total_order Logical flag: was total-order requested?

total_index Length-p vector: per-parameter maximum *first-order* index across outputs (used for ranking; kept under this historical name for backwards compatibility).

rank Parameter indices ordered by descending total_index.

n, n_permutations, param_names, output_names, p_adjust Metadata.

call The matched call.

References

Da Veiga, S. (2015). Global sensitivity analysis with dependence measures. *Journal of Statistical Computation and Simulation*, 85(7), 1283-1305.

Gretton, A., Bousquet, O., Smola, A., & Scholkopf, B. (2005). Measuring statistical dependence with Hilbert-Schmidt norms. *ALT*, 63-77.

See Also

hsic_identifiability(), hsic_test()

Other sensitivity and identifiability: hsic_identifiability(), lhs_design()

Examples

```
set.seed(1)
n <- 80L
theta <- matrix(stats::runif(n * 3L), nrow = n,
                dimnames = list(NULL, c("active", "weak", "inert")))
y <- 2 * theta[, "active"] +
     0.5 * theta[, "weak"] +
     stats::rnorm(n, sd = 0.1)
fit <- hsic_sensitivity(theta, y, total_order = TRUE,
                        n_permutations = 199L, seed = 1L)
fit
```

`hsic_test` *HSIC Independence Test*

Description

Tests whether two variables are independent using the Hilbert-Schmidt Independence Criterion (HSIC). Uses a permutation test for inference.

Usage

```
hsic_test(
  x,
  y,
  kernel_x = kernel_spec(),
  kernel_y = kernel_spec(),
  n_permutations = 500L,
  alpha = 0.05,
  seed = NULL
)
```

Arguments

`x`	Numeric vector, matrix, or data.frame. First variable.
`y`	Numeric vector, matrix, or data.frame. Second variable.
`kernel_x`	Kernel specification for `x`. Default is RBF with median heuristic.
`kernel_y`	Kernel specification for `y`. Default is RBF with median heuristic.
`n_permutations`	Integer. Number of permutations for the null distribution. Default is 500.
`alpha`	Numeric. Significance level. Default is 0.05.
`seed`	Integer or `NULL`. Random seed for reproducibility.

Value

An object of class `"kernel_test_result"` with components:

statistic The observed HSIC test statistic.

p_value Permutation p-value.

method `"HSIC"`.

n Sample size.

n_permutations Number of permutations used.

null_distribution Vector of permuted HSIC values.

kernel_x, kernel_y Kernel specifications used (with resolved bandwidths).

call The matched call.

References

Gretton, A., Fukumizu, K., Teo, C. H., Song, L., Scholkopf, B., & Smola, A. J. (2008). A kernel statistical test of independence. *NeurIPS*, 20.

See Also

Other independence and two-sample tests: `mmd_test()`

Examples

```
set.seed(42)
n <- 200
x <- rnorm(n)

# Dependent case
y_dep <- x^2 + rnorm(n, sd = 0.5)
result <- hsic_test(x, y_dep)
print(result)

# Independent case
y_ind <- rnorm(n)
result <- hsic_test(x, y_ind)
print(result)
```

`hsic_test_nystrom` *HSIC Independence Test via Low-Rank Factorisation*

Description

Accelerated HSIC test using either Nystrom (default) or random Fourier features (RFF) factorisations of the input kernel matrices. For large `n` this scales as `O(n m)` per permutation instead of `O(n^2)`, with `m << n` controlling the speed / accuracy trade-off.

Usage

```
hsic_test_nystrom(
  x,
  y,
  kernel_x = kernel_spec(),
  kernel_y = kernel_spec(),
  method = c("nystrom", "rff"),
  m = 100L,
  m_x = NULL,
  m_y = NULL,
  n_permutations = 500L,
  alpha = 0.05,
  seed = NULL,
```

```
  regularise = 1e-06
)
```

Arguments

x, y	Numeric matrices (or vectors). Same number of rows.
kernel_x, kernel_y	kernel_spec()s for the two factors.
method	Character. "nystrom" (default) or "rff".
m	Integer. Rank used for the approximation (number of Nystrom landmarks or RFF features). Used for both factors unless m_x / m_y are supplied.
m_x, m_y	Optional integers overriding m per factor.
n_permutations	Integer. Default 500L.
alpha	Numeric in (0, 1). Default 0.05.
seed	Integer or NULL.
regularise	Ridge for Nystrom Cholesky (ignored under method = "rff"). Default 1e-6.

Details

The test uses the *biased* HSIC estimator $(1/n^2)\mathrm{tr}(HK_xHK_y)$, which is the form that factorises cleanly through low-rank approximations. The bias is O(1/n) and negligible in the large-n regime where Nystrom / RFF are useful.

The permutation null is built by row-permuting the (centred) y factor; per-permutation cost is O(n m_x m_y).

Value

An object of class "kernel_test_result" with the standard fields (statistic, p_value, method, n, n_permutations, null_distribution, kernel_x, kernel_y, call) plus:

approximation "nystrom" or "rff".

m_x, m_y Effective ranks used.

References

Williams, C. K. I., & Seeger, M. (2001). Using the Nystrom method to speed up kernel machines. *NeurIPS*, 13.

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *NeurIPS*, 20.

Gretton, A., Bousquet, O., Smola, A., & Scholkopf, B. (2005). Measuring statistical dependence with Hilbert-Schmidt norms. *ALT*, 63-77.

See Also

hsic_test(), nystrom_factor(), rff_features()

Other low-rank acceleration: concordance_test_nystrom(), ksd_test_nystrom(), nystrom_factor(), rff_features()

Examples

```
set.seed(1)
n <- 1500L
x <- stats::rnorm(n)
y <- x^2 + stats::rnorm(n, sd = 0.5)
fit <- hsic_test_nystrom(x, y, m = 80L,
                         n_permutations = 99L, seed = 1L)
fit
```

`kernel_causal_test` *Unified Kernel Causal Test*

Description

A convenience wrapper that dispatches to the appropriate test function based on the `method` argument.

Usage

```
kernel_causal_test(
  formula,
  data,
  method = c("dr-date", "dr-dett", "bd-hsic"),
  ...
)
```

Arguments

`formula`	A formula of the form `y ~ treatment | confounders`.
`data`	A data.frame or data.table.
`method`	Character. Test method: `"dr-date"` (default), `"dr-dett"`, or `"bd-hsic"`.
`...`	Additional arguments passed to the specific test function.

Value

An object of class `"kernel_test_result"`.

See Also

Other causal association tests: `bd_hsic_test()`, `hierarchical_test()`

Examples

```
set.seed(42)
n <- 200
dat <- data.frame(
  y = rnorm(n),
  treatment = rbinom(n, 1, 0.5),
  x1 = rnorm(n),
  x2 = rnorm(n)
)
dat$y <- dat$y + 0.5 * dat$treatment + 0.3 * dat$x1

result <- kernel_causal_test(y ~ treatment | x1 + x2,
  data = dat, method = "dr-date",
  n_permutations = 100, seed = 1
)
print(result)
```

---

kernel_downscale *Kernel-Based Statistical Downscaling*

Description

Predicts fine-resolution outputs at new coarse-resolution inputs via conditional mean embedding (CME) regression in an RKHS. Given paired coarse-fine training data (`coarse`, `fine`), fits the operator $E[Y \mid X = x]$ in closed form via kernel ridge regression and returns predictions at `new_coarse`. This is the Park-Muandet-Fukumizu-Sejdinovic conditional-mean-embedding scheme, specialised to the regression form needed for spatial / temporal downscaling.

Usage

```
kernel_downscale(
  coarse,
  fine,
  new_coarse,
  kernel_coarse = kernel_spec(),
  kernel_fine = kernel_spec(),
  lambda = "cv",
  return_weights = FALSE
)
```

Arguments

`coarse`	Numeric matrix `n x d_coarse` of training coarse-resolution inputs. Vectors are coerced via `as.matrix()`.
`fine`	Numeric matrix `n x d_fine` of training fine-resolution outputs. Multivariate outputs (`d_fine` > 1) are supported and predicted jointly.

`new_coarse` Numeric matrix `n_new x d_coarse` of coarse inputs at which to predict the fine outputs.

`kernel_coarse, kernel_fine`
`kernel_spec()` for the coarse and fine spaces. Defaults to RBF with median heuristic. `kernel_fine` is used only for the bandwidth-CV step; predictions are returned in the original `fine` units.

`lambda` Ridge regularisation parameter for the CME ridge regression. If `"cv"` (default), selected by leave-one-out cross-validation over `10^seq(-6, 1, length.out = 15)`.

`return_weights` Logical. If `TRUE`, the result carries the `n_new x n_train` weight matrix used to combine training fine values. Default `FALSE` (saves memory for large designs).

Details

Typical ag-systems use: coarse climate-grid inputs (e.g. monthly temperature, rainfall on a 25 km grid) -> fine-resolution outputs (paddock yield, biomass) at the same time index. Train on years where both coarse and fine are observed; predict fine outputs at new coarse inputs.

Compared to a linear regression baseline, the kernel approach captures non-linear coarse-fine relationships without specifying the functional form. Compared to deep-learning downscalers, it has a closed-form solution, uses orders-of-magnitude less data, and carries an interpretable kernel-bandwidth degrees-of-freedom knob.

Value

An object of class `"kernel_downscale"` with components:

prediction `n_new x d_fine` matrix of predicted fine outputs at `new_coarse`.

n_train Number of training pairs used.

n_new Number of prediction points.

lambda Regularisation used (CV-selected when `lambda = "cv"`).

kernel_coarse, kernel_fine Resolved kernel specs.

weights Optional `n_new x n_train` weight matrix.

call The matched call.

References

Park, J., Muandet, K., Fukumizu, K., & Sejdinovic, D. (2013). *Kernel embeddings of conditional distributions.* IEEE Signal Processing Magazine.

Muandet, K., Fukumizu, K., Sriperumbudur, B., & Scholkopf, B. (2017). *Kernel mean embedding of distributions: A review and beyond.* Foundations and Trends in Machine Learning, 10(1-2).

See Also

`fit_cme()`, `dist_regression()` (for downscaling when each coarse "input" is a bag of points rather than a single vector).

Other downscaling and embeddings: `aggregate_downscale()`, `dist_regression()`, `fit_cme()`, `posterior_sample_aggregate()`

Examples

```
set.seed(1)
n <- 80L
coarse <- matrix(stats::rnorm(n * 2L), n, 2L,
                 dimnames = list(NULL, c("temp", "rainfall")))
fine <- cbind(
  yield   = 2 * coarse[, "rainfall"] -
            0.5 * coarse[, "temp"]^2 +
            stats::rnorm(n, sd = 0.2),
  biomass = coarse[, "temp"] + coarse[, "rainfall"]^2 +
            stats::rnorm(n, sd = 0.2)
)
# Predict at a held-out grid
new_coarse <- matrix(stats::rnorm(20L * 2L), 20L, 2L,
                     dimnames = list(NULL, c("temp", "rainfall")))
fit <- kernel_downscale(coarse, fine, new_coarse)
print(fit)
head(fit$prediction)
```

---

kernel_matrix *Compute a Kernel Matrix*

Description

Computes the kernel (Gram) matrix between two sets of observations.

Usage

```
kernel_matrix(x, y = NULL, kernel = kernel_spec())
```

Arguments

x	Numeric matrix (n x d) or vector.
y	Numeric matrix (m x d) or vector. If NULL (default), computes the kernel matrix of x with itself.
kernel	A kernel_spec object. Default is RBF with median heuristic bandwidth.

Value

An n x m numeric matrix.

See Also

Other kernel primitives: kernel_spec(), select_bandwidth()

Examples

```
x <- matrix(rnorm(100), 50, 2)
K <- kernel_matrix(x)
dim(K)  # 50 x 50
```

`kernel_spec` *Create a Kernel Specification*

Description

Constructs a kernel specification object used throughout `kernR` for computing kernel matrices. Supports RBF (Gaussian), Matern, linear, and polynomial kernels.

Usage

```
kernel_spec(
  type = c("rbf", "matern", "linear", "polynomial"),
  bandwidth = "median",
  nu = 2.5,
  degree = 2L,
  offset = 1
)
```

Arguments

`type`	Character. Kernel type: `"rbf"` (default), `"matern"`, `"linear"`, or `"polynomial"`.
`bandwidth`	Numeric or `"median"`. Lengthscale parameter for RBF and Matern kernels. If `"median"` (default), the median heuristic is used to select bandwidth automatically from the data.
`nu`	Numeric. Smoothness parameter for the Matern kernel. Common choices: 0.5 (Laplace), 1.5, 2.5, Inf (RBF). Default is 2.5.
`degree`	Integer. Degree for polynomial kernel. Default is 2.
`offset`	Numeric. Offset for polynomial kernel. Default is 1.

Value

An object of class `"kernel_spec"`.

See Also

Other kernel primitives: `kernel_matrix()`, `select_bandwidth()`

Examples

```
# Default RBF kernel with median heuristic bandwidth
k <- kernel_spec()

# RBF with fixed bandwidth
k <- kernel_spec("rbf", bandwidth = 1.0)

# Matern kernel
k <- kernel_spec("matern", nu = 1.5)

# Linear kernel (no bandwidth needed)
k <- kernel_spec("linear")
```

`ksd_test` *Kernel Stein Discrepancy Goodness-of-Fit Test*

Description

Tests whether a sample is consistent with a target distribution `p`, where `p` is supplied through its score $\nabla_x \log p(x)$ rather than a reference sample. The statistic is the (unbiased) kernel Stein discrepancy (KSD); calibration uses a wild bootstrap of the degenerate U-statistic null. Because only the score enters, the target may be known up to an unknown normalising constant.

Usage

```
ksd_test(
  x,
  score = NULL,
  kernel = c("imq", "rbf"),
  beta = -0.5,
  bandwidth = "median",
  n_boot = 1000L,
  alpha = 0.05,
  seed = NULL,
  n_exact_max = 5000L
)
```

Arguments

`x`	Numeric vector, matrix, or data.frame. The `n x d` sample to test, one observation per row. At least five rows are required.
`score`	Either `NULL` or a function. When `NULL` (default) the target is the standard multivariate normal, with score $-x$. When a function, it must accept the `n x d` sample matrix and return the `n x d` matrix of scores $\nabla_x \log p(x)$ evaluated row-wise; see `gaussian_score()` for the multivariate-normal factory.

kernel	Character. Base kernel for the Stein kernel: "imq" (inverse multi-quadric, default) or "rbf" (Gaussian).
beta	Numeric in (-1, 0). Exponent of the IMQ kernel; ignored when kernel = "rbf". Default -0.5.
bandwidth	Numeric or "median". The IMQ offset c or the RBF bandwidth h. "median" (default) uses the median-heuristic bandwidth of the sample.
n_boot	Integer. Number of wild-bootstrap replicates for the null. Default 1000.
alpha	Numeric in (0, 1). Significance level for the verdict. Default 0.05.
seed	Integer or NULL. Random seed for reproducibility.
n_exact_max	Integer or Inf. Sample-size ceiling for the exact O(n^2) test. Above it, the call is delegated to ksd_test_nystrom() (with a message; the verdict object records approximation = "nystrom"). Inf forces the exact test at any size. Default 5000L.

Details

KSD is the score-based, one-sample complement to mmd_test(): where MMD compares two samples, KSD compares a sample against a *density*. The calibration framing is direct – given posterior or ensemble draws and the score of the distribution they claim to represent, KSD asks whether the draws actually follow that distribution. It is sensitive to mean, variance, and tail mis-specification.

The default base kernel is the inverse multi-quadric (IMQ), $k(x, y) = (c^2 + \|x - y\|^2)^\beta$ with $\beta \in (-1, 0)$. Gorham & Mackey (2017) show the IMQ Stein discrepancy detects non-convergence in regimes where the Gaussian (RBF) Stein discrepancy is blind, particularly as dimension grows; the RBF base kernel remains available via kernel = "rbf". The offset c (IMQ) and bandwidth h (RBF) default to the median heuristic over the sample.

Reproducibility: the wild-bootstrap multipliers are drawn through R's RNG, so a non-NULL seed makes the p-value reproducible under the active RNG kind (the R default Mersenne-Twister unless changed by the caller).

The exact test materialises the n x n Stein-kernel matrix, so memory and compute scale as O(n^2). To keep large samples tractable without a silent loss of exactness, a sample with more than n_exact_max rows is delegated to ksd_test_nystrom() – a low-rank approximation that is *announced* by a message and *recorded* in the returned object's approximation and m fields, so the result stays reproducible from the object. Set n_exact_max = Inf to force the exact O(n^2) test at any size, or call ksd_test_nystrom() directly to control the approximation rank m.

Value

An object of class c("ksd_test", "kernel_test_result") carrying the standard kernel_test_result fields plus:

statistic The unbiased KSD U-statistic.

p_value Upper-tail wild-bootstrap p-value (with +1 correction).

stein_kernel Base kernel used ("imq" or "rbf").

beta IMQ exponent, or NA for the RBF kernel.

bandwidth Resolved IMQ offset or RBF bandwidth.

surprise_bits Shannon-information surprise -log2(p_value).

alpha, reject Verdict level and p_value <= alpha.

Author(s)

Max Moldovan, <max.moldovan@adelaide.edu.au>

References

Liu, Q., Lee, J. D., & Jordan, M. I. (2016). A kernelized Stein discrepancy for goodness-of-fit tests. *Proceedings of the 33rd International Conference on Machine Learning*, PMLR 48, 276-284.

Chwialkowski, K., Strathmann, H., & Gretton, A. (2016). A kernel test of goodness of fit. *Proceedings of the 33rd International Conference on Machine Learning*, PMLR 48, 2606-2615.

Gorham, J., & Mackey, L. (2017). Measuring sample quality with kernels. *Proceedings of the 34th International Conference on Machine Learning*, PMLR 70, 1292-1301.

See Also

mmd_test(), mmd_ppc(), gaussian_score()

Other goodness-of-fit tests: concordance_test(), concordance_test_nystrom(), coverage_test(), gaussian_score(), ksd_test_nystrom(), numeric_score()

Examples

```
set.seed(1)

# Well-specified: standard-normal sample against standard-normal target
x_ok <- matrix(stats::rnorm(400L), ncol = 2L)
fit_ok <- ksd_test(x_ok, n_boot = 199L, seed = 1L)
fit_ok

# Mis-specified: mean-shifted sample against the same target
x_bad <- x_ok + 1
fit_bad <- ksd_test(x_bad, n_boot = 199L, seed = 1L)
fit_bad

# Explicit non-standard target via the Gaussian score factory
sig <- matrix(c(1, 0.6, 0.6, 1), nrow = 2L)
x_cor <- x_ok %*% chol(sig)
ksd_test(x_cor, score = gaussian_score(sigma = sig),
         n_boot = 199L, seed = 1L)
```

---

ksd_test_nystrom *Accelerated Kernel Stein Discrepancy Goodness-of-Fit Test (Nystrom)*

Description

Low-rank counterpart to ksd_test() for large samples. The n x n Stein-kernel matrix is never materialised: it is replaced by a Nystrom factor F (n x m, m << n) with $FF^\top \approx U$, and both the KSD U-statistic and its wild-bootstrap null are computed from F in O(n m) rather than O(n^2). Because the Langevin Stein kernel is itself positive semi-definite, $FF^\top$ is a valid Stein kernel in its own right, so this is exactly the ksd_test() procedure applied to the rank-m kernel $FF^\top$: the wild-bootstrap calibration of the degenerate U-statistic null is preserved, and the approximation trades statistical power – not test validity – for speed.

Usage

```
ksd_test_nystrom(
  x,
  score = NULL,
  kernel = c("imq", "rbf"),
  beta = -0.5,
  bandwidth = "median",
  method = c("nystrom"),
  m = 100L,
  n_boot = 1000L,
  alpha = 0.05,
  seed = NULL,
  regularise = 1e-06
)
```

Arguments

x	Numeric vector, matrix, or data.frame. The n x d sample to test, one observation per row. At least five rows are required.
score	Either NULL or a function. When NULL (default) the target is the standard multivariate normal, with score $-x$. When a function, it must accept the n x d sample matrix and return the n x d matrix of scores $\nabla_x \log p(x)$ evaluated row-wise; see gaussian_score() for the multivariate-normal factory.
kernel	Character. Base kernel for the Stein kernel: "imq" (inverse multi-quadric, default) or "rbf" (Gaussian).
beta	Numeric in (-1, 0). Exponent of the IMQ kernel; ignored when kernel = "rbf". Default -0.5.
bandwidth	Numeric or "median". The IMQ offset c or the RBF bandwidth h. "median" (default) uses the median-heuristic bandwidth of the sample.
method	Character. Factorisation method; currently "nystrom" only.
m	Integer. Number of Nystrom landmarks (the approximation rank); capped at n - 1. Larger m improves power at higher cost. Default 100L.
n_boot	Integer. Number of wild-bootstrap replicates for the null. Default 1000.
alpha	Numeric in (0, 1). Significance level for the verdict. Default 0.05.
seed	Integer or NULL. Random seed for reproducibility.
regularise	Small positive numeric. Ridge added to the landmark Stein block before its Cholesky, for numerical stability. Default 1e-6.

Details

Currently Nystrom-only. Random-Fourier-feature factorisation of the Stein kernel requires the analytic Fourier derivatives of the base kernel and is deferred; the `method` argument is reserved for that extension.

Use `ksd_test()` for exact results at moderate `n`; reach for this when the `O(n^2)` Stein matrix is the bottleneck. The verdict object and its fields match `ksd_test()` plus `approximation` and `m`.

Value

An object of class `c("ksd_test", "kernel_test_result")` carrying the same fields as `ksd_test()` plus:

approximation `"nystrom"`.

m Effective rank used for the factorisation.

Author(s)

Max Moldovan, <max.moldovan@adelaide.edu.au>

References

Liu, Q., Lee, J. D., & Jordan, M. I. (2016). A kernelized Stein discrepancy for goodness-of-fit tests. *ICML*, PMLR 48, 276-284.

Chwialkowski, K., Strathmann, H., & Gretton, A. (2016). A kernel test of goodness of fit. *ICML*, PMLR 48, 2606-2615.

Williams, C. K. I., & Seeger, M. (2001). Using the Nystrom method to speed up kernel machines. *NeurIPS*, 13.

See Also

`ksd_test()`, `nystrom_factor()`, `hsic_test_nystrom()`, `concordance_test_nystrom()`

Other goodness-of-fit tests: `concordance_test()`, `concordance_test_nystrom()`, `coverage_test()`, `gaussian_score()`, `ksd_test()`, `numeric_score()`

Other low-rank acceleration: `concordance_test_nystrom()`, `hsic_test_nystrom()`, `nystrom_factor()`, `rff_features()`

Examples

```
set.seed(1)
x_ok  <- matrix(stats::rnorm(4000L), ncol = 2L)
fit_ok <- ksd_test_nystrom(x_ok, m = 80L, n_boot = 199L, seed = 1L)
fit_ok

x_bad <- x_ok + 1
ksd_test_nystrom(x_bad, m = 80L, n_boot = 199L, seed = 1L)
```

lhs_design *Latin-Hypercube Design Over Bounded Parameters*

Description

Generates a Latin-hypercube sample of size `n` over the parameter bounds in `bounds`. Each column is a random permutation of stratified uniform draws (one per equal-width bin in `(0, 1]`), then scaled to the supplied parameter range. The result is reproducible when `seed` is supplied.

Usage

```
lhs_design(n, bounds, seed = NULL)
```

Arguments

`n`	Integer. Number of design points (rows). Must be `>= 2`.
`bounds`	Two-column numeric matrix or data.frame of `[lower, upper]` bounds, with one row per parameter. If named, row names propagate to the column names of the returned design.
`seed`	Integer or `NULL`. Random seed for reproducibility.

Details

This is a lightweight helper aimed at pre-PESTO screening: produce a design matrix to feed an APSIM (or any) simulator, then pass the resulting input/output pairs to `hsic_identifiability()` to flag unidentifiable parameters before ensemble-smoother calibration.

Value

A numeric matrix of dimension `n x nrow(bounds)`. Column names are inherited from `rownames(bounds)` when available, otherwise `theta1`, `theta2`, ...

References

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2), 239-245.

See Also

`hsic_identifiability()`

Other sensitivity and identifiability: `hsic_identifiability()`, `hsic_sensitivity()`

Examples

```
bounds <- rbind(
  slope     = c(0.1, 2.0),
  intercept = c(-1, 1),
  noise_sd  = c(0.05, 0.5)
)
design <- lhs_design(50, bounds, seed = 1)
head(design)
```

---

mmd_ppc *MMD Posterior-Predictive Check*

Description

Model-free verdict on whether a posterior-predictive ensemble is consistent with held-out observations, via the Maximum Mean Discrepancy (MMD) two-sample test. Wraps `mmd_test()` with a posterior-predictive framing and adds a Shannon-information *surprise* diagnostic (`-log2(p)`) for intuitive interpretation: 0 bits = no surprise (`p = 1`); ~4.32 bits = `p = 0.05`; the maximum achievable surprise at `n_permutations = B` is `log2(B + 1)`.

Usage

```
mmd_ppc(x, ...)

## Default S3 method:
mmd_ppc(
  x,
  observed,
  kernel = kernel_spec(),
  n_permutations = 500L,
  alpha = 0.05,
  seed = NULL,
  ...
)

## S3 method for class 'pesto_ensemble'
mmd_ppc(x, observed = NULL, ...)

## S3 method for class 'pesto_ensemble_manifest'
mmd_ppc(x, observed, outputs = NULL, ...)
```

Arguments

`x`	Either a numeric matrix `M x d` of posterior-predictive draws, or a `pesto_ensemble` object (see `pesto_ensemble()`).

...	Additional arguments (currently unused; reserved for future Nystrom acceleration).
observed	Numeric matrix n_obs x d of held-out observations. When x is a pesto_ensemble carrying its own observed slot, may be left NULL to use the bundled observations.
kernel	Kernel specification. Default is RBF with median heuristic over the pooled posterior + observed sample.
n_permutations	Integer. Permutations for the null. Default 500.
alpha	Numeric in (0, 1). Significance level used for the verdict. Default 0.05.
seed	Integer or NULL. Random seed for reproducibility.
outputs	Optional character vector of output column names from the manifest to test against. Defaults to all numeric output columns (the real_name column is excluded). Used only for the pesto_ensemble_manifest method.

Details

Use after an ensemble-smoother run (PESTO IES, EnKF, etc.) to ask: *does the calibrated model produce predictive draws that match the held-out year / paddock / season at the distributional level?* The MMD test is sensitive to mean, variance, and tail differences – strictly more informative than RMSE on the posterior predictive mean.

The pesto_ensemble_manifest method records provenance from the input manifest in result$pesto_metadata: run_id, pesto_version, method, outputs_used, and fidelity (the multi-fidelity provenance record, or NULL for a single-fidelity run), so the PPC verdict traces back to the producing ensemble and the fidelity it was calibrated at.

Value

An object of class c("mmd_ppc", "kernel_test_result") with the standard kernel_test_result fields plus:

n_posterior Number of posterior-predictive draws.

n_observed Number of held-out observations.

surprise_bits Shannon-information surprise -log2(p_value).

alpha Verdict significance level.

reject Logical: p_value <= alpha.

pesto_metadata Carried through from pesto_ensemble input, when provided; otherwise NULL.

References

Gretton, A., Borgwardt, K. M., Rasch, M. J., Scholkopf, B., & Smola, A. (2012). A kernel two-sample test. *JMLR*, 13, 723-773.

Gelman, A., Meng, X.-L., & Stern, H. (1996). Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica*, 6(4), 733-760.

See Also

mmd_test(), pesto_ensemble()

Other posterior predictive checks: pesto_ensemble()

Examples

```
set.seed(1)
# Calibrated model: posterior matches truth
post <- matrix(stats::rnorm(400L), ncol = 2L)
obs  <- matrix(stats::rnorm(40L),  ncol = 2L)
fit_ok <- mmd_ppc(post, obs, n_permutations = 199L, seed = 1L)
fit_ok

# Miscalibrated model: posterior is mean-shifted
obs_shift <- obs + 1.5
fit_bad <- mmd_ppc(post, obs_shift, n_permutations = 199L, seed = 1L)
fit_bad
```

---

mmd_test *MMD Two-Sample Test*

Description

Tests whether two samples come from the same distribution using the Maximum Mean Discrepancy (MMD). Uses a permutation test for inference.

Usage

```
mmd_test(
  x,
  y,
  kernel = kernel_spec(),
  n_permutations = 500L,
  alpha = 0.05,
  seed = NULL
)
```

Arguments

x	Numeric vector, matrix, or data.frame. First sample.
y	Numeric vector, matrix, or data.frame. Second sample.
kernel	Kernel specification. Default is RBF with median heuristic.
n_permutations	Integer. Number of permutations. Default is 500.
alpha	Numeric. Significance level. Default is 0.05.
seed	Integer or NULL. Random seed for reproducibility.

Value

An object of class "kernel_test_result" with components:

statistic The observed MMD^2 statistic (unbiased).

p_value Permutation p-value.

method "MMD".

n Total sample size (n_x + n_y).

n_permutations Number of permutations used.

null_distribution Vector of permuted MMD^2 values.

kernel_x Kernel specification used.

call The matched call.

References

Gretton, A., Borgwardt, K. M., Rasch, M. J., Scholkopf, B., & Smola, A. (2012). A kernel two-sample test. *JMLR*, 13, 723-773.

See Also

Other independence and two-sample tests: hsic_test()

Examples

```
set.seed(42)

# Same distribution
x <- matrix(rnorm(200), 100, 2)
y <- matrix(rnorm(200), 100, 2)
result <- mmd_test(x, y)
print(result)

# Different distributions
y_shifted <- matrix(rnorm(200, mean = 1), 100, 2)
result <- mmd_test(x, y_shifted)
print(result)
```

---

numeric_score *Finite-difference score from a log-density*

Description

Builds a score function – the gradient of the log density – by central finite differences, for use as the score argument of ksd_test(). The target need only be expressible as a function returning a (possibly unnormalised) log density; any additive normalising constant cancels in the gradient, so the constant may be omitted.

Usage

```
numeric_score(log_density, h = 1e-04)
```

Arguments

log_density	A function accepting an n x d numeric matrix and returning a numeric vector of length n: the log density (up to an additive constant) evaluated row-wise.
h	Positive numeric. Finite-difference step. Default 1e-4. Too large biases the gradient; too small amplifies floating-point noise.

Details

This makes ksd_test() usable against any target a user can write down as a log density, not only targets with a hand-derived score. It is also the kernR-side adapter for a log-posterior contract: wrapping an exported log-posterior evaluator in numeric_score() yields the score ksd_test() needs to check whether posterior draws are calibrated against that posterior, with no dependency added on the producer – the evaluator is passed in as an ordinary function.

Central differences cost 2 * d evaluations of log_density per call, where d is the dimension. For a cheap closed-form target prefer gaussian_score() or a hand-written score; numeric_score() is the general fallback.

Value

A function of one argument (an n x d numeric matrix) returning the n x d matrix of finite-difference scores, suitable as the score argument of ksd_test().

Author(s)

Max Moldovan, <max.moldovan@adelaide.edu.au>

See Also

ksd_test(), gaussian_score()

Other goodness-of-fit tests: concordance_test(), concordance_test_nystrom(), coverage_test(), gaussian_score(), ksd_test(), ksd_test_nystrom()

Examples

```
set.seed(1)
x <- matrix(stats::rnorm(200L), ncol = 2L)

# Standard-normal log density (unnormalised): -||x||^2 / 2
ld <- function(z) -0.5 * rowSums(z^2)
s <- numeric_score(ld)

# Matches the closed-form standard-normal score -x to finite-difference order
max(abs(s(x) - (-x)))

# Use directly in a goodness-of-fit test
```

```
ksd_test(x, score = numeric_score(ld), n_boot = 199L, seed = 1L)
```

`nystrom_factor` *Nystrom Low-Rank Kernel Factorisation*

Description

Builds a Nystrom approximation $K \approx FF^\top$ of an `n x n` kernel matrix using `m << n` uniformly-sampled landmarks. The returned factor `F` is an `n x m` matrix; downstream kernel computations (HSIC, MMD, etc.) that can be expressed in `F` reduce from `O(n^2)` to `O(n m)` cost.

Usage

```
nystrom_factor(
  x,
  kernel = kernel_spec(),
  m = 100L,
  regularise = 1e-06,
  seed = NULL
)
```

Arguments

`x`	Numeric matrix `n x d` (or vector). Data points.
`kernel`	A `kernel_spec()`. Default RBF with median heuristic.
`m`	Integer. Number of landmark points. Capped at `nrow(x) - 1`. Default `100L`.
`regularise`	Small positive numeric. Ridge added to `W` before Cholesky for numerical stability. Default `1e-6`.
`seed`	Integer or `NULL`. Random seed for landmark sampling.

Details

The construction is:

1. Sample `m` landmark indices uniformly without replacement.
2. Compute the landmark Gram $W = K_{mm}$ (`m x m`) and the cross-Gram $C = K_{nm}$ (`n x m`).
3. Stabilise: $W_\epsilon = W + \epsilon I$.
4. Cholesky factor $W_\epsilon = LL^\top$.
5. Return $F = CL^{-\top}$, so that $FF^\top = CW_\epsilon^{-1}C^\top \approx K$.

Any `kernel_spec()` is supported. Bandwidth selection (median heuristic) is performed on the full dataset before landmark sampling.

Value

An object of class `"kernel_factor"` with components:

F The `n x m_eff` factor matrix.

method `"nystrom"`.

m Effective rank (`<= m`).

kernel Resolved kernel spec.

n Number of rows in the input.

References

Williams, C. K. I., & Seeger, M. (2001). Using the Nystrom method to speed up kernel machines. *NeurIPS*, 13.

See Also

`rff_features()`, `hsic_test_nystrom()`

Other low-rank acceleration: `concordance_test_nystrom()`, `hsic_test_nystrom()`, `ksd_test_nystrom()`, `rff_features()`

Examples

```
set.seed(1)
x <- matrix(stats::rnorm(2000L), ncol = 2L)
f <- nystrom_factor(x, m = 80L, seed = 1L)
dim(f$F)
```

---

pesto_ensemble *PESTO Ensemble Manifest (Constructor)*

Description

Lightweight constructor for a posterior-predictive ensemble produced by PESTO (or any compatible UQ engine). Bundles a posterior-predictive sample matrix with optional held-out observations and free-form metadata, providing a stable interface for `mmd_ppc()` and other downstream verdict-layer tools.

Usage

```
pesto_ensemble(posterior, observed = NULL, metadata = list())
```

Arguments

posterior	Numeric matrix M x d: M posterior-predictive draws over d output dimensions (e.g. yield, biomass).
observed	Optional numeric matrix n_obs x d of held-out observations. May be NULL when observations are supplied later to mmd_ppc().
metadata	Optional named list of free-form metadata (run id, ensemble seed, holdout year, etc.).

Details

This is the kernR-side schema for the cross-package contract; until PESTO ships its native manifest emitter, callers can construct the object directly from in-memory matrices.

Value

An object of class "pesto_ensemble".

See Also

mmd_ppc()

Other posterior predictive checks: mmd_ppc()

Examples

```
set.seed(1)
post <- matrix(stats::rnorm(200L), ncol = 2L)
obs  <- matrix(stats::rnorm(20L),  ncol = 2L)
ens  <- pesto_ensemble(post, obs, metadata = list(holdout_year = 2018))
ens
```

---

plot.hsic_identifiability

Plot an HSIC Identifiability Scan

Description

Bar plot of per-parameter maximum HSIC across outputs, ordered by magnitude. Bars for identifiable parameters are coloured; non- identifiable parameters are shown in grey.

Usage

```
## S3 method for class 'hsic_identifiability'
plot(x, col_yes = "#0072B2", col_no = "grey70", ...)
```

Arguments

x	An hsic_identifiability object.
col_yes, col_no	Bar colours for identifiable / non-identifiable parameters.
...	Additional arguments passed to graphics::barplot().

Value

Invisibly returns x. Side effect: produces a base R plot.

Examples

```
set.seed(1)
n <- 50
theta <- matrix(stats::runif(n * 3), nrow = n,
                dimnames = list(NULL, c("active", "active2", "inert")))
y <- theta[, 1] + theta[, 2]^2 + stats::rnorm(n, sd = 0.1)
fit <- hsic_identifiability(theta, y, n_permutations = 199, seed = 1)
plot(fit)
```

---

plot.hsic_sensitivity *Plot HSIC-Sensitivity Indices*

Description

Bar plot of per-parameter HSIC-Sensitivity Index, ordered by first-order index magnitude. When total_order was set on the fit, which = "total" shows the total-order indices and which = "both" shows side-by-side bars for S and T.

Usage

```
## S3 method for class 'hsic_sensitivity'
plot(
  x,
  which = c("first", "total", "both"),
  alpha = 0.05,
  col_sig = "#0072B2",
  col_nonsig = "grey70",
  col_total = "#D55E00",
  ...
)
```

Arguments

x	An hsic_sensitivity object.
which	Character. "first" (default), "total", or "both". "total" / "both" require x$total_order to be TRUE.
alpha	Numeric in (0, 1). Significance level for colour coding (first-order only; ignored if the object carries no p-values or under which = "total").
col_sig, col_nonsig, col_total	Bar colours.
...	Additional arguments passed to graphics::barplot().

Value

Invisibly returns x. Side effect: produces a base R plot.

---

plot.kernel_test_result

Plot a Kernel Test Result

Description

Plots the permutation null distribution with the observed statistic.

Usage

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

Arguments

x	A kernel_test_result object.
...	Additional arguments (currently ignored).

Value

Invisibly returns x. Side effect: produces a base R plot.

Examples

```
set.seed(42)
x_data <- rnorm(100)
y_data <- x_data + rnorm(100, sd = 0.5)
res <- hsic_test(x_data, y_data)
plot(res)
```

plot_weights *Plot Weight Diagnostics*

Description

Plots the distribution of importance weights with effective sample size annotation.

Usage

```
plot_weights(weights, main = "Weight Distribution")
```

Arguments

weights	Numeric vector of importance weights.
main	Title. Default is "Weight Distribution".

Value

Invisibly returns weights.

See Also

Other density ratio and propensity: assess_overlap(), effective_sample_size(), estimate_density_ratio(), estimate_propensity(), fit_density_ratio(), predict_density_ratio()

Examples

```
set.seed(1L)
weights <- rgamma(200L, shape = 2, rate = 2)
plot_weights(weights)
```

posterior_sample_aggregate *Sample from the posterior of an aggregate-downscale fit*

Description

Draws n samples from the per-component posterior mixture. Each draw picks a component by posterior_weights, then samples from N(posterior_components_means[[k]], posterior_components_covariances

Usage

```
posterior_sample_aggregate(object, n = 1000L, seed = NULL)
```

Arguments

object	An aggregate_downscale fit.
n	Integer. Number of posterior samples. Default 1000L.
seed	Integer or NULL. Random seed.

Value

An n x dim_x numeric matrix.

See Also

Other downscaling and embeddings: aggregate_downscale(), dist_regression(), fit_cme(), kernel_downscale()

Examples

```
set.seed(1L)
A <- matrix(c(0.5, 0.5), nrow = 1L)
prior <- list(
  means = list(c(0, 0), c(2, 2)),
  covariances = list(diag(2L), diag(2L)),
  weights = c(0.5, 0.5)
)
fit <- aggregate_downscale(y = 1.0, aggregator = A,
                           latent_prior = prior, sigma_y = 0.2)
draws <- posterior_sample_aggregate(fit, n = 500L, seed = 1L)
colMeans(draws)
```

---

predict.cme_fit *Predict Conditional Mean Embedding Weights at New Points*

Description

Returns the row weights $\alpha(x^*) = k(x^*, X_{\text{train}})W$ from a fitted fit_cme() object. Each row of the result is the weight vector for combining training-y quantities (kernel values or values themselves) to produce a CME prediction at x_new.

Usage

```
## S3 method for class 'cme_fit'
predict(object, x_new, ...)
```

Arguments

object	A cme_fit object.
x_new	Numeric matrix of new conditioning points.
...	Currently ignored.

Details

For a typical "predict Y at new X" workflow use `kernel_downscale()`, which combines this with the training Y matrix to return predictions directly.

Value

An n_new x n_train matrix of embedding weights.

---

predict.dist_regression *Predict from a Fitted Distribution Regression Model*

Description

Predict from a Fitted Distribution Regression Model

Usage

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

Arguments

`object`	A `dist_regression` fit.
`newdata`	A list of new bags (matrices with the same number of columns as the training bags). Bag sizes may differ from training.
`...`	Currently ignored.

Value

Numeric vector (length `length(newdata)`) when the model was fitted with a scalar `y`; numeric matrix (`length(newdata) x d_y`) for multivariate `y`.

---

predict_density_ratio *Predict from a Fitted Density-Ratio Model*

Description

Applies a `density_ratio_fit` object (from `fit_density_ratio()`) to new `(x, z)` rows. All four backends compute ratios in log-space internally for numerical stability; `type` controls the returned representation.

Usage

```
predict_density_ratio(
  object,
  new_x,
  new_z,
  type = c("log_ratio", "weight", "ratio")
)
```

Arguments

object	A density_ratio_fit.
new_x	Numeric vector or matrix. Treatment values to evaluate.
new_z	Numeric matrix or data.frame. Confounders to evaluate.
type	Character. Return type: "log_ratio" – natural-log density ratio (default; preferred for downstream calculation); "ratio" – raw density ratio (exp(log_ratio)); "weight" – IES-compatible normalised weights (positive, sum-to-n_new).

Value

Numeric vector of length nrow(new_x).

See Also

fit_density_ratio().

Other density ratio and propensity: assess_overlap(), effective_sample_size(), estimate_density_ratio(), estimate_propensity(), fit_density_ratio(), plot_weights()

Examples

```
set.seed(1L)
n <- 200L
z <- matrix(rnorm(n * 2L), n, 2L)
x <- z[, 1L] + rnorm(n)
fit <- fit_density_ratio(x, z, method = "logistic", seed = 1L)
weights <- predict_density_ratio(fit, new_x = x, new_z = z,
                                 type = "weight")
summary(weights)
```

---

print.cme_fit *Print a Conditional Mean Embedding Fit*

Description

Prints a compact summary of a fitted conditional mean embedding: the training-sample size, the input and output dimensions, the resolved kernels, and the ridge regularisation parameter that was used.

Usage

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

Arguments

x	A cme_fit object returned by fit_cme().
...	Unused; present for S3 generic compatibility.

Value

The cme_fit object x, invisibly.

See Also

fit_cme()

Examples

```
set.seed(1L)
x <- matrix(rnorm(60L), ncol = 2L)
y <- matrix(x[, 1L] + rnorm(30L, sd = 0.2), ncol = 1L)
print(fit_cme(x, y, lambda = 1e-2))
```

---

rff_features *Random Fourier Features for the RBF Kernel*

Description

Builds an n x D feature map Φ such that $\Phi\Phi^\top \approx K$ for the RBF kernel $K(x, y) = \exp(-\|x - y\|^2/(2\sigma^2))$, via Rahimi & Recht (2007): draw $\omega_k \sim N(0, \sigma^{-2} I_d)$ and $b_k \sim U[0, 2\pi]$, then $\phi_k(x) = \sqrt{2/D}\cos(\omega_k^\top x + b_k)$.

Usage

```
rff_features(x, kernel = kernel_spec("rbf"), D = 200L, seed = NULL)
```

Arguments

x	Numeric matrix n x d (or vector).
kernel	A kernel_spec() with type = "rbf". Bandwidth may be "median" (resolved against x) or a positive numeric.
D	Integer. Number of random features. Larger D -> better approximation but higher memory / compute. Default 200L.
seed	Integer or NULL. Random seed.

Details

Currently RBF-only; other shift-invariant kernels (Matern with specific nu) require their own Fourier spectra and are not yet implemented.

Value

An object of class "kernel_factor" with components:

F The n x D random-feature matrix.

method "rff".

m Equal to D.

kernel Resolved kernel spec.

n Number of rows in the input.

omega, b Random draws (kept for reproducible re-encoding).

References

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *NeurIPS*, 20.

See Also

nystrom_factor(), hsic_test_nystrom()

Other low-rank acceleration: concordance_test_nystrom(), hsic_test_nystrom(), ksd_test_nystrom(), nystrom_factor()

Examples

```
set.seed(1)
x <- matrix(stats::rnorm(2000L), ncol = 2L)
f <- rff_features(x, D = 150L, seed = 1L)
dim(f$F)
```

---

select_bandwidth *Select Kernel Bandwidth*

Description

Computes a bandwidth (lengthscale) for RBF or Matern kernels using the specified method.

Usage

```
select_bandwidth(x, method = "median")
```

Arguments

x	Numeric matrix or vector.
method	Character. "median" (default), "scott", or a positive number for fixed bandwidth.

Details

- "median": The median heuristic sets bandwidth to the square root of the median of pairwise squared distances. Robust default for most kernel tests (Gretton et al., 2012).
- "scott": Scott's rule: n^(-1/(d+4)) * sd_pooled. Good for density estimation but may undersmooth for testing.

Value

A positive numeric scalar.

See Also

Other kernel primitives: kernel_matrix(), kernel_spec()

Examples

```
x <- matrix(rnorm(200), 100, 2)
select_bandwidth(x, "median")
```

Index