Package 'bioLeak'

May 21, 2026

Type Package

Title Leakage-Safe Modeling and Auditing for Genomic and Clinical Data

Version 0.3.8

Description Prevents and detects information leakage in biomedical machine learning.
Provides leakage-resistant split policies (subject-grouped, batch-blocked, study leave-out, time-ordered),
guarded preprocessing (train-only imputation, normalization, filtering, feature selection),
cross-validated fitting with common learners, permutation-
gap auditing, batch and fold association tests,
and duplicate detection.

License MIT + file LICENSE

URL https://github.com/selcukorkmaz/bioLeak

BugReports https://github.com/selcukorkmaz/bioLeak/issues

Encoding UTF-8

Depends R (>= 4.3)

Imports digest, generics, methods, stats, utils, SummarizedExperiment, graphics, hardhat, parsnip

Suggests BiocParallel, splitGraph, cli, dials, FNN, future, future.apply, ggplot2, glmnet, mice, missForest, pkgload, ranger, randomForest, recipes, RANN, rsample, tune, VIM, withr, workflows, xgboost, yardstick, pROC, PRROC, survival, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.3

Collate 'classes.R' 'accessors.R' 'audit.R' 'audit_report.R' 'benchmark_suite.R' 'ci_delta.R' 'conditions.R' 'cv_uncertainty.R' 'cvauc_if.R' 'delta_lsi.R' 'diagnostics_extra.R' 'fit_resample.R' 'globals.R' 'guard_to_recipe.R' 'guards.R' 'imports.R' 'impute_guarded.R' 'make_split_plan.R' 'permute_labels.R' 'plotting_audit.R'

'provenance.R' 'simulate_suite.R' 'splitgraph_adapter.R'
'summary.R' 'tidymodels.R' 'time_series_blocks.R'
'tune_resample.R' 'utils_metrics.R' 'utils_print.R'
'utils_se.R'

NeedsCompilation no

Author Selcuk Korkmaz [aut, cre] (ORCID:
<https://orcid.org/0000-0003-4632-6850>)

Maintainer Selcuk Korkmaz <selcukorkmaz@gmail.com>

Repository CRAN

Date/Publication 2026-05-21 18:40:02 UTC

Contents

as_leaksplits . . . 3
as_rsample . . . 4
audit_batch_assoc . . . 5
audit_duplicates . . . 6
audit_info . . . 7
audit_leakage . . . 7
audit_leakage_by_learner . . . 13
audit_perm_gap . . . 15
audit_report . . . 16
audit_target_assoc . . . 18
benchmark_leakage_suite . . . 19
calibration_summary . . . 20
check_split_overlap . . . 21
confounder_sensitivity . . . 22
cv_ci . . . 23
delta_lsi . . . 24
dlsi_ci . . . 27
dlsi_metric . . . 27
dlsi_p_value . . . 28
dlsi_repeats . . . 29
dlsi_robust . . . 30
dlsi_R_eff . . . 30
dlsi_tier . . . 31
fit_metrics . . . 32
fit_resample . . . 33
guard_ensure_levels . . . 36
guard_fit . . . 37
guard_to_recipe . . . 38
impute_guarded . . . 39
LeakSplits-class . . . 40
make_split_plan . . . 42
plot,LeakAudit,missing-method . . . 45
plot,LeakDeltaLSI,missing-method . . . 45

plot,LeakFit,missing-method . . . 46
plot_calibration . . . 47
plot_confounder_sensitivity . . . 48
plot_dlsi_repeats . . . 50
plot_fold_balance . . . 50
plot_overlap_checks . . . 51
plot_perm_distribution . . . 53
plot_time_acf . . . 54
predict.GuardFit . . . 55
print.LeakTune . . . 56
show,LeakAudit-method . . . 57
show,LeakFit-method . . . 58
show,LeakSplits-method . . . 59
simulate_leakage_suite . . . 60
summary.LeakAudit . . . 63
summary.LeakDeltaLSI . . . 64
summary.LeakFit . . . 65
summary.LeakTune . . . 66
tune_resample . . . 67

Index **71**

`as_leaksplits` *Convert a splitGraph split_spec into bioLeak splits*

Description

Consume a 'split_spec' produced by **splitGraph** and build a corresponding `LeakSplits` object via `make_split_plan`. The spec supplies the grouping/blocking/ordering assignments; the caller supplies the observation frame (features + outcome), joined on sample id.

Usage

```
as_leaksplits(spec, data, outcome, sample_id_col = "sample_id", v = 5, ...)
```

Arguments

`spec`	A `split_spec` object from **splitGraph**.
`data`	A data.frame (or SummarizedExperiment) containing at least one identifier column matching `sample_id_col` and an `outcome` column.
`outcome`	Name of the outcome column in `data`.
`sample_id_col`	Name of the sample-id column in `data` (default `"sample_id"`).
`v`	Number of CV folds to request from `make_split_plan()`.
`...`	Additional arguments forwarded to `make_split_plan` (e.g. `stratify`, `seed`, `horizon`, `purge`).

Details

The mapping from `spec$constraint_mode` to `make_split_plan(mode=)` is:

- `"subject"` -> `"subject_grouped"`
- `"batch"` -> `"batch_blocked"`
- `"study"` -> `"study_loocv"`
- `"time"` -> `"time_series"`
- `"composite"` -> `"combined"`

Blocking variables declared on the spec (`batch_group`, `study_group`) and ordering (`order_rank`) are forwarded automatically when relevant.

Value

A `LeakSplits` object.

See Also

`make_split_plan`, `as_rsample`

`as_rsample` *Convert LeakSplits to an rsample resample set*

Description

Convert LeakSplits to an rsample resample set

Usage

```
as_rsample(x, data = NULL, ...)
```

Arguments

`x`	LeakSplits object created by [make_split_plan()].
`data`	Optional data.frame used to populate rsample splits. When NULL, the stored ‘coldata‘ from ‘x‘ is used (if available).
`...`	Additional arguments passed to methods (unused).

Value

An rsample `rset` object compatible with tidymodels workflows. The returned object is a tibble with class `rset` containing:

`splits` List-column of `rsplit` objects, each with `analysis` (training indices) and `assessment` (test indices).

`id` Character column with fold identifiers (e.g., "Fold1").

`id2` Character column with repeat identifiers (e.g., "Repeat1") when multiple repeats are present; otherwise absent.

The object also carries attributes for `group`, `batch`, `study`, `time` (when available from the original `LeakSplits`), and `bioLeak_mode` indicating the original splitting mode. This allows the splits to be used with `tune::tune_grid()`, `rsample::fit_resamples()`, and other tidymodels functions.

Examples

```
if (requireNamespace("rsample", quietly = TRUE)) {
  df <- data.frame(
    subject = rep(1:10, each = 2),
    outcome = rbinom(20, 1, 0.5),
    x1 = rnorm(20),
    x2 = rnorm(20)
  )
  splits <- make_split_plan(df, outcome = "outcome",
                        mode = "subject_grouped", group = "subject", v = 5)
  rset <- as_rsample(splits, data = df)
}
```

---

audit_batch_assoc *Batch / study association results from a LeakAudit*

Description

Returns the batch/study chi-squared association data frame stored in a [‘LeakAudit‘] object. Columns include the metadata column, repeat, chi-squared statistic, degrees of freedom, p-value, and Cramer’s V.

Usage

```
audit_batch_assoc(audit)

## S4 method for signature 'LeakAudit'
audit_batch_assoc(audit)
```

Arguments

`audit`	A [‘LeakAudit‘] object returned by [audit_leakage()].

Details

Implemented as an S4 generic with a method for [‘LeakAudit‘]; visible via ‘methods(class = "LeakAudit")‘.

Value

A ‘data.frame‘ with one row per (metadata column, repeat).

See Also

[LeakClasses], [audit_leakage()]

`audit_duplicates` *Near-duplicate pairs from a LeakAudit*

Description

Returns the near-duplicate sample pairs data frame stored in a [‘LeakAudit‘] object. Each row is a unique (row_a, row_b) pair above the configured cosine-similarity threshold that crossed train/test partitions in at least one fold.

Usage

```
audit_duplicates(audit)

## S4 method for signature 'LeakAudit'
audit_duplicates(audit)
```

Arguments

`audit`	A [‘LeakAudit‘] object returned by [audit_leakage()].

Details

Implemented as an S4 generic with a method for [‘LeakAudit‘]; visible via ‘methods(class = "LeakAudit")‘.

Value

A ‘data.frame‘ with one row per detected near-duplicate pair.

See Also

[LeakClasses], [audit_leakage()]

`audit_info` *Auxiliary information from a LeakAudit*

Description

Returns the auxiliary information list stored in a [‘LeakAudit‘] object. The list typically contains multivariate-target-scan results, configuration flags, permutation-test diagnostics, and provenance metadata.

Usage

```
audit_info(audit)

## S4 method for signature 'LeakAudit'
audit_info(audit)
```

Arguments

`audit`	A [‘LeakAudit‘] object returned by [audit_leakage()].

Details

Implemented as an S4 generic with a method for [‘LeakAudit‘]; visible via ‘methods(class = "LeakAudit")‘.

Value

A named ‘list‘.

See Also

[LeakClasses], [audit_leakage()]

`audit_leakage` *Audit leakage and confounding*

Description

Computes a post-hoc leakage audit for a resampled model fit. The audit (1) runs a permutation-gap test comparing observed cross-validated performance to a label-permutation null (by default refitting when data are available; otherwise using fixed predictions), (2) tests whether fold assignments are associated with batch or study metadata (confounding by design), (3) scans features for unusually strong outcome proxies, and (4) flags duplicate or near-duplicate samples in a reference feature matrix.

The returned [LeakAudit] summarizes these diagnostics. It relies on the stored predictions, splits, and optional metadata; it does not refit models unless ‘perm_refit = TRUE‘ (or ‘perm_refit = "auto"‘ with a valid ‘perm_refit_spec‘). Results are conditional on the chosen metric and supplied metadata/features and should be interpreted as diagnostics, not proof of leakage or its absence.

Usage

```
audit_leakage(
  fit,
 metric = c("auc", "pr_auc", "accuracy", "macro_f1", "log_loss", "rmse", "cindex"),
  B = 200,
  perm_stratify = FALSE,
  perm_refit = "auto",
  perm_refit_auto_max = 200,
  perm_refit_spec = NULL,
  perm_mode = NULL,
  time_block = c("circular", "stationary"),
  block_len = NULL,
  include_z = TRUE,
  ci_method = c("if", "bootstrap"),
  boot_B = 400,
  parallel = FALSE,
  seed = 1,
  return_perm = TRUE,
  batch_cols = NULL,
  coldata = NULL,
  X_ref = NULL,
  target_scan = TRUE,
  target_scan_multivariate = TRUE,
  target_scan_multivariate_B = 100,
  target_scan_multivariate_components = 10,
  target_scan_multivariate_interactions = TRUE,
  target_threshold = 0.9,
  target_p_adjust = c("none", "BH", "BY", "holm", "bonferroni"),
  target_alpha = 0.05,
  feature_space = c("raw", "rank"),
  sim_method = c("cosine", "pearson"),
  sim_threshold = 0.995,
  nn_k = 50,
  max_pairs = 5000,
  duplicate_scope = c("train_test", "all"),
  learner = NULL,
  strict_align = FALSE
)
```

Arguments

`fit`	A [LeakFit] object from [fit_resample()] containing cross-validated predictions and split metadata. If predictions include learner IDs for multiple models, you must supply ‘learner‘ to select one; if learner IDs are absent, the audit uses all predictions and may mix learners.
`metric`	Character scalar. One of ‘"auc"‘, ‘"pr_auc"‘, ‘"accuracy"‘, ‘"macro_f1"‘, ‘"log_loss"‘, ‘"rmse"‘, or ‘"cindex"‘. Defaults to ‘"auc"‘. This controls the observed performance statistic, the permutation null, and the sign of the reported gap.

`B` Integer scalar. Number of permutations used to build the null distribution (default 200). Larger values reduce Monte Carlo error but increase runtime.

`perm_stratify` Logical scalar or ‘"auto"‘. If TRUE, permutations are stratified within each fold (factor levels; numeric outcomes are binned into quantiles when enough non-missing values are available). If FALSE, no stratification is used. Defaults to FALSE. Stratification only applies when ‘coldata‘ supplies the outcome; otherwise labels are shuffled within each fold.

`perm_refit` Logical scalar or ‘"auto"‘. If FALSE, permutations keep predictions fixed and shuffle labels (association test). If TRUE, each permutation refits the model on permuted outcomes using ‘perm_refit_spec‘. Refit-based permutations are slower but better approximate a full null distribution. The default is ‘"auto"‘, which refits only when ‘perm_refit_spec‘ is provided and ‘B‘ is less than or equal to ‘perm_refit_auto_max‘; otherwise it falls back to fixed-prediction permutations.

`perm_refit_auto_max`
Integer scalar. Maximum ‘B‘ allowed for ‘perm_refit = "auto"‘ to trigger refitting. Defaults to 200.

`perm_refit_spec`
List of inputs used when ‘perm_refit = TRUE‘. Required elements: ‘x‘ (data used for fitting) and ‘learner‘ (parsnip model_spec, workflow, or legacy learner). Optional elements: ‘outcome‘ (defaults to ‘fit@outcome‘), ‘preprocess‘, ‘learner_args‘, ‘custom_learners‘, ‘class_weights‘, ‘positive_class‘, and ‘parallel‘. Survival outcomes are not supported for refit-based permutations.

`perm_mode` Optional character scalar to override the permutation mode used for restricted shuffles. One of ‘"subject_grouped"‘, ‘"batch_blocked"‘, ‘"study_loocv"‘, or ‘"time_series"‘. Defaults to the split metadata when available (including rsample-derived modes).

`time_block` Character scalar, ‘"circular"‘ or ‘"stationary"‘. Controls block permutation for ‘time_series‘ splits; ignored for other split modes. Default is ‘"circular"‘.

`block_len` Integer scalar or NULL. Block length for time-series permutations. NULL selects ‘max(5, floor(0.1 * fold_size))‘. Larger values preserve more temporal structure and yield a more conservative null.

`include_z` Logical scalar. If TRUE (default), include the z-score for the permutation gap when a standard error is available; if FALSE, ‘z‘ is NA.

`ci_method` Character scalar, ‘"if"‘ or ‘"bootstrap"‘. Controls how the standard error and confidence interval for the permutation gap are estimated. Default is ‘"if"‘. ‘"if"‘ uses an influence-function estimate when available; ‘"bootstrap"‘ resamples permutation values ‘boot_B‘ times. Failed estimates yield NA.

`boot_B` Integer scalar. Number of bootstrap resamples when ‘ci_method = "bootstrap"‘ (default 400). Larger values are more stable but slower.

`parallel` Logical scalar. If TRUE and ‘future.apply‘ is available, permutations run in parallel. Results should match sequential execution. Default is FALSE.

`seed` Integer scalar. Random seed used for permutations and bootstrap resampling; changing it changes the randomization but not the observed metric. Default is 1.

`return_perm` Logical scalar. If TRUE (default), stores the permutation distribution in ‘audit@perm_values‘. Set FALSE to reduce memory use.

batch_cols
: Character vector. Names of ‘coldata‘ columns to test for association with fold assignment. If NULL, defaults to any of ‘"batch"‘, ‘"plate"‘, ‘"center"‘, ‘"site"‘, ‘"study"‘ found in ‘coldata‘. Changing this controls which batch tests appear in ‘batch_assoc‘.

coldata
: Optional data.frame of sample-level metadata. Rows must align to prediction ids via row names, a ‘row_id‘ column, or row order. Used to build restricted permutations (when the outcome column is present), compute batch associations, and supply outcomes for target scans. If NULL, uses ‘fit@splits@info$coldata‘ when available. If alignment fails, restricted permutations are disabled with a warning.

X_ref
: Optional numeric matrix/data.frame (samples x features). Used for duplicate detection and the target leakage scan. If NULL, uses ‘fit@info$X_ref‘ when available. Rows must align to sample ids (split order) via row names, a ‘row_id‘ column, or row order; misalignment disables these checks.

target_scan
: Logical scalar. If TRUE (default), computes per-feature outcome associations on ‘X_ref‘ and flags proxy features; if FALSE, or if ‘X_ref‘/outcomes are unavailable, ‘target_assoc‘ is empty. Not available for survival outcomes.

target_scan_multivariate
: Logical scalar. If TRUE (default), fits a simple multivariate/interaction model on ‘X_ref‘ using the stored splits and reports a permutation-based score/p-value. This is slower and only implemented for binomial and gaussian tasks.

target_scan_multivariate_B
: Integer scalar. Number of permutations for the multivariate scan (default 100). Larger values stabilize the p-value.

target_scan_multivariate_components
: Integer scalar. Maximum number of principal components used in the multivariate scan (default 10).

target_scan_multivariate_interactions
: Logical scalar. If TRUE (default), adds pairwise interactions among the top components in the multivariate scan.

target_threshold
: Numeric scalar in (0,1). Threshold applied to the association score used to flag proxy features. Higher values are stricter. Default is 0.9.

target_p_adjust
: Character scalar. Multiple-testing correction method applied to finite ‘target_assoc$p_value‘ values. One of ‘"none"‘ (default), ‘"BH"‘, ‘"BY"‘, ‘"holm"‘, or ‘"bonferroni"‘. Adds columns ‘p_value_adj‘ and ‘flag_fdr‘ to ‘target_assoc‘.

target_alpha
: Numeric scalar in (0,1). Significance level used for ‘flag_fdr‘ when ‘target_p_adjust != "none"‘. Default is 0.05.

feature_space
: Character scalar, ‘"raw"‘ or ‘"rank"‘. If ‘"rank"‘, each row of ‘X_ref‘ is rank-transformed before similarity calculations. This affects duplicate detection only. Default is ‘"raw"‘.

sim_method
: Character scalar, ‘"cosine"‘ or ‘"pearson"‘. Similarity metric for duplicate detection. ‘"pearson"‘ row-centers before cosine. Default is ‘"cosine"‘.

sim_threshold
: Numeric scalar in (0,1). Similarity cutoff for reporting duplicate pairs (default 0.995). Higher values yield fewer pairs.

`nn_k` Integer scalar. For large datasets (‘n > 3000‘) with ‘RANN‘ installed, checks only the nearest ‘nn_k‘ neighbors per row. Larger values increase sensitivity but slow the search. Ignored when full comparisons are used. Default is 50.

`max_pairs` Integer scalar. Maximum number of duplicate pairs returned. If more pairs are found, only the most similar are kept. This does not affect permutation results. Default is 5000.

`duplicate_scope` Character scalar. One of ‘"train_test"‘ (default) or ‘"all"‘. ‘"train_test"‘ retains only near-duplicate pairs that appear in train vs test in at least one repeat; ‘"all"‘ reports all near-duplicate pairs in ‘X_ref‘ regardless of fold assignment.

`learner` Optional character scalar. When predictions include multiple learner IDs, selects the learner to audit. If NULL and multiple learners are present, the function errors; if predictions lack learner IDs, this argument is ignored with a warning. Default is NULL.

`strict_align` Logical scalar. If TRUE, errors instead of warning when `X_ref` or `coldata` cannot be aligned to predictions by row names or IDs and would otherwise fall back to row-order matching. Default is FALSE for backward compatibility. Set to TRUE in production pipelines to catch silent misalignment.

Details

The ‘permutation_gap‘ slot reports ‘metric_obs‘, ‘perm_mean‘, ‘perm_sd‘, ‘gap‘, ‘z‘, ‘p_value‘, and ‘n_perm‘. The gap is defined as ‘metric_obs - perm_mean‘ for metrics where higher is better (AUC, PR-AUC, accuracy, macro-F1, C-index) and ‘perm_mean - metric_obs‘ for RMSE/log-loss. By default, ‘perm_refit = "auto"‘ refits models when refit data are available and ‘B‘ is not too large; otherwise it keeps predictions fixed and shuffles labels. Fixed-prediction permutations quantify prediction-label association rather than a full refit null. Set ‘perm_refit = FALSE‘ to force fixed predictions, or ‘perm_refit = TRUE‘ (with ‘perm_refit_spec‘) to always refit.

‘batch_assoc‘ contains chi-square tests between fold assignment and each ‘batch_cols‘ variable (‘stat‘, ‘df‘, ‘pval‘, ‘cramer_v‘). ‘target_assoc‘ reports feature-wise outcome associations on ‘X_ref‘; numeric features use AUC (binomial), ‘eta_sq‘ (multiclass), or correlation (gaussian), while categorical features use Cramer’s V (binomial/multiclass) or ‘eta_sq‘ from a one-way ANOVA (gaussian). The ‘score‘ column is the scaled effect size used for heuristic flagging (‘flag = score >= target_threshold‘). When ‘target_p_adjust != "none"‘, finite ‘p_value‘ entries also receive multiplicity-adjusted ‘p_value_adj‘ and ‘flag_fdr = (p_value_adj <= target_alpha)‘. The univariate target leakage scan can miss multivariate proxies, interaction leakage, or features not included in ‘X_ref‘. The multivariate scan (enabled by default for supported tasks) adds a model-based proxy check but still only covers features present in ‘X_ref‘.

Duplicate detection compares rows of ‘X_ref‘ using the chosen ‘sim_method‘ (cosine on L2-normalized rows, or Pearson via row-centering), optionally after rank transformation (‘feature_space = "rank"‘). By default, ‘duplicate_scope = "train_test"‘ filters to pairs that appear in train vs test in at least one repeat; set ‘duplicate_scope = "all"‘ to include within-fold duplicates. The ‘duplicates‘ slot returns index pairs and similarity values for near-duplicate samples. Only duplicates present in ‘X_ref‘ can be detected, and checks are skipped if inputs cannot be aligned to splits.

Value

A `LeakAudit` S4 object containing:

`fit` The `LeakFit` object that was audited.

`permutation_gap` One-row data.frame from the permutation-gap test with columns: `metric_obs` (observed cross-validated metric), `perm_mean` (mean of permuted metrics), `perm_sd` (standard deviation), `gap` (observed minus permuted mean, or vice versa for loss metrics), `z` (standardized gap), `p_value` (permutation p-value), and `n_perm` (number of permutations). A large positive gap and small p-value suggest the model captures signal beyond random label assignment.

`perm_values` Numeric vector of length `B` containing the metric value from each permutation. Useful for plotting the null distribution. Empty if `return_perm = FALSE`.

`batch_assoc` Data.frame of chi-square association tests between fold assignment and batch/study metadata, with columns: `variable`, `stat` (chi-square statistic), `df` (degrees of freedom), `pval`, and `cramer_v` (effect size). Small p-values indicate potential confounding by design.

`target_assoc` Data.frame of per-feature outcome associations with columns: `feature`, `type` (`"numeric"` or `"categorical"`), `metric` (AUC, correlation, eta_sq, or Cramer's V depending on task), `value`, `score` (scaled effect size), `p_value`, `n`, and `flag` (TRUE if `score >= target_threshold`). Flagged features may indicate target leakage.

`duplicates` Data.frame of near-duplicate sample pairs with columns: `i`, `j` (row indices in `X_ref`), `sim` (similarity value), and `cross_fold` (whether the pair spans train vs test). Duplicates across folds can inflate performance.

`trail` List capturing audit parameters and intermediate results for reproducibility, including `metric`, `B`, `seed`, `perm_stratify`, `perm_refit`, and timing info.

`info` List with additional metadata including multivariate scan results when `target_scan_multivariate = TRUE`.

Use `summary()` to print a human-readable report. For programmatic access to slot contents, the recommended interface is the set of S4 accessor methods registered for `LeakAudit`:

- `audit_perm_gap(audit)` – permutation-gap test data frame (the `permutation_gap` slot).
- `audit_batch_assoc(audit)` – batch / study chi-squared association data frame (`batch_assoc`).
- `audit_target_assoc(audit)` – per-predictor target-association scan results (`target_assoc`).
- `audit_duplicates(audit)` – near-duplicate sample-pair data frame (`duplicates`).
- `audit_info(audit)` – auxiliary information list including the multivariate target-scan results (`info`).

The remaining slots (`fit`, `perm_values`, `trail`) do not have dedicated accessors; the full list of accessor methods for the class is available through `methods(class = "LeakAudit")`.

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = rbinom(12, 1, 0.5),
  x1 = rnorm(12),
  x2 = rnorm(12)
)
```

```
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject", v = 3,
                      progress = FALSE)

custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = as.data.frame(x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object,
                                newdata = as.data.frame(newdata),
                                type = "response"))
    }
  )
)

fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)

audit <- audit_leakage(fit, metric = "auc", B = 10,
                       X_ref = df[, c("x1", "x2")])
```

`audit_leakage_by_learner` *Audit leakage per learner*

Description

Runs [audit_leakage()] separately for each learner recorded in a [LeakFit] and returns a named list of [LeakAudit] objects. Use this when a single fit contains predictions for multiple models and you want model-specific audits. If predictions do not include learner IDs, only a single audit can be run and requesting multiple learners is an error.

Usage

```
audit_leakage_by_learner(
  fit,
 metric = c("auc", "pr_auc", "accuracy", "macro_f1", "log_loss", "rmse", "cindex"),
  learners = NULL,
  parallel_learners = FALSE,
  mc.cores = NULL,
  ...
)
```

Arguments

`fit`	A [LeakFit] object produced by [fit_resample()]. It must contain predictions and split metadata. Learner IDs must be present in predictions to audit multiple models.
`metric`	Character scalar. One of '"auc"', '"pr_auc"', '"accuracy"', '"macro_f1"', '"log_loss"', '"rmse"', or '"cindex"'. Controls which metric is audited for each learner.
`learners`	Character vector or NULL. If NULL (default), audits all learners found in predictions. If provided, must match learner IDs stored in the predictions. Supplying more than one learner requires learner IDs.
`parallel_learners`	Logical scalar. If TRUE, runs per-learner audits in parallel using 'future.apply' (if installed). This changes runtime but not the audit results.
`mc.cores`	Integer scalar or NULL. Number of workers used when 'parallel_learners = TRUE'. Defaults to the minimum of available cores and the number of learners.
`...`	Additional named arguments forwarded to [audit_leakage()] for each learner. These control the audit itself. Common options include: 'B' (integer permutations), 'perm_stratify' (logical or '"auto"'), 'perm_refit' (logical), 'perm_refit_spec' (list), 'time_block' (character), 'block_len' (integer or NULL), 'include_z' (logical), 'ci_method' (character), 'boot_B' (integer), 'parallel' (logical), 'seed' (integer), 'return_perm' (logical), 'batch_cols' (character vector), 'coldata' (data.frame), 'X_ref' (matrix/data.frame), 'target_scan' (logical), 'target_threshold' (numeric), 'target_p_adjust' (character), 'target_alpha' (numeric), 'feature_space' (character), 'sim_method' (character), 'sim_threshold' (numeric), 'nn_k' (integer), 'max_pairs' (integer), and 'duplicate_scope' (character). See [audit_leakage()] for full definitions; changing these values changes each learner's audit.

Value

A named list of `LeakAudit` objects, where each element is keyed by the learner ID (character string). Each `LeakAudit` object contains the same slots as described in `audit_leakage`: `fit`, `permutation_gap`, `perm_values`, `batch_assoc`, `target_assoc`, `duplicates`, `trail`, and `info`. Use `names()` to retrieve learner IDs, and access individual audits with `[[learner_id]]` or `$learner_id`. Each audit reflects the performance and diagnostics for that specific learner's predictions.

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = factor(rep(c(0, 1), 6)),
  x1 = rnorm(12),
  x2 = rnorm(12)
)
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject",
                      v = 3, progress = FALSE)
custom <- list(
  glm = list(
```

```
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = data.frame(y = y, x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object,
                                newdata = as.data.frame(newdata),
                                type = "response"))
    }
  )
)
custom$glm2 <- custom$glm
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = c("glm", "glm2"), custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)
audits <- audit_leakage_by_learner(fit, metric = "auc", B = 10,
                                   perm_stratify = FALSE)
names(audits)
```

`audit_perm_gap` *Permutation-gap test results from a LeakAudit*

Description

Returns the permutation-gap test data frame stored in a [‘LeakAudit‘] object. Columns include the observed metric, permuted-null mean and SD, gap, z-score, and permutation p-value.

Usage

```
audit_perm_gap(audit)

## S4 method for signature 'LeakAudit'
audit_perm_gap(audit)
```

Arguments

`audit`	A [‘LeakAudit‘] object returned by [audit_leakage()].

Details

Implemented as an S4 generic with a method for [‘LeakAudit‘]; visible via ‘methods(class = "LeakAudit")‘.

Value

A ‘data.frame‘ with one row per (mechanism class, repeat), summarising the permutation-gap test.

See Also

[LeakClasses], [audit_leakage()], [audit_target_assoc()]

`audit_report` *Render an HTML audit report*

Description

Creates an HTML report that summarizes a leakage audit for a resampled model. The report is built from a [LeakAudit] (or created from a [LeakFit]) and presents: cross-validated metric summaries, a label-permutation association test of the chosen performance metric (auto-refit when refit data are available; otherwise fixed predictions), batch or study association tests between metadata and predictions, confounder sensitivity plots, calibration checks for binomial tasks, a target leakage scan based on feature-outcome similarity (with multivariate scan enabled by default for supported tasks), and duplicate detection across training and test folds. The output is a self-contained HTML file with tables and plots for these checks plus the audit parameters used.

Usage

```
audit_report(
  audit,
  output_file = "bioLeak_audit_report.html",
  output_dir = tempdir(),
  quiet = TRUE,
  open = FALSE,
  ...
)
```

Arguments

`audit`	A [LeakAudit] object from [audit_leakage()] or a [LeakFit] object from [fit_resample()]. If a [LeakAudit] is supplied, the report uses its stored results verbatim. If a [LeakFit] is supplied, ‘audit_report()‘ first computes a new audit via [audit_leakage(...)]; the fit must contain predictions and split metadata. When multiple learners were fit, pass a ‘learner‘ argument via ‘...‘ to select a single model.
`output_file`	Character scalar. File name for the HTML report. Defaults to ‘"bioLeak_audit_report.html"‘. If a relative name is provided, it is created inside ‘output_dir‘. Changing this value only changes the file name, not the audit content.
`output_dir`	Character scalar. Directory path where the report is written. Defaults to [tempdir()]. The directory must exist or be creatable by ‘rmarkdown::render()‘. Changing this value only changes the output location.
`quiet`	Logical scalar passed to ‘rmarkdown::render()‘. Defaults to ‘TRUE‘. When ‘FALSE‘, knitting output and warnings are printed to the console. This does not change audit results.
`open`	Logical scalar. Defaults to ‘FALSE‘. When ‘TRUE‘, opens the generated report in a browser via [utils::browseURL()]. This does not change the report contents.

... Additional named arguments forwarded to [audit_leakage()] only when ‘audit‘ is a [LeakFit]. These control how the audit is computed and therefore change the report. Typical examples include ‘metric‘ (character), ‘B‘ (integer), ‘perm_stratify‘ (logical), ‘batch_cols‘ (character vector), ‘X_ref‘ (matrix/data.frame), ‘sim_method‘ (character), and ‘duplicate_scope‘ (character). When omitted, [audit_leakage()] defaults are used. Ignored when ‘audit‘ is already a [LeakAudit].

Details

The report does not refit models or reprocess data unless ‘perm_refit‘ triggers refitting (‘TRUE‘ or ‘"auto"‘ with a valid ‘perm_refit_spec‘); it otherwise inspects the predictions and metadata stored in the input. Results are conditional on the provided splits, selected metric, and any feature matrix supplied to [audit_leakage()]. The univariate target leakage scan can miss multivariate proxies, interaction leakage, or features not included in ‘X_ref‘; the multivariate scan (enabled by default for supported tasks) adds a model-based check but still only uses features in ‘X_ref‘. A non-significant result does not prove the absence of leakage, especially with small ‘B‘ or incomplete metadata. Rendering requires the ‘rmarkdown‘ package and ‘ggplot2‘ for plots.

Value

Character string containing the absolute file path to the generated HTML report. The report is a self-contained HTML file that can be opened in any web browser. It includes sections for: cross-validated metric summaries, label-permutation test results (gap, p-value), batch/study association tests, confounder sensitivity analysis, calibration diagnostics (for binomial tasks), target leakage scan results, and duplicate detection findings. The path can be used with `browseURL` to open the report programmatically.

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = factor(rep(c(0, 1), 6)),
  x1 = rnorm(12),
  x2 = rnorm(12)
)

splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject",
                      v = 3, progress = FALSE)

custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = data.frame(y = y, x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object,
                                newdata = as.data.frame(newdata),
```

```
                                type = "response"))
    }
  )
)

fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)

audit <- audit_leakage(fit, metric = "auc", B = 5, perm_stratify = FALSE)

if (requireNamespace("rmarkdown", quietly = TRUE) &&
    requireNamespace("ggplot2", quietly = TRUE) &&
    isTRUE(rmarkdown::pandoc_available("1.12.3"))) {
  out_file <- audit_report(audit, output_dir = tempdir(), quiet = TRUE)
  out_file
}
```

`audit_target_assoc` *Target-leakage scan results from a LeakAudit*

Description

Returns the target-association data frame stored in a [‘LeakAudit‘] object. Each row is one predictor with its association score (rescaled AUC; ‘|AUC - 0.5| * 2‘), threshold-based flag, and (where applicable) p-value.

Usage

```
audit_target_assoc(audit)

## S4 method for signature 'LeakAudit'
audit_target_assoc(audit)
```

Arguments

`audit`	A [‘LeakAudit‘] object returned by [audit_leakage()].

Details

Implemented as an S4 generic with a method for [‘LeakAudit‘]; visible via ‘methods(class = "LeakAudit")‘.

Value

A ‘data.frame‘ with one row per predictor.

See Also

[LeakClasses], [audit_leakage()]

benchmark_leakage_suite *Simulation benchmark matrix for leakage diagnostics*

Description

Runs a reproducible grid of simulation scenarios across modalities, leakage mechanisms, and split modes using [simulate_leakage_suite()]. This function is designed as a benchmarking harness to quantify detection rates and performance inflation under controlled settings.

Usage

```
benchmark_leakage_suite(
  modalities = c("omics", "imaging_tabular", "ehr_tabular"),
  leakages = c("none", "subject_overlap", "batch_confounded", "peek_norm", "lookahead"),
  modes = c("subject_grouped", "batch_blocked", "time_series"),
  learner = c("glmnet", "ranger"),
  seeds = 1:5,
  B = 200,
  alpha = 0.05,
  parallel = FALSE
)
```

Arguments

modalities	Character vector selecting predefined modality profiles. Supported values: ‘"omics"‘, ‘"imaging_tabular"‘, ‘"ehr_tabular"‘.
leakages	Character vector of leakage mechanisms passed to [simulate_leakage_suite()].
modes	Character vector of split modes passed to [simulate_leakage_suite()].
learner	Character scalar. ‘"glmnet"‘ (default) or ‘"ranger"‘.
seeds	Integer vector of Monte Carlo seeds.
B	Integer scalar. Number of permutations per scenario.
alpha	Numeric scalar in (0, 1). Detection threshold applied to permutation p-values.
parallel	Logical scalar. If TRUE, evaluates scenarios in parallel when ‘future.apply‘ is available.

Value

A data.frame with one row per simulation seed/scenario and columns: ‘modality‘, ‘leakage‘, ‘mode‘, ‘seed‘, observed metric, gap, p-value, and a logical ‘detected‘ flag. A scenario-level summary is attached as ‘attr(x, "summary")‘.

`calibration_summary` *Calibration diagnostics for binomial predictions*

Description

Computes reliability curve summaries and calibration metrics for a binomial [LeakFit] using out-of-fold predictions.

Usage

```
calibration_summary(fit, bins = 10, min_bin_n = 5, learner = NULL)
```

Arguments

`fit`	A [LeakFit] object from [fit_resample()].
`bins`	Integer number of probability bins for the calibration curve.
`min_bin_n`	Minimum samples per bin used in plotting; bins smaller than this are retained in the output but can be filtered by the caller.
`learner`	Optional character scalar. When predictions include multiple learners, selects the learner to summarize.

Value

A list with a ‘curve‘ data.frame and a one-row ‘metrics‘ data.frame containing ECE, MCE, and Brier score.

Examples

```
set.seed(42)
df <- data.frame(
  subject = rep(1:15, each = 2),
  outcome = factor(rep(c(0, 1), 15)),
  x1 = rnorm(30),
  x2 = rnorm(30)
)
splits <- make_split_plan(df, outcome = "outcome",
                          mode = "subject_grouped", group = "subject",
                          v = 3, progress = FALSE)
custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = as.data.frame(x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object, newdata = as.data.frame(newdata),
                                type = "response"))
    }
```

```
  )
)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)
cal <- calibration_summary(fit, bins = 5)
cal$metrics
```

`check_split_overlap` *Check split overlap invariants*

Description

Verifies that a `LeakSplits` object satisfies the expected no-overlap constraints for one or more grouping columns. For each fold, the function checks that no group-level value appearing in the test set is also present in the training set.

Usage

```
check_split_overlap(splits, coldata = NULL, cols = NULL, stop_on_fail = TRUE)
```

Arguments

`splits`	A `LeakSplits` object from `make_split_plan`.
`coldata`	A data.frame of sample metadata. When `NULL` (default), the function uses `splits@info$coldata` if available.
`cols`	Character vector of column names to check for overlap. When `NULL` (default), the function infers columns from the split mode (e.g., `group` for `subject_grouped`, `batch` for `batch_blocked`, both axes for `combined`).
`stop_on_fail`	Logical; if `TRUE` (default), raises an error when any overlap is detected.

Value

A data.frame with one row per fold-by-column combination and columns `fold`, `repeat_id`, `col`, `n_overlap` (number of overlapping group values), and `pass` (logical). Invisible. Raises an error if any fold fails and `stop_on_fail = TRUE`.

confounder_sensitivity *Confounder sensitivity summaries*

Description

Computes performance metrics within confounder strata to surface potential confounding. Requires aligned metadata in ‘coldata‘.

Usage

```
confounder_sensitivity(
  fit,
  confounders = NULL,
  metric = NULL,
  min_n = 10,
  coldata = NULL,
  numeric_bins = 4,
  learner = NULL,
  strict_align = FALSE
)
```

Arguments

fit	A [LeakFit] object from [fit_resample()].
confounders	Character vector of columns in ‘coldata‘ to evaluate. Defaults to common batch/study identifiers when available.
metric	Metric name to compute within each stratum. Defaults to the first metric used in the fit (or task defaults if unavailable).
min_n	Minimum samples per stratum; smaller strata return NA metrics.
coldata	Optional data.frame of sample metadata. Defaults to ‘fit@splits@info$coldata‘ when available.
numeric_bins	Integer number of quantile bins for numeric confounders with many unique values.
learner	Optional character scalar. When predictions include multiple learners, selects the learner to summarize.
strict_align	Logical scalar. If TRUE, errors when coldata cannot be aligned by row names or IDs and would fall back to row-order matching. Default is FALSE.

Value

A data.frame with per-confounder, per-level metrics and counts.

Examples

```
set.seed(42)
df <- data.frame(
  subject = rep(1:15, each = 2),
  outcome = factor(rep(c(0, 1), 15)),
  batch = factor(rep(c("A", "B", "C"), 10)),
  x1 = rnorm(30),
  x2 = rnorm(30)
)
splits <- make_split_plan(df, outcome = "outcome",
                          mode = "subject_grouped", group = "subject",
                          v = 3, progress = FALSE)
custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = as.data.frame(x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object, newdata = as.data.frame(newdata),
                                type = "response"))
    }
  )
)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)
confounder_sensitivity(fit, confounders = "batch", coldata = df)
```

---

cv_ci *Confidence intervals for cross-validated metrics*

Description

Computes per-learner confidence intervals for each metric column in a per-fold metrics data.frame. Supports the standard normal/t approach and the Nadeau-Bengio (2003) corrected variance for repeated K-fold CV.

Usage

```
cv_ci(
  metrics_df,
  level = 0.95,
  method = c("normal", "nadeau_bengio"),
  n_train = NULL,
  n_test = NULL
)
```

Arguments

metrics_df	Data.frame with columns fold, learner, and one or more numeric metric columns.
level	Confidence level (default 0.95).
method	One of "normal" or "nadeau_bengio".
n_train	Average number of training samples per fold. Used only when method = "nadeau_bengio". NULL to use fallback variance.
n_test	Average number of test samples per fold. Used only when method = "nadeau_bengio". NULL to use fallback variance.

Value

A data.frame with learner and, for each metric, columns <metric>_mean, <metric>_sd, <metric>_ci_lo, and <metric>_ci_hi.

---

delta_lsi *Delta Leakage Sensitivity Index (Delta LSI)*

Description

Compares a naive (potentially leaky) cross-validation pipeline against a guarded (leakage-protected) pipeline and quantifies leakage-induced performance inflation using the Leakage Sensitivity Index (LSI).

Usage

```
delta_lsi(
  fit_leaky,
  fit_guarded,
  metric = "auc",
  exchangeability = c("iid", "by_group", "within_batch", "blocked_time"),
  learner = NULL,
  higher_is_better = NULL,
  block_size = NULL,
  M_boot = 2000L,
  M_flip = 10000L,
  strict = FALSE,
  return_details = FALSE,
  seed = 42L,
  ...
)
```

Arguments

fit_leaky	A LeakFit object from the leaky (unprotected) evaluation pipeline.
fit_guarded	A LeakFit object from the guarded (leakage-protected) evaluation pipeline.
metric	Character. Performance metric to compare. Must appear in fit@metrics of both fits (e.g., "auc", "rmse").
exchangeability	Character. Exchangeability assumption for the sign-flip test. One of "iid" (default), "by_group", "within_batch", "blocked_time". "blocked_time" activates a block sign-flip procedure that flips contiguous groups of repeats together, preserving serial autocorrelation under the null; see block_size. "by_group" and "within_batch" are stored and reported but inference still uses the iid sign-flip procedure (a warning is issued; contributions welcome). "iid" (default) applies the standard independent sign-flip test.
learner	Optional character. Learner name to select from multi-learner fits. If NULL, the first learner found in fit@metrics is used.
higher_is_better	Logical or NULL. Whether a higher value of metric indicates better performance. When NULL (default), auto-detected from the metric name: "rmse", "mse", "mae", "log_loss", "brier", "error", "loss", and "deviance" are treated as lower-is-better; all others default to higher-is-better. Setting this correctly ensures that a positive delta_lsi always indicates leakage inflation (the naive pipeline is artificially more optimistic than the guarded one).
block_size	Integer or NULL. Block length for the block sign-flip test, used only when exchangeability = "blocked_time". When NULL (default), the block size is auto-estimated from the first-order autocorrelation of $\{\Delta_r\}$ and capped at floor(R/3) to ensure at least three independent blocks. A warning is issued when the estimate is used with R_eff < 20 because the AR(1) estimate is noisy at small sample sizes. Provide an explicit integer to override auto-estimation.
M_boot	Integer. Number of bootstrap samples for BCa CI (default 2000).
M_flip	Integer. Maximum Monte Carlo samples for sign-flip test when R_eff > 15 (default 10000).
strict	Logical. If TRUE, error on insufficient R_eff instead of a warning.
return_details	Logical. If TRUE, include the per-repeat Δ_r vector and the original fit objects in the info slot.
seed	Integer. Random seed for bootstrap and sign-flip test.
...	Unused. Reserved for deprecated aliases such as fit_naive.

Details

Method: For each fit, per-fold metric values are extracted from fit@metrics (or recomputed from fit@predictions if necessary). Fold test-set sizes are used as weights to aggregate fold metrics into per-repeat estimates μ_r. The repeat-level delta $\Delta_r = s \cdot (\mu_r^{\text{naive}} - \mu_r^{\text{guarded}})$ captures leakage-induced performance inflation for each CV repeat, where $s = +1$ for higher-is-better metrics (e.g., AUC) and $s = -1$ for lower-is-better metrics (e.g., RMSE), so that $\Delta_r > 0$ always indicates the naive pipeline is more optimistic than the guarded one.

The **delta_lsi** point estimate is the Huber M-estimator (k = 1.345) applied to $\{\Delta_r\}$, which is robust to occasional outlier repeats. **delta_metric** is the arithmetic mean of $\{\Delta_r\}$ for easy interpretation in the original metric's units.

Pairing requires that `fit_leaky` and `fit_guarded` share *identical fold structures* (same test-set membership per fold) in addition to the same number of repeats. When repeat counts match but fold structures differ, a warning is issued and the fits are treated as unpaired.

When $R_{\text{eff}} \geq 5$ (equal, paired repeats), a sign-flip randomization test (Phipson & Smyth, 2010) is performed: under H_0 (no leakage) the sign of each Δ_r is exchangeable. All 2^R sign combinations are enumerated exactly for $R \leq 15$ (no continuity correction); Monte Carlo sampling is used for larger R with the Phipson & Smyth (2010) correction.

BCa bootstrap confidence intervals (Efron, 1987) require $R_{\text{eff}} \geq 10$.

Inference tiers:

`"A_full_inference"` R_eff >= 20: point + BCa CI + sign-flip p-value; `inference_ok = TRUE`

`"B_signflip_ci"` 10 <= R_eff < 20: point + sign-flip p-value + BCa CI

`"C_signflip"` 5 <= R_eff < 10: point + sign-flip p-value (no CI)

`"D_insufficient"` R_eff < 5 or unpaired: point estimate only

Value

A `LeakDeltaLSI` object. Use `summary()` to print a formatted report. For programmatic access to inflation estimates, confidence intervals, p-values, tier labels, and the per-repeat fold-level deltas, use the S4 accessor methods registered for `LeakDeltaLSI`:

- `dlsi_metric(dlsi)` – raw mean of per-repeat metric differences (the `delta_metric` slot).
- `dlsi_robust(dlsi)` – Huber-robust point estimate (the `delta_lsi` slot).
- `dlsi_ci(dlsi, which = "robust" | "metric")` – BCa bootstrap confidence interval for either the robust or the raw estimate.
- `dlsi_p_value(dlsi)` – sign-flip randomization-test p-value.
- `dlsi_tier(dlsi)` – inference-tier label (`"A_full_inference"`, `"B_signflip_ci"`, `"C_signflip"`, or `"D_insufficient"`).
- `dlsi_R_eff(dlsi)` – effective number of paired repeats contributing to the inference.
- `dlsi_repeats(dlsi, which = "naive" | "guarded")` – per-repeat metric data frame for the requested pipeline.

The full list of accessor methods for the class is available through `methods(class = "LeakDeltaLSI")`.

See Also

`audit_leakage`, `fit_resample`, `LeakDeltaLSI`

dlsi_ci *BCa confidence interval from a LeakDeltaLSI*

Description

Returns the bias-corrected and accelerated (BCa) bootstrap confidence interval stored in a [‘LeakDeltaLSI‘] object. By default returns the interval for the Huber-robust delta estimate; set ‘which = "metric"‘ to return the interval for the raw metric difference instead.

Usage

```
dlsi_ci(dlsi, which = c("robust", "metric"))

## S4 method for signature 'LeakDeltaLSI'
dlsi_ci(dlsi, which = c("robust", "metric"))
```

Arguments

dlsi	A [‘LeakDeltaLSI‘] object returned by [delta_lsi()].
which	Either ‘"robust"‘ (default) for the Huber estimate’s confidence interval, or ‘"metric"‘ for the raw arithmetic mean’s confidence interval.

Details

Implemented as an S4 generic with a method for [‘LeakDeltaLSI‘]; visible via ‘methods(class = "LeakDeltaLSI")‘.

Value

A length-two numeric vector ‘c(lower, upper)‘. Returns ‘c(NA_real_, NA_real_)‘ when the interval is not computed (for example, when the inference tier did not include CIs).

See Also

[LeakClasses], [delta_lsi()], [dlsi_robust()], [dlsi_metric()]

dlsi_metric *Raw delta-metric estimate from a LeakDeltaLSI*

Description

Returns the arithmetic mean of the per-repeat raw metric differences (leaky minus guarded) stored in a [‘LeakDeltaLSI‘] object.

Usage

```
dlsi_metric(dlsi)

## S4 method for signature 'LeakDeltaLSI'
dlsi_metric(dlsi)
```

Arguments

dlsi	A [‘LeakDeltaLSI‘] object returned by [delta_lsi()].

Details

Implemented as an S4 generic with a method for [‘LeakDeltaLSI‘]; visible via ‘methods(class = "LeakDeltaLSI")‘.

Value

A length-one numeric scalar.

See Also

[LeakClasses], [delta_lsi()], [dlsi_robust()], [dlsi_ci()]

---

dlsi_p_value *Sign-flip p-value from a LeakDeltaLSI*

Description

Returns the paired sign-flip randomization-test p-value stored in a [‘LeakDeltaLSI‘] object.

Usage

```
dlsi_p_value(dlsi)

## S4 method for signature 'LeakDeltaLSI'
dlsi_p_value(dlsi)
```

Arguments

dlsi	A [‘LeakDeltaLSI‘] object returned by [delta_lsi()].

Details

Implemented as an S4 generic with a method for [‘LeakDeltaLSI‘]; visible via ‘methods(class = "LeakDeltaLSI")‘.

Value

A length-one numeric scalar in ‘[0, 1]‘, or ‘NA_real_‘ when the inference tier did not include hypothesis testing.

See Also

[LeakClasses], [delta_lsi()], [dlsi_tier()]

`dlsi_repeats` *Per-repeat metric data frames from a LeakDeltaLSI*

Description

Returns the per-repeat metric data frame for one of the two pipelines stored in a [‘LeakDeltaLSI‘] object. The naive (or leaky) pipeline’s repeats are returned by default; set ‘which = "guarded"‘ to return the guarded pipeline’s repeats.

Usage

```
dlsi_repeats(dlsi, which = c("naive", "guarded"))

## S4 method for signature 'LeakDeltaLSI'
dlsi_repeats(dlsi, which = c("naive", "guarded"))
```

Arguments

`dlsi`	A [‘LeakDeltaLSI‘] object returned by [delta_lsi()].
`which`	Either ‘"naive"‘ (default) for the naive/leaky pipeline’s per-repeat data frame, or ‘"guarded"‘ for the guarded pipeline’s per-repeat data frame.

Details

Implemented as an S4 generic with a method for [‘LeakDeltaLSI‘]; visible via ‘methods(class = "LeakDeltaLSI")‘.

Value

A ‘data.frame‘ with one row per repeat.

See Also

[LeakClasses], [delta_lsi()]

dlsi_robust *Huber-robust delta_lsi point estimate from a LeakDeltaLSI*

Description

Returns the Huber-robust point estimate of the per-repeat delta values stored in a [‘LeakDeltaLSI‘] object.

Usage

```
dlsi_robust(dlsi)

## S4 method for signature 'LeakDeltaLSI'
dlsi_robust(dlsi)
```

Arguments

dlsi	A [‘LeakDeltaLSI‘] object returned by [delta_lsi()].

Details

Implemented as an S4 generic with a method for [‘LeakDeltaLSI‘]; visible via ‘methods(class = "LeakDeltaLSI")‘.

Value

A length-one numeric scalar.

See Also

[LeakClasses], [delta_lsi()], [dlsi_metric()], [dlsi_ci()]

dlsi_R_eff *Effective number of paired repeats from a LeakDeltaLSI*

Description

Returns the effective number of paired repeats ‘R_eff‘ stored in a [‘LeakDeltaLSI‘] object. This is the count of repeats that contribute to the inference; it equals the smaller of the leaky and guarded fits’ repeat counts when the comparison is paired.

Usage

```
dlsi_R_eff(dlsi)

## S4 method for signature 'LeakDeltaLSI'
dlsi_R_eff(dlsi)
```

Arguments

dlsi	A [‘LeakDeltaLSI‘] object returned by [delta_lsi()].

Details

Implemented as an S4 generic with a method for [‘LeakDeltaLSI‘]; visible via ‘methods(class = "LeakDeltaLSI")‘.

Value

A length-one integer.

See Also

[LeakClasses], [delta_lsi()], [dlsi_tier()]

---

dlsi_tier *Inference tier from a LeakDeltaLSI*

Description

Returns the inference tier label stored in a [‘LeakDeltaLSI‘] object. Possible values are ‘"A_full_inference"‘ (‘R_eff >= 20‘), ‘"B_signflip_ci"‘ (‘R_eff >= 10‘), ‘"C_signflip"‘ (‘R_eff >= 5‘), or ‘"D_insufficient"‘ (‘R_eff < 5‘).

Usage

```
dlsi_tier(dlsi)

## S4 method for signature 'LeakDeltaLSI'
dlsi_tier(dlsi)
```

Arguments

dlsi	A [‘LeakDeltaLSI‘] object returned by [delta_lsi()].

Details

Implemented as an S4 generic with a method for [‘LeakDeltaLSI‘]; visible via ‘methods(class = "LeakDeltaLSI")‘.

Value

A length-one character string giving the tier label.

See Also

[LeakClasses], [delta_lsi()], [dlsi_R_eff()]

`fit_metrics` *Per-fold metric data frame from a LeakFit*

Description

Returns the per-fold metric data frame stored in a [‘LeakFit‘] object. Each row is one (fold, repeat, learner) combination; columns include the requested metric values such as ‘auc‘ and any task-specific performance scores.

Usage

```
fit_metrics(fit)

## S4 method for signature 'LeakFit'
fit_metrics(fit)
```

Arguments

`fit`	A [‘LeakFit‘] object returned by [fit_resample()].

Details

Implemented as an S4 generic with a method for [‘LeakFit‘]; visible via ‘methods(class = "LeakFit")‘.

Value

A ‘data.frame‘ with one row per (fold, repeat, learner) combination.

See Also

[LeakClasses], [fit_resample()], [audit_perm_gap()]

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = factor(rep(c("a","b"), 6), levels = c("a","b")),
  x1 = rnorm(12), x2 = rnorm(12)
)
splits <- make_split_plan(df, outcome = "outcome",
                          mode = "subject_grouped", group = "subject", v = 3)
## Not run:
  fit <- fit_resample(df, outcome = "outcome", splits = splits,
                      learner = parsnip::logistic_reg() |>
                                parsnip::set_engine("glm"),
                      metrics = "auc")
  fit_metrics(fit)
```

```
## End(Not run)
```

`fit_resample` *Fit and evaluate with leakage guards over predefined splits*

Description

Performs cross-validated model training and evaluation using leakage-protected preprocessing (guard_fit) and user-specified learners.

Usage

```
fit_resample(
  x,
  outcome,
  splits,
  preprocess = list(impute = list(method = "median"), normalize = list(method =
   "zscore"), filter = list(var_thresh = 0, iqr_thresh = 0), fs = list(method = "none")),
  learner = c("glmnet", "ranger"),
  learner_args = list(),
  custom_learners = list(),
  metrics = c("auc", "pr_auc", "accuracy"),
  class_weights = NULL,
  positive_class = NULL,
  classification_threshold = 0.5,
  parallel = FALSE,
  refit = TRUE,
  seed = 1,
  split_cols = "auto",
  store_refit_data = TRUE
)
```

Arguments

`x`	SummarizedExperiment or matrix/data.frame
`outcome`	outcome column name (if x is SE or data.frame), or a length-2 character vector of time/event column names for survival outcomes.
`splits`	LeakSplits object from make_split_plan(), or an ‘rsample‘ rset/rsplit.
`preprocess`	list(impute, normalize, filter=list(...), fs) or a ‘recipes::recipe‘ object. When a recipe is supplied, the guarded preprocessing pipeline is bypassed and the recipe is prepped on training data only. Recipe/workflow leakage guardrails run before fitting; configure policy via `options(bioLeak.validation_mode = "warn" | "error" | "off")`.
`learner`	parsnip model_spec (or list of model_spec objects) describing the model(s) to fit, or a ‘workflows::workflow‘. For legacy use, a character vector of learner names (e.g., "glmnet", "ranger") or custom learner IDs is still supported.

`learner_args` list of additional arguments passed to legacy learners (ignored when ‘learner‘ is a parsnip model_spec).

`custom_learners`
named list of custom learner definitions used only with legacy character learners. Each entry must contain `fit` and `predict` functions. The `fit` function should accept `x`, `y`, `task`, and `weights`, and return a model object. The `predict` function should accept `object`, `newdata`, and `task`. For binomial/regression/survival tasks it should return a numeric vector; for multiclass tasks it should return either class labels or a matrix/data.frame of class probabilities.

`metrics` named list of metric functions, vector of metric names, or a ‘yardstick::metric_set‘. When a yardstick metric set (or list of yardstick metric functions) is supplied, metrics are computed using yardstick with the positive class set to the second factor level.

`class_weights` optional named numeric vector of weights for binomial or multiclass outcomes

`positive_class` optional value indicating the positive class for binomial outcomes. When set, the outcome levels are reordered so that `positive_class` is treated as the positive class (level 2). If NULL, the second factor level is used.

`classification_threshold`
Numeric threshold in `[0, 1]` used to convert binomial probabilities into class predictions for `pred_class` and accuracy metrics. Ignored for non-binomial tasks.

`parallel` logical, use future.apply for multicore execution

`refit` logical, if TRUE retrain final model on full data

`seed` integer, for reproducibility

`split_cols` Optional named list/character vector or ‘"auto"‘ (default) overriding group/batch/study/time column names when ‘splits‘ is an rsample object and its attributes are missing. ‘"auto"‘ falls back to common metadata column names (e.g., ‘group‘, ‘subject‘, ‘batch‘, ‘study‘, ‘time‘). Supported names are ‘group‘, ‘batch‘, ‘study‘, and ‘time‘.

`store_refit_data`
Logical; when TRUE (default), stores the original data and learner configuration inside the fit to enable refit-based permutation tests without manual ‘perm_refit_spec‘ setup.

Details

Preprocessing is fit on the training fold and applied to the test fold, preventing leakage from global imputation, scaling, or feature selection. When a ‘recipes::recipe‘ or ‘workflows::workflow‘ is supplied, the recipe is prepped on the training fold and baked on the test fold. For data.frame or matrix inputs, columns used to define splits (outcome, group, batch, study, time) are excluded from the predictor matrix. Use `learner_args` to pass model-specific arguments, either as a named list keyed by learner or a single list applied to all learners. For custom learners, `learner_args[[name]]` may be a list with `fit` and `predict` sublists to pass distinct arguments to each stage. For binomial tasks, predictions and metrics assume the positive class is the second factor level; use `positive_class` to control this. Use `classification_threshold` to change the probability cutoff used for class labels and accuracy. Parsnip learners must support probability predictions for binomial metrics (AUC/PR-AUC/accuracy) and multiclass log-loss when requested.

Value

A LeakFit S4 object containing:

splits The LeakSplits object used for resampling.

metrics Data.frame of per-fold, per-learner performance metrics with columns fold, learner, and one column per requested metric.

metric_summary Data.frame summarizing metrics across folds for each learner with columns learner, and <metric>_mean and <metric>_sd for each requested metric.

audit Data.frame with per-fold audit information including fold, n_train, n_test, learner, and features_final (number of features after preprocessing).

predictions List of data.frames containing out-of-fold predictions with columns id (sample identifier), truth (true outcome), pred (predicted value or probability), fold, and learner. For classification tasks, includes pred_class. For multiclass, includes per-class probability columns.

preprocess List of preprocessing state objects from each fold, storing imputation parameters, normalization statistics, and feature selection results.

learners List of fitted model objects from each fold.

outcome Character string naming the outcome variable.

task Character string indicating the task type ("binomial", "multiclass", "gaussian", or "survival").

feature_names Character vector of feature names after preprocessing.

info List of additional metadata including hash, metrics_used, class_weights, positive_class, sample_ids, fold_status, refit, final_model (refitted model if refit = TRUE), final_preprocess, learner_names, and perm_refit_spec (for permutation-based audits).

Use summary() to print a formatted report. For programmatic access to slot contents, the recommended interface is the S4 accessor method registered for LeakFit:

- fit_metrics(fit) – per-fold metric data frame (the metrics slot).

Slots without dedicated accessors (predictions, info, audit, learners, preprocess, feature_names, outcome, task, splits, metric_summary) are read directly via the standard @ operator when needed; the list of all accessor methods for the class is available through methods(class = "LeakFit").

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:10, each = 2),
  outcome = rbinom(20, 1, 0.5),
  x1 = rnorm(20),
  x2 = rnorm(20)
)
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject", v = 5)

# glmnet learner (requires glmnet package)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
```

```
                      learner = "glmnet", metrics = "auc")
summary(fit)

# Custom learner (logistic regression) - no extra packages needed
custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = as.data.frame(x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
    as.numeric(stats::predict(object, newdata = as.data.frame(newdata), type = "response"))
    }
  )
)
fit2 <- fit_resample(df, outcome = "outcome", splits = splits,
                     learner = "glm", custom_learners = custom,
                     metrics = "accuracy")

summary(fit2)
```

`guard_ensure_levels` *Ensure consistent categorical levels for guarded preprocessing*

Description

Converts character/logical columns to factors and aligns factor levels with a training-time `levels_map`. Adds a dummy level when a column has only one observed level so that downstream one-hot encoding retains a column.

Usage

```
guard_ensure_levels(df, levels_map = NULL, dummy_prefix = "__dummy__")
```

Arguments

`df`	data.frame to normalize factor levels.
`levels_map`	optional named list of factor levels learned from training data.
`dummy_prefix`	prefix used when adding a dummy level to single-level factors.

Value

List with elements `data` (data.frame) and `levels` (named list of levels).

guard_fit *Fit leakage-safe preprocessing pipeline*

Description

Builds and fits a guarded preprocessing pipeline on training data, then constructs a transformer for consistent application to new data.

Usage

```
guard_fit(
  X,
  y = NULL,
  steps = list(),
  task = c("binomial", "multiclass", "gaussian", "survival")
)
```

Arguments

X	matrix/data.frame of predictors (training).
y	Optional outcome for supervised feature selection.
steps	List of configuration options (see Details).
task	"binomial", "multiclass", "gaussian", or "survival".

Details

The pipeline applies, in order:

- Winsorization (optional) to limit outliers.
- Imputation learned on training data only.
- Normalization (z-score or robust).
- Variance/IQR filtering.
- Feature selection (optional; t-test, lasso, PCA).

All statistics are estimated on the training data and re-used for new data.

Value

An object of class "GuardFit" with elements ‘transform‘, ‘state‘, ‘p_out‘, and ‘steps‘.

See Also

[predict_guard()]

Examples

```
x <- data.frame(a = c(1, 2, NA), b = c(3, 4, 5))
fit <- guard_fit(x, y = c(1, 2, 3),
                 steps = list(impute = list(method = "median")),
                 task = "gaussian")
fit$transform(x)
```

`guard_to_recipe` *Convert guard preprocessing steps to a recipes recipe*

Description

Maps bioLeak guard preprocessing steps (impute, normalize, filter, fs) to their closest **recipes** equivalents. Requires the **recipes** package. Steps that have no direct recipe equivalent are skipped with a warning.

Usage

```
guard_to_recipe(steps, formula, training_data)
```

Arguments

`steps`	A named list of guard preprocessing steps, e.g., `list(impute = list(method = "median"), normalize = list(method = "zscore"))`.
`formula`	A model formula (e.g., `outcome ~ .`).
`training_data`	A data.frame used to initialize the recipe.

Details

Mapping:

- `impute$method = "median"`: `step_impute_median(all_numeric_predictors())`
- `impute$method = "knn"`: `step_impute_knn(all_predictors(), neighbors = k)`
- `impute$method = "missForest"` or `"mice"`: Warning + `step_impute_median()` fallback
- `normalize$method = "zscore"`: `step_normalize(all_numeric_predictors())`
- `normalize$method = "robust"`: Warning + `step_normalize()` fallback
- `normalize$method = "none"`: No step added
- `filter$var_thresh > 0`: `step_nzv(all_numeric_predictors())`
- `fs$method = "pca"`: `step_pca(all_numeric_predictors(), num_comp = ncomp)`
- `fs$method = "ttest"` or `"lasso"`: Warning, skipped (no recipe equivalent)

Value

A `recipes::recipe` object with the mapped steps added.

impute_guarded *Leakage-safe data imputation via guarded preprocessing*

Description

Fits imputation parameters on the training data only, then applies the same guarded transformation to the test data. This function is a thin wrapper around the guarded preprocessing used by `fit_resample()`. Output is the transformed feature matrix used by the guarded pipeline (categorical variables are one-hot encoded).

Usage

```
impute_guarded(
  train,
  test,
  method = c("median", "knn", "missForest", "none"),
  constant_value = 0,
  k = 5,
  seed = 123,
  winsor = TRUE,
  winsor_thresh = 3,
  parallel = FALSE,
  return_outliers = FALSE,
  vars = NULL
)
```

Arguments

`train`	data frame (training set)
`test`	data frame (test set)
`method`	one of "median", "knn", "missForest", or "none"
`constant_value`	unused; retained for backward compatibility
`k`	number of neighbors for kNN imputation (if method = "knn")
`seed`	unused; retained for backward compatibility. Set seed before calling this function if reproducibility is needed.
`winsor`	logical; apply MAD-based winsorization before imputation
`winsor_thresh`	numeric; MAD cutoff (default = 3)
`parallel`	logical; unused (kept for compatibility)
`return_outliers`	logical; unused (outlier flags not returned)
`vars`	optional character vector; impute only selected variables

Value

A list (S3 class `"LeakImpute"`) with elements `train`, `test`, `model`, `method`, `summary`, and `outliers`.

See Also

[fit_resample()], [predict_guard()]

Examples

```
train <- data.frame(x = c(1, 2, NA, 4), y = c(NA, 1, 1, 0))
test <- data.frame(x = c(NA, 5), y = c(1, NA))
imp <- impute_guarded(train, test, method = "median", winsor = FALSE)
imp$train
imp$test
```

LeakSplits-class *S4 Classes for bioLeak Pipeline*

Description

These classes capture splits, model fits, and audit diagnostics produced by `make_split_plan()`, `fit_resample()`, and `audit_leakage()`.

Usage

```
## S4 method for signature 'LeakDeltaLSI'
show(object)
```

Arguments

`object`	A `LeakDeltaLSI` object.

Value

An S4 object of the respective class.

Slots

`mode` Splitting mode. One of "subject_grouped", "batch_blocked", "study_loocv", "time_series", or "combined".

`indices` List of resampling descriptors (train/test indices when available)

`info` (LeakSplits) Metadata associated with the split plan (mode, coldata, hash, etc.)

`splits` A [‘LeakSplits‘] object used for resampling

`metrics` Model performance metrics per resample

`metric_summary` Summary of metrics across resamples

`audit` Audit information per resample

`predictions` List of prediction objects

`preprocess` Preprocessing steps used during fitting

`learners` Learner definitions used in the pipeline

`outcome` Outcome variable name

`task` Modeling task name

`feature_names` Feature names included in the model

`info` (LeakFit) Metadata about the model fit (sample IDs, timings, provenance, etc.)

`fit` A ['LeakFit'] object used to generate the audit

`permutation_gap` Data frame summarising permutation gaps

`perm_values` Numeric vector of permutation-based scores

`batch_assoc` Data frame of batch associations

`target_assoc` Data frame of feature-wise outcome associations

`duplicates` Data frame detailing duplicate records

`trail` List capturing audit trail information

`info` (LeakAudit) Metadata about the audit (mechanism summary, settings, provenance, etc.)

`metric` Performance metric compared between pipelines

`exchangeability` Exchangeability assumption used for the sign-flip test

`tier` Inference tier label based on effective number of repeats

`strict` Whether strict mode was requested

`R_eff` Effective number of paired repeats available for inference

`delta_lsi` Huber-robust point estimate of repeat-level metric difference

`delta_lsi_ci` BCa 95% CI for delta_lsi (NA when R_eff < 10)

`delta_metric` Arithmetic mean of repeat-level metric differences

`delta_metric_ci` BCa 95% CI for delta_metric (NA when R_eff < 10)

`p_value` Sign-flip randomization test p-value (NA when R_eff < 5 or unpaired)

`inference_ok` TRUE when tier A (R_eff >= 20, paired, finite p and CI)

`folds_naive` Per-fold data frame for the naive pipeline

`folds_guarded` Per-fold data frame for the guarded pipeline

`repeats_naive` Per-repeat aggregate data frame for the naive pipeline

`repeats_guarded` Per-repeat aggregate data frame for the guarded pipeline

`info` (LeakDeltaLSI) Metadata including R_naive, R_guarded, paired status, and block details

Accessors (LeakFit)

The S4 accessor method [fit_metrics()] returns the per-fold metric data frame. It is listed under `methods(class = "LeakFit")` alongside the `show` and `summary` methods.

Plot method (LeakFit)

`plot(<LeakFit>)` dispatches to [plot_fold_balance()] by default. Use the `which` argument to switch to one of `"overlap"`, `"calibration"` (binary outcomes only), `"time_acf"` (time-ordered splits), or `"confounder_sensitivity"`.

Accessors (LeakAudit)

The S4 accessor methods [audit_perm_gap()], [audit_batch_assoc()], [audit_target_assoc()], [audit_duplicates()], and [audit_info()] return the corresponding slots. They are listed under `methods(class = "LeakAudit")` alongside the `show` and `summary` methods.

Plot method (LeakAudit)

`plot(<LeakAudit>)` dispatches to [plot_perm_distribution()], rendering the permutation-null distribution with the observed metric and permuted mean marked.

Accessors (LeakDeltaLSI)

The S4 accessor methods [dlsi_metric()], [dlsi_robust()], [dlsi_ci()], [dlsi_p_value()], [dlsi_tier()], [dlsi_R_eff()], and [dlsi_repeats()] return the corresponding components. They are listed under `methods(class = "LeakDeltaLSI")` alongside the `show` and `summary` methods.

Plot method (LeakDeltaLSI)

`plot(<LeakDeltaLSI>)` dispatches to [plot_dlsi_repeats()], rendering the per-repeat Δ_r scatter with the Huber-robust point estimate, the arithmetic mean, and the BCa bootstrap confidence interval band.

See Also

[make_split_plan()], [fit_resample()], [audit_leakage()]

[fit_resample()], [fit_metrics()], [plot_fold_balance()]

[audit_leakage()], [audit_report()], [audit_perm_gap()], [plot_perm_distribution()]

[delta_lsi()], [dlsi_metric()], [dlsi_ci()], [plot_dlsi_repeats()]

`make_split_plan` *Create leakage-resistant splits*

Description

Generates leakage-safe cross-validation splits for common biomedical setups: subject-grouped, batch-blocked, study leave-one-out, and time-series rolling-origin. Supports repeats, optional stratification, nested inner CV, and optional prediction horizon/purge/embargo gaps for time series. Note that splits store explicit indices, which can be memory-intensive for large `n` and many repeats.

Usage

```
make_split_plan(
  x,
  outcome = NULL,
 mode = c("subject_grouped", "batch_blocked", "study_loocv", "time_series", "combined"),
  group = NULL,
```

```
  batch = NULL,
  study = NULL,
  time = NULL,
  primary_axis = NULL,
  secondary_axis = NULL,
  constraints = NULL,
  v = 5,
  repeats = 1,
  stratify = FALSE,
  nested = FALSE,
  seed = 1,
  horizon = 0,
  purge = 0,
  embargo = 0,
  progress = TRUE,
  compact = FALSE,
  strict = TRUE
)
```

Arguments

x	SummarizedExperiment or data.frame/matrix (samples x features). If SummarizedExperiment, metadata are taken from colData(x). If data.frame, metadata are taken from x (columns referenced by group, batch, study, time, outcome).
outcome	character, outcome column name (used for stratification).
mode	one of "subject_grouped","batch_blocked","study_loocv","time_series","combined".
group	subject/group id column (for subject_grouped). Required when mode is 'subject_grouped'; use 'group = "row_id"' to explicitly request sample-wise CV.
batch	batch/plate/center column (for batch_blocked).
study	study id column (for study_loocv).
time	time column (numeric or POSIXct) for time_series.
primary_axis	List with elements type (one of "subject", "batch", "study") and col (column name). Used only when mode = "combined" to define the primary grouping axis. Deprecated in favor of constraints; still supported for backward compatibility.
secondary_axis	List with elements type and col. Used only when mode = "combined" to define the secondary constraint axis. Training sets exclude samples whose secondary-axis levels appear in the test set. Deprecated in favor of constraints; still supported for backward compatibility.
constraints	A list of constraint specifications for mode = "combined". Each element is a list with type (one of "subject", "batch", "study") and col (column name). The first element defines the primary grouping axis (fold driver); subsequent elements define exclusion constraints (training samples sharing constraint-axis levels with the test set are removed). Requires at least 2 elements. Cannot be used together with primary_axis/secondary_axis.
v	integer, number of folds (k) or rolling partitions.

repeats	integer, number of repeats (>=1) for non-LOOCV modes.
stratify	logical, keep outcome proportions similar across folds. For grouped modes, stratification is applied at the group level (by majority class per group) if outcome is provided; otherwise ignored.
nested	logical, whether to attach inner CV splits (per outer fold) using the same mode on the outer training set (with v folds, 1 repeat).
seed	integer seed.
horizon	numeric (>=0), minimal time gap for time_series so that the training set only contains samples with time < min(test_time) when horizon = 0, and time <= min(test_time) - horizon otherwise.
purge	numeric (>=0), additional gap removed immediately before each time-series test block.
embargo	numeric (>=0), additional exclusion window anchored at the end of each time-series test block. Training rows with time > max(test_time) - embargo are removed.
progress	logical, print progress for large jobs.
compact	logical; store fold assignments instead of explicit train/test indices to reduce memory usage for large datasets. Not supported when nested = TRUE.
strict	logical; deprecated and ignored. 'subject_grouped' always requires a non-NULL 'group'.

Value

A LeakSplits S4 object containing:

mode Character string indicating the splitting mode ("subject_grouped", "batch_blocked", "study_loocv", or "time_series").

indices List of fold descriptors, each containing train (integer vector of training indices), test (integer vector of test indices), fold (fold number), and repeat_id (repeat identifier). When compact = TRUE, indices are stored as fold assignments instead.

info List of metadata including outcome, v, repeats, seed, grouping columns (group, batch, study, time), stratify, nested, horizon, purge, embargo, summary (data.frame of fold sizes), hash (reproducibility checksum), inner (nested inner splits if nested = TRUE), and coldata (sample metadata).

Use the show method to print a summary; downstream access to the indices and metadata is normally done through the functions that consume a LeakSplits (for example fit_resample) rather than by reading slots directly.

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:10, each = 2),
  outcome = rbinom(20, 1, 0.5),
  x1 = rnorm(20),
  x2 = rnorm(20)
```

```
)
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject", v = 5)
```

`plot,LeakAudit,missing-method`
Plot method for LeakAudit

Description

Diagnostic plot for a [‘LeakAudit‘] object. The default diagnostic is the permutation-distribution histogram produced by `plot_perm_distribution`.

Usage

```
## S4 method for signature 'LeakAudit,missing'
plot(x, y, ...)
```

Arguments

`x`	A [‘LeakAudit‘] object.
`y`	Unused; present for S4 compatibility with `base::plot`.
`...`	Additional arguments passed to `plot_perm_distribution`.

Value

Invisibly returns the list produced by `plot_perm_distribution` (observed value, permuted mean, permutation values, ggplot object).

See Also

`plot_perm_distribution`, `LeakAudit`

`plot,LeakDeltaLSI,missing-method`
Plot method for LeakDeltaLSI

Description

Diagnostic plot for a [‘LeakDeltaLSI‘] object: per-repeat Δ_r scatter with the Huber-robust point estimate, the arithmetic mean, and the BCa bootstrap confidence interval band. This is the diagnostic shown as Figure 4 panel (b) of the manuscript.

Usage

```
## S4 method for signature 'LeakDeltaLSI,missing'
plot(x, y, ...)
```

Arguments

x	A [‘LeakDeltaLSI‘] object.
y	Unused; present for S4 compatibility with base::plot.
...	Additional arguments (currently unused).

Value

Invisibly returns the list produced by plot_dlsi_repeats: per-repeat deltas, the robust and arithmetic-mean estimates, the BCa interval, and the ggplot object.

See Also

plot_dlsi_repeats, LeakDeltaLSI

---

plot,LeakFit,missing-method

Plot method for LeakFit

Description

Diagnostic plot for a [‘LeakFit‘] object. The default diagnostic is the fold-balance check produced by plot_fold_balance, which works for any classification task. Use the which argument to switch to one of the other diagnostics in the package: "overlap" (plot_overlap_checks), "calibration" (plot_calibration; binary outcomes only), "time_acf" (plot_time_acf; time-ordered splits), or "confounder_sensitivity" (plot_confounder_sensitivity).

Usage

```
## S4 method for signature 'LeakFit,missing'
plot(
  x,
  y,
  which = c("fold_balance", "overlap", "calibration", "time_acf",
    "confounder_sensitivity"),
  ...
)
```

Arguments

x	A [‘LeakFit‘] object.
y	Unused; present for S4 compatibility with base::plot.
which	One of "fold_balance" (default), "overlap", "calibration", "time_acf", "confounder_sensitivity".
...	Additional arguments passed to the selected helper.

Value

Invisibly returns the list produced by the selected helper.

See Also

plot_fold_balance, plot_overlap_checks, plot_calibration, plot_time_acf, plot_confounder_sensitivity, LeakFit

---

plot_calibration *Plot calibration curve for binomial predictions*

Description

Visualizes observed outcome rates versus predicted probabilities across bins to diagnose calibration (binomial tasks only). Requires ggplot2.

Usage

```
plot_calibration(fit, bins = 10, min_bin_n = 5, learner = NULL)
```

Arguments

fit	LeakFit.
bins	Number of probability bins to use.
min_bin_n	Minimum samples per bin shown in the plot.
learner	Optional character scalar. When predictions include multiple learners, selects the learner to summarize.

Value

A list containing the calibration curve, metrics, and a ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  df <- data.frame(
    subject = rep(1:15, each = 2),
    outcome = factor(rep(c(0, 1), 15)),
    x1 = rnorm(30),
    x2 = rnorm(30)
  )
  splits <- make_split_plan(df, outcome = "outcome",
                            mode = "subject_grouped", group = "subject",
                            v = 3, progress = FALSE)
  custom <- list(
    glm = list(
      fit = function(x, y, task, weights, ...) {
        stats::glm(y ~ ., data = as.data.frame(x),
                   family = stats::binomial(), weights = weights)
      },
      predict = function(object, newdata, task, ...) {
        as.numeric(stats::predict(object, newdata = as.data.frame(newdata),
                                  type = "response"))
      }
    )
  )
  fit <- fit_resample(df, outcome = "outcome", splits = splits,
                      learner = "glm", custom_learners = custom,
                      metrics = "auc", refit = FALSE, seed = 1)
  plot_calibration(fit, bins = 5)
}
```

---

plot_confounder_sensitivity

Plot confounder sensitivity

Description

Shows performance metrics across confounder strata to assess sensitivity to batch/study effects. Requires ggplot2.

Usage

```
plot_confounder_sensitivity(
  fit,
  confounders = NULL,
  metric = NULL,
  min_n = 10,
  coldata = NULL,
  numeric_bins = 4,
```

```
  learner = NULL
)
```

Arguments

fit	LeakFit.
confounders	Character vector of columns in ‘coldata‘ to evaluate.
metric	Metric name to compute within each stratum.
min_n	Minimum samples per stratum to display.
coldata	Optional data.frame of sample metadata.
numeric_bins	Number of quantile bins for numeric confounders.
learner	Optional character scalar. When predictions include multiple learners, selects the learner to summarize.

Value

A list containing the sensitivity table and a ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  df <- data.frame(
    subject = rep(1:15, each = 2),
    outcome = factor(rep(c(0, 1), 15)),
    batch = factor(rep(c("A", "B", "C"), 10)),
    x1 = rnorm(30),
    x2 = rnorm(30)
  )
  splits <- make_split_plan(df, outcome = "outcome",
                            mode = "subject_grouped", group = "subject",
                            v = 3, progress = FALSE)
  custom <- list(
    glm = list(
      fit = function(x, y, task, weights, ...) {
        stats::glm(y ~ ., data = as.data.frame(x),
                   family = stats::binomial(), weights = weights)
      },
      predict = function(object, newdata, task, ...) {
        as.numeric(stats::predict(object, newdata = as.data.frame(newdata),
                                  type = "response"))
      }
    )
  )
  fit <- fit_resample(df, outcome = "outcome", splits = splits,
                      learner = "glm", custom_learners = custom,
                      metrics = "auc", refit = FALSE, seed = 1)
  plot_confounder_sensitivity(fit, confounders = "batch", coldata = df)
}
```

plot_dlsi_repeats *Plot per-repeat Δ_r values from a LeakDeltaLSI object*

Description

Visualises the per-repeat metric differences (leaky minus guarded) for a LeakDeltaLSI object, overlaid with the robust Huber point estimate, the arithmetic mean, and the BCa bootstrap confidence interval. This is the diagnostic shown as Figure 4 panel (b) of the manuscript. Requires ggplot2.

Usage

```
plot_dlsi_repeats(dlsi)
```

Arguments

dlsi	A LeakDeltaLSI object produced by delta_lsi.

Value

A list with the per-repeat deltas, the robust and arithmetic-mean estimates, the BCa confidence interval, and the ggplot object.

See Also

delta_lsi, dlsi_repeats

plot_fold_balance *Plot fold balance of class counts per fold*

Description

Displays a bar chart of class counts per fold. For binomial tasks, it also overlays the positive proportion to diagnose stratification issues. The positive class is taken from fit@info$positive_class when available; otherwise the second factor level is used. For multiclass tasks, the plot shows per-class counts without a proportion line. Only available for classification tasks. Requires ggplot2.

Usage

```
plot_fold_balance(fit)
```

Arguments

fit	LeakFit.

Value

A list containing the fold summary, positive class (if binomial), and a ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  df <- data.frame(
    subject = rep(1:15, each = 2),
    outcome = factor(rep(c(0, 1), 15)),
    x1 = rnorm(30),
    x2 = rnorm(30)
  )
  splits <- make_split_plan(df, outcome = "outcome",
                            mode = "subject_grouped", group = "subject",
                            v = 3, progress = FALSE)
  custom <- list(
    glm = list(
      fit = function(x, y, task, weights, ...) {
        stats::glm(y ~ ., data = as.data.frame(x),
                   family = stats::binomial(), weights = weights)
      },
      predict = function(object, newdata, task, ...) {
        as.numeric(stats::predict(object, newdata = as.data.frame(newdata),
                                  type = "response"))
      }
    )
  )
  fit <- fit_resample(df, outcome = "outcome", splits = splits,
                      learner = "glm", custom_learners = custom,
                      metrics = "auc", refit = FALSE, seed = 1)
  plot_fold_balance(fit)
}
```

`plot_overlap_checks` *Plot overlap diagnostics between train/test groups*

Description

Checks whether the same group identifiers appear in both the training and test partitions within each resample. This is designed to detect leakage from grouped or repeated-measures data (for example, the same subject, batch, plate, or study appearing on both sides of a fold) when group-wise splitting is expected.

Usage

```
plot_overlap_checks(fit, column = NULL)
```

Arguments

`fit`	A 'LeakFit' object produced by [fit_resample()]. It must contain the split indices and the associated metadata in 'fit@splits@info$coldata'. The metadata rows must align with the data used to create the splits.

`column` Character scalar naming the metadata column to check (for example ‘"subject"‘ or ‘"batch"‘). The function compares unique values of this column between train and test within each resample. There is no default: ‘NULL‘ or an unknown column triggers an error. Changing ‘column‘ changes which kind of leakage (subject-level, batch-level, etc.) is tested and therefore the overlap counts.

Details

For each resample in ‘fit@splits@indices‘, the function counts the number of unique values of ‘column‘ in the train and test sets and the size of their intersection. Any non-zero overlap indicates that at least one group appears in both train and test for that resample. The check is metadata-based only: it relies on exact matches of the supplied column and does not inspect features or outcomes. It only checks train vs test within each resample, so it will not detect overlaps across different resamples or other leakage mechanisms. Inconsistent IDs or missing values in the metadata can hide or inflate overlaps. ‘NA‘ values are treated as regular identifiers and will count toward overlap if they appear in both partitions. Requires ggplot2.

Value

A list returned invisibly with:

- ‘overlap_counts‘: data.frame with one row per resample and columns ‘fold‘ (resample index in ‘fit@splits@indices‘), ‘overlap‘ (unique IDs shared by train and test), ‘train‘ (unique IDs in train), and ‘test‘ (unique IDs in test).
- ‘column‘: the metadata column name used for the check.
- ‘plot‘: the ggplot object showing the three count series across folds.

The plot is also printed. When any overlap is detected, the plot adds a warning annotation.

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = rbinom(12, 1, 0.5),
  x1 = rnorm(12),
  x2 = rnorm(12)
)
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject", v = 3)
custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = as.data.frame(x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object, newdata = as.data.frame(newdata),
                                type = "response"))
    }
  )
```

```
)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "accuracy", refit = FALSE)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  out <- plot_overlap_checks(fit, column = "subject")
  out$overlap_counts
}
```

`plot_perm_distribution` *Plot permutation distribution for a LeakAudit object*

Description

Visualizes the label-permutation metric distribution and marks the observed and permuted-mean values to help assess leakage signals. Requires ggplot2.

Usage

```
plot_perm_distribution(audit)
```

Arguments

`audit`	LeakAudit.

Value

A list containing the observed value, permuted mean, permutation values, and a ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  df <- data.frame(
    subject = rep(1:15, each = 2),
    outcome = factor(rep(c(0, 1), 15)),
    x1 = rnorm(30),
    x2 = rnorm(30)
  )
  splits <- make_split_plan(df, outcome = "outcome",
                            mode = "subject_grouped", group = "subject",
                            v = 3, progress = FALSE)
  custom <- list(
    glm = list(
      fit = function(x, y, task, weights, ...) {
        stats::glm(y ~ ., data = as.data.frame(x),
                   family = stats::binomial(), weights = weights)
      },
```

```
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object, newdata = as.data.frame(newdata),
                                type = "response"))
    }
  )
)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)
audit <- audit_leakage(fit, metric = "auc", B = 20)
plot_perm_distribution(audit)
}
```

`plot_time_acf` *Plot ACF of test predictions for time-series leakage checks*

Description

Uses the autocorrelation function of out-of-fold predictions to detect temporal dependence that may indicate leakage. Predictions are ordered by the split time column before computing the ACF. Requires numeric predictions (regression or survival). Requires ggplot2.

Usage

```
plot_time_acf(fit, lag.max = 20)
```

Arguments

`fit`	LeakFit.
`lag.max`	maximum lag to show.

Value

A list with the autocorrelation results, `lag.max`, and a ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  df <- data.frame(
    id = 1:30,
    time = seq.Date(as.Date("2020-01-01"), by = "day", length.out = 30),
    y = rnorm(30),
    x1 = rnorm(30),
    x2 = rnorm(30)
  )
  splits <- make_split_plan(df, outcome = "y", mode = "time_series",
                            time = "time", v = 3, progress = FALSE)
```

```
  custom <- list(
    lm = list(
      fit = function(x, y, task, weights, ...) {
        stats::lm(y ~ ., data = data.frame(y = y, x))
      },
      predict = function(object, newdata, task, ...) {
        as.numeric(stats::predict(object, newdata = as.data.frame(newdata)))
      }
    )
  )
  fit <- fit_resample(df, outcome = "y", splits = splits,
                      learner = "lm", custom_learners = custom,
                      metrics = "rmse", refit = FALSE, seed = 1)
  plot_time_acf(fit, lag.max = 10)
}
```

`predict.GuardFit` *Apply a fitted GuardFit transformer to new data*

Description

Applies the preprocessing steps stored in a `GuardFit` object to new data without refitting any statistics. This is designed to prevent validation leakage that would occur if imputation, scaling, filtering, or feature selection were recomputed on evaluation data. It enforces the training schema by aligning columns and factor levels, and it errors when a numeric-only training fit receives non-numeric predictors. It does not detect label leakage, duplicate samples, or train/test contamination.

`predict.GuardFit()` is the canonical S3 method — callers can use `predict(fit, newdata)` on a `GuardFit` object and the right method is dispatched. `predict_guard()` is retained as a backward-compatible thin alias that simply forwards to the S3 method, so existing code that calls `predict_guard(fit, x)` continues to work.

Usage

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

predict_guard(fit, newdata)
```

Arguments

`object`, `fit`	A `GuardFit` object created by [guard_fit()]. Contains the training-time preprocessing settings and statistics. Changing the object (for example, a different imputation method or feature selection step) changes the output columns and values. `object` is the canonical name (matching the S3 `predict()` generic); `fit` is the legacy name accepted only by `predict_guard()`.

newdata	A matrix or data.frame of predictors with one row per sample. This required argument (no default) is transformed using the training-time parameters in the fit only. Missing columns are added and filled, extra columns are dropped, and factor levels are aligned to the training levels; if the training fit was numeric-only, non-numeric columns in newdata trigger an error.
...	Ignored. Present so that the S3 method signature matches the [stats::predict()] generic; additional arguments are silently dropped.

Value

A data.frame of transformed predictors with the same number of rows as newdata. Column order and content match the training pipeline and may include derived features (one-hot encodings, missingness indicators, or PCA components). This output is not a prediction; it is intended as input to a downstream model and assumes the training-time preprocessing is valid for the new data.

Examples

```
x_train <- data.frame(a = c(1, 2, NA, 4), b = c(10, 11, 12, 13))
fit <- guard_fit(
  x_train,
  y = c(0.1, 0.2, 0.3, 0.4),
  steps = list(impute = list(method = "median")),
  task = "gaussian"
)
x_new <- data.frame(a = c(NA, 5), b = c(9, 14))
## Canonical: dispatch through the predict() generic.
out <- predict(fit, x_new)
out
## Equivalent legacy form (kept for backward compatibility).
identical(out, predict_guard(fit, x_new))
```

---

print.LeakTune *Print a LeakTune object*

Description

Brief one-screen auto-print representation of a ‘LeakTune‘ result returned by [tune_resample()]. Use [summary()] for the full diagnostic report (outer-loop metrics, selected hyperparameters, fold-by-fold detail, and refit summary).

Usage

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

Arguments

x	A ‘LeakTune‘ object returned by [tune_resample()].
...	Ignored; present so that the S3 signature matches [base::print()].

Value

Invisibly returns ‘x‘.

`show,LeakAudit-method` *Display summary for LeakAudit objects*

Description

Prints a brief one-screen summary of a LeakAudit, including task and outcome, the permutation-gap statistic, and counts of batch-association rows, target-leakage features, and duplicate pairs. Use `summary()` for the full diagnostic report.

Usage

```
## S4 method for signature 'LeakAudit'
show(object)
```

Arguments

`object`	A `LeakAudit` object.

Value

No return value, called for side effects (prints a brief summary to the console). Returns `object` invisibly.

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = rbinom(12, 1, 0.5),
  x1 = rnorm(12),
  x2 = rnorm(12)
)
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject", v = 3,
                      progress = FALSE)
custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = as.data.frame(x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object,
                                newdata = as.data.frame(newdata),
                                type = "response"))
    }
```

```
  )
)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)
aud <- audit_leakage(fit, metric = "auc", B = 10,
                     X_ref = df[, c("x1", "x2")])
show(aud)
```

---

show,LeakFit-method *Display summary for LeakFit objects*

Description

Prints a brief one-screen summary of a LeakFit, including task and outcome, fold count and status (successful, skipped, failed), and the headline cross-validated metric. Use `summary()` for the full per-fold diagnostic report.

Usage

```
## S4 method for signature 'LeakFit'
show(object)
```

Arguments

`object`	A `LeakFit` object.

Value

No return value, called for side effects (prints a brief summary to the console). Returns `object` invisibly.

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = rbinom(12, 1, 0.5),
  x1 = rnorm(12),
  x2 = rnorm(12)
)
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject", v = 3,
                      progress = FALSE)
custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = as.data.frame(x),
                 family = stats::binomial(), weights = weights)
```

```
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object,
                                newdata = as.data.frame(newdata),
                                type = "response"))
    }
  )
)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)
show(fit)
```

`show,LeakSplits-method` *Display summary for LeakSplits objects*

Description

Prints fold counts, sizes, and hash metadata for quick inspection.

Usage

```
## S4 method for signature 'LeakSplits'
show(object)
```

Arguments

`object`	LeakSplits object.

Value

No return value, called for side effects (prints a summary to the console showing mode, fold count, repeats, outcome, stratification status, nested status, per-fold train/test sizes, and the reproducibility hash).

Examples

```
df <- data.frame(
  subject = rep(1:10, each = 2),
  outcome = rbinom(20, 1, 0.5),
  x1 = rnorm(20),
  x2 = rnorm(20)
)
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject", v = 5)
show(splits)
```

simulate_leakage_suite

Simulate leakage scenarios and audit results

Description

Simulates synthetic binary classification datasets with optional leakage mechanisms, fits a model using a leakage-aware cross-validation scheme, and summarizes the permutation-gap audit for each Monte Carlo seed. The suite is designed to surface validation failures such as subject overlap across folds, batch-confounded outcomes, global normalization/summary leakage, and time-series look-ahead. The output is a per-seed summary of observed CV performance and its gap versus a label-permutation null; it does not return fitted models or the full audit object. Results are limited to the built-in data generator and leakage types implemented here, and should be interpreted as a simulation-based sanity check rather than a comprehensive leakage detector for real data.

Usage

```
simulate_leakage_suite(
  n = 500,
  p = 20,
  prevalence = 0.5,
  mode = c("subject_grouped", "batch_blocked", "study_loocv", "time_series"),
  learner = c("glmnet", "ranger"),
 leakage = c("none", "subject_overlap", "batch_confounded", "peek_norm", "lookahead"),
  preprocess = NULL,
  rho = 0,
  K = 5,
  repeats = 1,
  horizon = 0,
  B = 200,
  seeds = 1:10,
  parallel = FALSE,
  signal_strength = 1,
  verbose = FALSE
)
```

Arguments

n	Integer scalar. Number of samples to simulate (default 500). Larger values stabilize the Monte Carlo summary but increase runtime.
p	Integer scalar. Number of baseline predictors before any leakage feature is added (default 20). Increasing p changes the signal-to-noise ratio and increases fitting time.
prevalence	Numeric scalar in (0, 1). Target prevalence of class 1 in the simulated outcome (default 0.5). Changing this alters class imbalance and can affect AUC and the permutation gap.

`mode` Character scalar. Cross-validation scheme passed to `make_split_plan()`; one of `"subject_grouped"`, `"batch_blocked"`, `"study_loocv"`, `"time_series"`. Defaults to `"subject_grouped"`. This controls how samples are grouped into folds (by subject, batch, study, or time) and therefore which leakage mechanisms are realistically challenged.

`learner` Character scalar. Base learner, `"glmnet"` (default) or `"ranger"`. Requires the corresponding package in `Suggests`. Switching learners changes the fitted model, runtime, and performance.

`leakage` Character scalar. Leakage mechanism to inject; one of `"none"`, `"subject_overlap"`, `"batch_confounded"`, `"peek_norm"`, `"lookahead"`. Leakage is added as an extra predictor: `"subject_overlap"` adds per-subject mean outcome, `"batch_confounded"` adds per-batch mean outcome, `"peek_norm"` adds the globally normalized (z-scored) outcome, and `"lookahead"` adds the next-time outcome. Changing this controls whether and how leakage is present.

`preprocess` Optional preprocessing list or recipe passed to [fit_resample()]. When NULL (default), the simulator uses the fit_resample defaults; for `"peek_norm"` leakage, normalization is set to `"none"` to avoid attenuating the constant leakage feature.

`rho` Numeric scalar in [-1, 1]. AR(1)-style autocorrelation applied to each predictor across row order (default 0). Higher absolute values increase serial correlation and make time-ordered leakage more pronounced.

`K` Integer scalar. Number of folds/partitions (default 5). Used as the fold count for `"subject_grouped"` and `"batch_blocked"`, and as the number of rolling partitions for `"time_series"`. Ignored for `"study_loocv"` (folds equal the number of studies).

`repeats` Integer scalar >= 1. Number of repeated CV runs for `"subject_grouped"` and `"batch_blocked"` (default 1). Increasing `repeats` increases the number of folds and runtime. Ignored for `"study_loocv"` and `"time_series"`.

`horizon` Numeric scalar >= 0. Minimum time gap enforced between train and test for `"time_series"` splits (default 0). Larger values make the split more conservative and can reduce leakage from temporal proximity.

`B` Integer scalar >= 1. Number of permutations used by `audit_leakage()` to compute the permutation gap and p-value (default 200). Larger values yield more stable p-values but increase runtime.

`seeds` Integer vector. Monte Carlo seeds (default `1:10`). One row of output is produced per seed; changing `seeds` changes the simulated datasets and splits.

`parallel` Logical scalar. If `TRUE`, evaluates seeds in parallel using `future.apply` (if installed). Results are identical to sequential execution; only runtime changes.

`signal_strength` Numeric scalar. Scales the linear predictor before sampling outcomes (default 1). Larger values increase class separation and tend to increase AUC; smaller values make the task harder.

`verbose` Logical scalar. If `TRUE`, prints progress messages for each seed. Does not affect results.

Details

The generator draws p standard normal predictors, builds a linear predictor from the first min(5, p) features, scales it by signal_strength, and samples a binary outcome to achieve the requested prevalence. Outcomes are returned as a two-level factor, so the audited metric is AUC. Simulated metadata include subject, batch, study, and time fields used by mode to create leakage-aware splits. Leakage mechanisms are injected by adding a single extra predictor as described in leakage. Parallel execution uses future.apply when installed and does not change results.

Value

A LeakSimResults data frame with one row per seed and columns:

- seed: seed used for data generation, splitting, and auditing.
- metric_obs: observed CV performance (AUC for this simulation).
- gap: permutation-gap statistic (observed minus permutation mean).
- p_value: permutation p-value for the gap.
- leakage: leakage scenario used.
- mode: CV mode used.

Only the permutation-gap summary is returned; fitted models, predictions, and other audit components are not included.

Note

This function is a general-purpose utility and its data-generation logic intentionally differs from the custom simulation used in the bioLeak manuscript (the manuscript's replication.R script, distributed with the manuscript supplementary materials). Specific differences:

- **peek_norm leakage**: this function uses a z-scored binary outcome as the leak feature; the manuscript uses a noisy continuous version (as.numeric(y) + rnorm(n, 0, 0.3)).
- **lookahead leakage**: this function shifts the binary outcome (c(y[-1], y[n])); the manuscript shifts a continuous biomarker (linpred + noise).
- **signal generation**: this function applies AR correlation to predictors via rho; the manuscript adds AR(1) noise directly to the linear predictor.
- **audit settings**: the manuscript uses perm_refit = FALSE and perm_stratify = TRUE; this function uses perm_refit = "auto" and the perm_stratify default (FALSE).

Users wishing to reproduce manuscript figures should run the manuscript-specific replication.R script directly rather than calling this function.

Examples

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  set.seed(1)
  res <- simulate_leakage_suite(
    n = 120, p = 6, prevalence = 0.4,
    mode = "subject_grouped",
    learner = "glmnet",
```

```
  leakage = "subject_overlap",
  K = 3, repeats = 1,
  B = 50, seeds = 1,
  parallel = FALSE
)
# One row per seed with observed AUC, permutation gap, and p-value
res
}
```

---

summary.LeakAudit *Summarize a leakage audit*

Description

Prints a concise, human-readable report for a ‘LeakAudit‘ object produced by [audit_leakage()]. The summary surfaces four diagnostics when available: label-permutation gap (prediction-label association by default), batch/study association tests (metadata aligned with fold splits), target leakage scan (features strongly associated with the outcome), and near-duplicate detection (high similarity in ‘X_ref‘). The output reflects the stored audit results only; it does not recompute any tests.

Usage

```
## S3 method for class 'LeakAudit'
summary(object, digits = 3, ...)
```

Arguments

object	A ‘LeakAudit‘ object from [audit_leakage()]. The summary reads stored results from ‘object‘ and prints them to the console.
digits	Integer number of digits to show when formatting numeric statistics in the console output. Defaults to ‘3‘. Increasing ‘digits‘ shows more precision; decreasing it shortens the printout without changing the underlying values.
...	Unused. Included for S3 method compatibility; additional arguments are ignored.

Details

The permutation test quantifies prediction-label association when using fixed predictions; refit-based permutations require ‘perm_refit = TRUE‘ (or ‘"auto"‘ with refit data). It does not by itself prove or rule out leakage. Batch association flags metadata that align with fold assignment; this may reflect study design rather than leakage. Target leakage scan uses univariate feature-outcome associations and can miss multivariate proxies, interaction leakage, or features not included in ‘X_ref‘. The multivariate scan (enabled by default for supported tasks) reports an additional model-based score. Duplicate detection only considers the provided ‘X_ref‘ features and the similarity threshold used during [audit_leakage()]. By default, ‘duplicate_scope = "train_test"‘ filters to pairs that cross train/test; set ‘duplicate_scope = "all"‘ to include within-fold duplicates. Sections are reported as "not available" when the corresponding audit component was not computed.

Value

Invisibly returns ‘object‘ after printing the summary.

See Also

[plot_perm_distribution()], [plot_fold_balance()], [plot_overlap_checks()]

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = rbinom(12, 1, 0.5),
  x1 = rnorm(12),
  x2 = rnorm(12)
)
splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject", v = 3)
custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = as.data.frame(x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object, newdata = as.data.frame(newdata),
                                type = "response"))
    }
  )
)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", refit = FALSE, seed = 1)
audit <- audit_leakage(fit, metric = "auc", B = 5,
                       X_ref = df[, c("x1", "x2")], seed = 1)
summary(audit) # prints the audit report and returns `audit` invisibly
```

`summary.LeakDeltaLSI` *Summarize a LeakDeltaLSI object*

Description

Prints a human-readable summary of the Delta LSI analysis comparing leaky vs guarded evaluation pipelines.

Usage

```
## S3 method for class 'LeakDeltaLSI'
summary(object, digits = 3L, ...)
```

Arguments

`object`	A `LeakDeltaLSI` object from `delta_lsi`.
`digits`	Integer. Number of decimal places to show (default 3).
`...`	Unused.

Value

Invisibly returns `object`.

`summary.LeakFit` *Summarize a LeakFit object*

Description

Prints a compact console report for a [LeakFit] object created by [fit_resample()]. The report lists task/outcome metadata, learners, total folds, and cross-validated metrics summarized as mean and standard deviation across completed folds, plus a small audit table with per-fold train/test sizes and retained feature counts.

Usage

```
## S3 method for class 'LeakFit'
summary(object, digits = 3, ...)
```

Arguments

`object`	A [LeakFit] object returned by [fit_resample()]. It should contain ‘metric_summary‘ and ‘audit‘ slots; missing entries result in empty sections in the printed report.
`digits`	Integer scalar. Number of decimal places to print in numeric summary tables. Defaults to 3; affects printed output only, not the returned data.
`...`	Unused. Included for S3 method compatibility; changing these values has no effect.

Details

This summary is meant for quick sanity checks of the resampling setup and performance. It does not run leakage diagnostics and will not detect target leakage, duplicate samples, or batch/study confounding; use [audit_leakage()] or ‘summary()‘ on a [LeakAudit] object for those checks.

Value

Invisibly returns ‘object@metric_summary‘, a data frame of per-learner metric means and standard deviations computed across folds. This function does not recompute metrics.

Examples

```
set.seed(1)
df <- data.frame(
  subject = rep(1:6, each = 2),
  outcome = factor(rep(c(0, 1), each = 6)),
  x1 = rnorm(12),
  x2 = rnorm(12)
)
splits <- make_split_plan(
  df,
  outcome = "outcome",
  mode = "subject_grouped",
  group = "subject",
  v = 3,
  stratify = TRUE,
  progress = FALSE
)
custom <- list(
  glm = list(
    fit = function(x, y, task, weights, ...) {
      stats::glm(y ~ ., data = data.frame(y = y, x),
                 family = stats::binomial(), weights = weights)
    },
    predict = function(object, newdata, task, ...) {
      as.numeric(stats::predict(object,
                                newdata = as.data.frame(newdata),
                                type = "response"))
    }
  )
)
fit <- fit_resample(df, outcome = "outcome", splits = splits,
                    learner = "glm", custom_learners = custom,
                    metrics = "auc", seed = 1)
summary_df <- summary(fit)
summary_df
```

`summary.LeakTune` *Summarize a nested tuning result*

Description

Prints a concise report for a ‘LeakTune‘ object produced by [tune_resample()]. The report highlights the tuning strategy, selection metric, and cross-validated performance across outer folds, plus a glimpse of the selected hyperparameters.

Usage

```
## S3 method for class 'LeakTune'
summary(object, digits = 3, ...)
```

Arguments

object	A [LeakTune] object returned by [tune_resample()].
digits	Integer scalar. Number of decimal places to print in numeric summary tables. Defaults to 3.
...	Unused. Included for S3 method compatibility.

Value

Invisibly returns ‘object$metric_summary‘, the data frame of per-learner metric means and standard deviations computed across outer folds.

---

tune_resample *Leakage-aware nested tuning with tidymodels*

Description

Runs nested cross-validation for hyperparameter tuning using leakage-aware splits. Inner resamples are constructed from each outer training fold to avoid information leakage during tuning. Requires tidymodels tuning packages and a workflow or recipe-based preprocessing. Survival tasks are not yet supported.

Usage

```
tune_resample(
  x,
  outcome,
  splits,
  learner,
  preprocess = NULL,
  grid = 10,
  metrics = NULL,
  positive_class = NULL,
  selection = c("best", "one_std_err"),
  selection_metric = NULL,
  inner_v = NULL,
  inner_repeats = 1,
  inner_seed = NULL,
  control = NULL,
  parallel = FALSE,
  refit = FALSE,
  seed = 1,
  split_cols = "auto",
  tune_threshold = FALSE,
  threshold_grid = seq(0.1, 0.9, by = 0.05),
  threshold_metric = "accuracy"
)
```

Arguments

x	SummarizedExperiment or matrix/data.frame.
outcome	Outcome column name (if x is SE or data.frame).
splits	LeakSplits object defining the outer resamples. If the splits do not already include inner folds, they are created from each outer training fold using the same split metadata. rsample splits must already include inner folds.
learner	A parsnip model_spec with tunable parameters, or a workflows workflow. When a model_spec is provided, a workflow is built using 'preprocess' or a formula.
preprocess	Optional 'recipes::recipe'. Required when you need preprocessing for tuning. Ignored when 'learner' is already a workflow. Recipe/workflow leakage guardrails run before tuning; configure policy via `options(bioLeak.validation_mode = "warn" | "error" | "off")`.
grid	Tuning grid passed to 'tune::tune_grid()'. Can be a data.frame or an integer size.
metrics	Character vector of metric names ('auc', 'pr_auc', 'accuracy', 'macro_f1', 'log_loss', 'rmse') or a yardstick metric set/list. Metrics are computed with yardstick; unsupported metrics are dropped with a warning. For binomial tasks, if any inner assessment fold contains a single class, probability metrics ('auc', 'roc_auc', 'pr_auc') are dropped for tuning with a warning.
positive_class	Optional value indicating the positive class for binomial outcomes. When set, the outcome levels are reordered so the positive class is second.
selection	Selection rule for tuning, either '"best"' or '"one_std_err"'.
selection_metric	Metric name used for selecting hyperparameters. Defaults to the first metric in 'metrics'. If the chosen metric yields no valid results, the first available metric is used with a warning.
inner_v	Optional number of folds for inner CV when inner splits are not precomputed. Defaults to the outer 'v'.
inner_repeats	Optional number of repeats for inner CV when inner splits are not precomputed. Defaults to 1.
inner_seed	Optional seed for inner split generation when inner splits are not precomputed. Defaults to the outer split seed.
control	Optional 'tune::control_grid()' settings for tuning.
parallel	Logical; passed to [fit_resample()] when evaluating outer folds (single-fold, no refit).
refit	Logical; if TRUE, refits a final tuned workflow on the full dataset using aggregated hyperparameters across all outer folds (median for numeric parameters, majority vote for categorical). This avoids nested-CV leakage that would occur from selecting a single fold's params.
seed	Integer seed for reproducibility.
split_cols	Optional named list/character vector or '"auto"' (default) overriding group/batch/study/time column names when 'splits' is an rsample object and its attributes are missing. '"auto"' falls back to common metadata column names (e.g., 'group', 'subject', 'batch', 'study', 'time'). Supported names are 'group', 'batch', 'study', and 'time'.

`tune_threshold` Logical; when ‘TRUE‘ for binomial tasks, selects a probability threshold from inner-fold predictions and applies it only to the corresponding outer-fold evaluation.

`threshold_grid` Numeric vector of thresholds in ‘[0, 1]‘ considered when ‘tune_threshold = TRUE‘.

`threshold_metric`
Metric used to pick thresholds when ‘tune_threshold = TRUE‘. Supported values are ‘"accuracy"‘, ‘"balanced_accuracy"‘, and ‘"f1"‘, or a custom function with signature ‘function(truth, pred_class, prob, threshold)‘.

Value

A list of class ‘"LeakTune"‘ with components:

`metrics` Outer-fold metrics.

`metric_summary` Mean/SD metrics across outer folds with columns `learner`, and `<metric>_mean` and `<metric>_sd` for each metric.

`best_params` Best hyperparameters per outer fold.

`inner_results` List of inner tuning results.

`outer_fits` List of outer LeakFit objects.

`thresholds` Per-fold threshold choices when threshold tuning is enabled.

`fold_status` Outer-fold status log with stage, status, reason, and notes.

`final_model` Optional final workflow fit when ‘refit = TRUE‘.

`info` Metadata about the tuning run.

Examples

```
if (requireNamespace("tune", quietly = TRUE) &&
    requireNamespace("recipes", quietly = TRUE) &&
    requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("rsample", quietly = TRUE) &&
    requireNamespace("workflows", quietly = TRUE) &&
    requireNamespace("yardstick", quietly = TRUE) &&
   requireNamespace("dials", quietly = TRUE)) {
  df <- data.frame(
    subject = rep(1:10, each = 2),
    outcome = factor(rep(c(0, 1), each = 10)),
    x1 = rnorm(20),
    x2 = rnorm(20)
  )
  splits <- make_split_plan(df, outcome = "outcome",
                      mode = "subject_grouped", group = "subject",
                      v = 3, nested = TRUE, stratify = TRUE)
  spec <- parsnip::logistic_reg(penalty = tune::tune(), mixture = 1) |>
    parsnip::set_engine("glmnet")
  rec <- recipes::recipe(outcome ~ x1 + x2, data = df)
  tuned <- tune_resample(df, outcome = "outcome", splits = splits,
                       learner = spec, preprocess = rec, grid = 5)
```

```
  tuned$metric_summary
}
```

Index

as_leaksplits, 3
as_rsample, *4*, 4
audit_batch_assoc, 5, *12*
audit_batch_assoc,LeakAudit-method *(*audit_batch_assoc*)*, 5
audit_duplicates, 6, *12*
audit_duplicates,LeakAudit-method *(*audit_duplicates*)*, 6
audit_info, 7, *12*
audit_info,LeakAudit-method *(*audit_info*)*, 7
audit_leakage, 7, *14*, *26*
audit_leakage_by_learner, 13
audit_perm_gap, *12*, 15
audit_perm_gap,LeakAudit-method *(*audit_perm_gap*)*, 15
audit_report, 16
audit_target_assoc, *12*, 18
audit_target_assoc,LeakAudit-method *(*audit_target_assoc*)*, 18

benchmark_leakage_suite, 19
browseURL, *17*

calibration_summary, 20
check_split_overlap, 21
confounder_sensitivity, 22
cv_ci, 23

delta_lsi, 24, *50*, *65*
dlsi_ci, *26*, 27
dlsi_ci,LeakDeltaLSI-method *(*dlsi_ci*)*, 27
dlsi_metric, *26*, 27
dlsi_metric,LeakDeltaLSI-method *(*dlsi_metric*)*, 27
dlsi_p_value, *26*, 28
dlsi_p_value,LeakDeltaLSI-method *(*dlsi_p_value*)*, 28
dlsi_R_eff, *26*, 30
dlsi_R_eff,LeakDeltaLSI-method *(*dlsi_R_eff*)*, 30
dlsi_repeats, *26*, 29, *50*
dlsi_repeats,LeakDeltaLSI-method *(*dlsi_repeats*)*, 29
dlsi_robust, *26*, 30
dlsi_robust,LeakDeltaLSI-method *(*dlsi_robust*)*, 30
dlsi_tier, *26*, 31
dlsi_tier,LeakDeltaLSI-method *(*dlsi_tier*)*, 31

fit_metrics, 32, *35*
fit_metrics,LeakFit-method *(*fit_metrics*)*, 32
fit_resample, *26*, 33, *44*

guard_ensure_levels, 36
guard_fit, 37
guard_to_recipe, 38

impute_guarded, 39

LeakAudit, *11*, *14*, *45*
LeakAudit-class *(*LeakSplits-class*)*, 40
LeakClasses *(*LeakSplits-class*)*, 40
LeakDeltaLSI, *26*, *40*, *46*, *50*
LeakDeltaLSI-class *(*LeakSplits-class*)*, 40
LeakFit, *25*, *35*, *47*
LeakFit-class *(*LeakSplits-class*)*, 40
LeakSplits, *21*, *44*
LeakSplits-class, 40

make_split_plan, *3*, *4*, *21*, 42

plot,LeakAudit,missing-method, 45
plot,LeakAudit-method *(*plot,LeakAudit,missing-method*)*, 45
plot,LeakDeltaLSI,missing-method, 45

plot,LeakDeltaLSI-method
(plot,LeakDeltaLSI,missing-method), 45
plot,LeakFit,missing-method, 46
plot,LeakFit-method
(plot,LeakFit,missing-method), 46
plot_calibration, *46*, *47*, 47
plot_confounder_sensitivity, *46*, *47*, 48
plot_dlsi_repeats, *46*, 50
plot_fold_balance, *46*, *47*, 50
plot_overlap_checks, *46*, *47*, 51
plot_perm_distribution, *45*, 53
plot_time_acf, *46*, *47*, 54
predict.GuardFit, 55
predict_guard *(*predict.GuardFit*)*, 55
print.LeakTune, 56

show,LeakAudit-method, 57
show,LeakDeltaLSI-method
(LeakSplits-class), 40
show,LeakFit-method, 58
show,LeakSplits-method, 59
simulate_leakage_suite, 60
summary.LeakAudit, 63
summary.LeakDeltaLSI, 64
summary.LeakFit, 65
summary.LeakTune, 66

tune_resample, 67