Package 'BORG'

May 6, 2026

Title Bounded Outcome Risk Guard for Model Evaluation

Version 0.3.1

Description Comprehensive toolkit for valid spatial, temporal, and grouped model evaluation. Automatically detects data dependencies (spatial autocorrelation, temporal structure, clustered observations), generates appropriate cross-validation schemes (spatial blocking, checkerboard, hexagonal, KNNDM, environmental blocking, leave-location-out, purged CV), and validates evaluation pipelines for leakage. Includes area of applicability (AOA) assessment following Meyer & Pebesma (2021) <doi:10.1111/2041-210X.13650>, forward feature selection with blocked CV, spatial thinning, block-permutation variable importance, extrapolation detection, and interactive visualizations. Integrates with 'tidymodels', 'caret', 'mlr3', 'ENMeval', and 'biomod2'. Based on evaluation principles described in Roberts et al. (2017) <doi:10.1111/ecog.02881>, Kaufman et al. (2012) <doi:10.1145/2382577.2382579>, Kapoor & Narayanan (2023) <doi:10.1016/j.patter.2023.100804>, and Linnenbrink et al. (2024) <doi:10.5194/gmd-17-5897-2024>.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

RoxygenNote 7.3.3

LinkingTo Rcpp

Imports Rcpp, methods, stats, utils

Suggests caret, ggplot2 (>= 3.4.0), leaflet, rsample, tidyselect, recipes, mlr3, sf, terra, tidyterra, ranger, parsnip, workflows, xgboost, lightgbm, future.apply, testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/gcol33/BORG, https://gillescolling.com/BORG/

BugReports https://github.com/gcol33/BORG/issues

Depends R (>= 4.1.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Gilles Colling [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-3070-6066>)

Maintainer Gilles Colling <gilles.colling051@gmail.com>

Repository CRAN

Date/Publication 2026-03-29 15:50:40 UTC

Contents

`as.data.frame.BorgDiagnosis`

Coerce BorgDiagnosis to Data Frame

Description

Converts a `BorgDiagnosis` object into a one-row data frame of diagnostic results for programmatic access.

Usage

```
## S3 method for class 'BorgDiagnosis'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A `BorgDiagnosis` object.
`row.names`	Optional row names for the output data frame.
`optional`	Logical. Passed to `data.frame()`.
`...`	Additional arguments passed to `data.frame()`.

Value

A one-row data frame with columns: `dependency_type`, `severity`, `recommended_cv`, `n_obs`, `spatial_detected`, `morans_i`, `temporal_detected`, `acf_lag1`, `clustered_detected`, `icc`.

See Also

`BorgDiagnosis`

as.data.frame.BorgRisk *Coerce BorgRisk to Data Frame*

Description

Converts a BorgRisk object into a data frame of detected risks.

Usage

```
## S3 method for class 'BorgRisk'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A BorgRisk object.
row.names	Optional row names for the output data frame.
optional	Logical. Passed to data.frame().
...	Additional arguments passed to data.frame().

Value

A data frame where each row corresponds to a detected risk. Columns are: type, severity, description, source_object, n_affected.

See Also

BorgRisk

audit_importance *Audit Feature Importance Calculations*

Description

Detects when feature importance (SHAP, permutation importance, etc.) is computed using test data, which can lead to biased feature selection and data leakage.

Usage

```
audit_importance(
  importance,
  data,
  train_idx,
  test_idx,
  method = "auto",
  model = NULL
)
```

Arguments

`importance`	A vector, matrix, or data frame of importance values.
`data`	The data used to compute importance.
`train_idx`	Integer vector of training indices.
`test_idx`	Integer vector of test indices.
`method`	Character indicating the importance method. One of "shap", "permutation", "gain", "impurity", or "auto" (default).
`model`	Optional fitted model object for additional validation.

Details

Feature importance computed on test data is a form of data leakage because:

- SHAP values computed on test data reveal test set structure
- Permutation importance on test data uses test labels
- Feature selection based on test importance leads to overfit models

This function checks if the data used for importance calculation includes test indices and flags potential violations.

Value

A BorgRisk object with audit results.

Examples

```
set.seed(42)
data <- data.frame(y = rnorm(100), x1 = rnorm(100), x2 = rnorm(100))
train_idx <- 1:70
test_idx <- 71:100

# Simulate importance values
importance <- c(x1 = 0.6, x2 = 0.4)

# Good: importance computed on training data
result <- audit_importance(importance, data[train_idx, ], train_idx, test_idx)

# Bad: importance computed on full data (includes test)
result_bad <- audit_importance(importance, data, train_idx, test_idx)
```

`audit_predictions` *Audit Predictions for Data Leakage*

Description

Validates that predictions were generated correctly without data leakage. Checks that predictions correspond to test data only and that the prediction process did not use information from the test set.

Usage

```
audit_predictions(
  predictions,
  train_idx,
  test_idx,
  actual = NULL,
  data = NULL,
  model = NULL
)
```

Arguments

`predictions`	Vector of predictions (numeric or factor).
`train_idx`	Integer vector of training indices.
`test_idx`	Integer vector of test indices.
`actual`	Optional vector of actual values for comparison.
`data`	Optional data frame containing the original data.
`model`	Optional fitted model object for additional checks.

Value

A BorgRisk object with audit results.

Examples

```
# Create data and split
set.seed(42)
data <- data.frame(y = rnorm(100), x = rnorm(100))
train_idx <- 1:70
test_idx <- 71:100

# Fit model and predict
model <- lm(y ~ x, data = data[train_idx, ])
predictions <- predict(model, newdata = data[test_idx, ])

# Audit predictions
result <- audit_predictions(predictions, train_idx, test_idx)
```

`autoplot.BorgDiagnosis` *Autoplot Method for BorgDiagnosis Objects*

Description

Creates a summary panel of detected dependency diagnostics.

Usage

```
## S3 method for class 'BorgDiagnosis'
autoplot(object, type = c("summary", "variogram"), ...)
```

Arguments

`object`	A `BorgDiagnosis` object.
`type`	Character. Plot type: `"summary"` (default) for dependency gauge panel, or `"variogram"` for empirical semivariogram showing spatial autocorrelation structure and suggested block size.
`...`	Additional arguments (currently unused).

Details

Requires the **ggplot2** package.

Value

A `ggplot` object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  d <- data.frame(
    site = rep(1:20, each = 10),
    value = rep(rnorm(20, sd = 2), each = 10) + rnorm(200, sd = 0.5)
  )
  diag <- borg_diagnose(d, groups = "site", target = "value")
  ggplot2::autoplot(diag)
}
```

`autoplot.BorgRisk` *Autoplot Method for BorgRisk Objects*

Description

Creates a ggplot2 visualization of detected risks from a BORG validation.

Usage

```
## S3 method for class 'BorgRisk'
autoplot(object, max_risks = 10, show_fixes = TRUE, ...)
```

Arguments

`object`	A `BorgRisk` object from `borg_inspect()` or `borg()` in validation mode.
`max_risks`	Integer. Maximum number of risks to display. Default: 10.
`show_fixes`	Logical. If TRUE (default), annotate each risk with a suggested fix from `borg_assimilate()` or manual action. If FALSE, show the risk description instead.
`...`	Additional arguments (currently unused).

Details

Requires the **ggplot2** package.

Value

A `ggplot` object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  data <- data.frame(x = 1:100, y = 101:200)
  result <- borg_inspect(data, train_idx = 1:60, test_idx = 51:100)
  ggplot2::autoplot(result)
}
```

`autoplot.borg_comparison` *Autoplot Method for borg_comparison Objects*

Description

Visualizes random vs blocked CV comparison results.

Usage

```
## S3 method for class 'borg_comparison'
autoplot(object, type = c("boxplot", "density", "paired"), ...)
```

Arguments

`object`	A `borg_comparison` object from `borg_compare_cv()`.
`type`	Character. Plot type: `"boxplot"` (default), `"density"`, or `"paired"`.
`...`	Additional arguments (currently unused).

Details

Requires the **ggplot2** package.

Value

A `ggplot` object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  d <- data.frame(
    site = rep(1:20, each = 10),
    x = rnorm(200),
    y = rep(rnorm(20, sd = 2), each = 10) + rnorm(200, sd = 0.5)
  )
  comp <- borg_compare_cv(d, formula = y ~ x, groups = "site", repeats = 3)
  ggplot2::autoplot(comp)
}
```

`autoplot.borg_cv` *Autoplot Method for borg_cv Objects*

Description

Visualizes cross-validation fold structure.

Usage

```
## S3 method for class 'borg_cv'
autoplot(
  object,
  type = c("folds", "spatial", "sizes"),
  data = NULL,
  coords = NULL,
  raster = NULL,
  ...
)
```

Arguments

`object`	A `borg_cv` object from `borg_cv()`.
`type`	Character. Plot type: `"folds"` (default), `"spatial"`, or `"sizes"`.
`data`	Optional data frame (or sf/SpatVector) for spatial plots. Required for `type = "spatial"`.
`coords`	Character vector of coordinate column names. Required for `type = "spatial"` when `data` is a plain data.frame.
`raster`	Optional `terra::SpatRaster`. When provided with `type = "spatial"`, the first layer is rendered as a background map behind the fold points. Requires the **tidyterra** package.
`...`	Additional arguments (currently unused).

Details

Requires the **ggplot2** package.

Value

A `ggplot` object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
  cv <- borg_cv(d, coords = c("x", "y"), target = "z")
```

```
  ggplot2::autoplot(cv, type = "sizes")
  ggplot2::autoplot(cv, type = "spatial", data = d, coords = c("x", "y"))
}
```

`autoplot.borg_fold_perf` *Autoplot Method for borg_fold_perf Objects*

Description

Autoplot Method for borg_fold_perf Objects

Usage

```
## S3 method for class 'borg_fold_perf'
autoplot(object, ...)
```

Arguments

`object`	A `borg_fold_perf` object.
`...`	Additional arguments (currently unused).

Value

A `ggplot` object. If spatial centroids are available, shows a map of per-fold performance. Otherwise, a bar chart.

`autoplot.borg_result` *Autoplot Method for borg_result Objects*

Description

Creates ggplot2 visualizations of BORG diagnosis + CV results.

Usage

```
## S3 method for class 'borg_result'
autoplot(
  object,
  type = c("split", "spatial", "temporal", "groups"),
  fold = 1,
  data = NULL,
  coords = NULL,
  raster = NULL,
```

```
  time = NULL,
  groups = NULL,
  ...
)
```

Arguments

object	A borg_result object from borg().
type	Character. Plot type: "split" (default), "spatial", "temporal", or "groups".
fold	Integer. Which fold to plot. Default: 1.
data	Optional data frame (or sf/SpatVector) for spatial plots. Required for type = "spatial" to obtain coordinates.
coords	Character vector of coordinate column names. Required for type = "spatial" when data is a plain data.frame.
raster	Optional terra::SpatRaster. When provided with type = "spatial", the first layer is rendered as a background map behind the fold points. Requires the **tidyterra** package.
time	Character name of the time column in data, or a vector of time values (length = nrow(data)). Required for type = "temporal".
groups	Character name of the grouping column in data, or a vector of group labels. Required for type = "groups".
...	Additional arguments (currently unused).

Details

Requires the **ggplot2** package. For spatial plots, the **sf** package is recommended for proper map projections.

Value

A ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(42)
  d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
  result <- borg(d, coords = c("x", "y"), target = "z")
  ggplot2::autoplot(result)
  ggplot2::autoplot(result, type = "spatial", data = d, coords = c("x", "y"))
}
```

borg *BORG: Guard Your Model Evaluation*

Description

The main entry point for BORG. Diagnoses data dependencies, generates valid cross-validation schemes, and validates evaluation workflows.

Usage

```
borg(
  data,
  coords = NULL,
  time = NULL,
  groups = NULL,
  target = NULL,
  formula = NULL,
  v = 5,
  train_idx = NULL,
  test_idx = NULL,
  buffer = NULL,
  env = NULL,
  repeats = 1L,
  output = c("list", "rsample", "caret", "mlr3"),
  ...
)
```

Arguments

data	A data frame, `sf` object, or `terra::SpatVector` to diagnose and create CV folds for. For `sf` and `SpatVector` inputs, coordinates are extracted automatically from the geometry and the `coords` argument is not needed.
coords	Character vector of length 2 specifying coordinate column names (e.g., `c("lon", "lat")`). Triggers spatial autocorrelation detection.
time	Character string specifying the time column name. Triggers temporal autocorrelation detection.
groups	Character string specifying the grouping column name (e.g., "site_id", "patient_id"). Triggers clustered structure detection.
target	Character string specifying the response variable column name. Used for more accurate autocorrelation diagnostics.
formula	A model formula (e.g. `y ~ x1 + x2`). If supplied, the response variable is extracted as `target` (unless `target` is also provided).
v	Integer. Number of CV folds. Default: 5.
train_idx	Integer vector of training indices. If provided along with `test_idx`, validates an existing split instead of generating one.

test_idx	Integer vector of test indices. Required if train_idx is provided.
buffer	Numeric. Spatial buffer distance (in coordinate units) applied around test-fold observations. Training points within this buffer are removed to reduce spatial autocorrelation leakage. Default: NULL (no buffer).
env	Environmental covariates used for environmental blocking. A SpatRaster, data frame, or matrix with one row per observation. Passed to borg_cv.
repeats	Integer. Number of times to repeat the CV fold generation with different random seeds. Default: 1 (no repetition).
output	Character. CV output format: "list" (default), "rsample", "caret", "mlr3". Ignored when validating an existing split.
...	Additional arguments passed to underlying functions.

Details

borg() operates in two modes:

Diagnosis Mode (Recommended): When called with structure hints (coords, time, groups) but without train_idx/test_idx, BORG:

1. Diagnoses data dependencies (spatial, temporal, clustered)
2. Estimates how much random CV would inflate metrics
3. Generates appropriate CV folds that respect the dependency structure
4. Returns everything needed to proceed with valid evaluation

This is the recommended workflow. Let BORG tell you how to split your data.

Validation Mode: When called with train_idx and test_idx, BORG validates the existing split:

- Checks for index overlap
- Validates group isolation (if groups specified)
- Validates temporal ordering (if time specified)
- Checks spatial separation (if coords specified)
- Detects preprocessing leakage, target leakage, etc.

Use this mode to verify splits you've created yourself.

Value

Depends on usage mode:

Diagnosis mode (no train_idx/test_idx): A list with class "borg_result" containing:

diagnosis A BorgDiagnosis object with dependency analysis

cv A borg_cv object with valid cross-validation folds

folds Shortcut to cv$folds for convenience

Validation mode (with train_idx/test_idx): A BorgRisk object containing the risk assessment of the provided split.

See Also

borg_diagnose for diagnosis only, borg_cv for CV generation only, borg_inspect for detailed object inspection.

Examples

```
# ===== DIAGNOSIS MODE (recommended) =====

# Spatial data: let BORG create valid folds
set.seed(42)
spatial_data <- data.frame(
  x = runif(200, 0, 100),
  y = runif(200, 0, 100),
  response = rnorm(200)
)

result <- borg(spatial_data, coords = c("x", "y"), target = "response")
result$diagnosis
result$folds[[1]]  # First fold's train/test indices

# Clustered data
clustered_data <- data.frame(
  site = rep(1:20, each = 10),
  value = rep(rnorm(20), each = 10) + rnorm(200, sd = 0.5)
)

result <- borg(clustered_data, groups = "site", target = "value")
result$diagnosis@recommended_cv  # "group_fold"

# Temporal data
temporal_data <- data.frame(
  date = seq(as.Date("2020-01-01"), by = "day", length.out = 200),
  value = cumsum(rnorm(200))
)

result <- borg(temporal_data, time = "date", target = "value")


# Get rsample-compatible output for tidymodels (requires rsample package)
result <- borg(spatial_data, coords = c("x", "y"), output = "rsample")


# ===== VALIDATION MODE =====

# Validate an existing split
data <- data.frame(x = 1:100, y = rnorm(100))
borg(data, train_idx = 1:70, test_idx = 71:100)

# Validate with group constraint
data$patient <- rep(1:10, each = 10)
borg(data, train_idx = 1:50, test_idx = 51:100, groups = "patient")
```

borg-wrappers *BORG-Guarded Cross-Validation Functions*

Description

These functions wrap common cross-validation functions from popular ML frameworks, adding automatic BORG validation. They block random CV when data dependencies are detected.

Details

BORG provides guarded versions of:

- `borg_vfold_cv()`: Wraps `rsample::vfold_cv()`
- `borg_group_vfold_cv()`: Wraps `rsample::group_vfold_cv()`
- `borg_initial_split()`: Wraps `rsample::initial_split()`

When dependencies are detected, these functions either:

1. Block the operation and suggest `borg_cv()` instead
2. Automatically switch to an appropriate blocked CV strategy

Value

No return value. This page documents the family of guarded CV wrapper functions; see individual functions for their return values.

BorgDiagnosis *BorgDiagnosis S4 Class*

Description

Holds the result of `borg_diagnose`: a structured assessment of data dependency patterns that affect cross-validation validity.

Usage

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

Arguments

`object`	A `BorgDiagnosis` object to be printed.

Value

The `BorgDiagnosis` object, returned invisibly. Called for the side effect of printing a diagnostic summary to the console.

Slots

`dependency_type` Character. Primary dependency type detected: "none", "spatial", "temporal", "clustered", or "mixed".

`severity` Character. Overall severity: "none", "moderate", "severe".

`recommended_cv` Character. Recommended CV strategy: "random", "spatial_block", "temporal_block", "group_fold", "spatial_temporal".

`spatial` List. Spatial autocorrelation diagnostics with elements: detected (logical), morans_i (numeric), morans_p (numeric), range_estimate (numeric), effective_n (numeric), coords_used (character).

`temporal` List. Temporal autocorrelation diagnostics with elements: detected (logical), acf_lag1 (numeric), ljung_box_p (numeric), decorrelation_lag (integer), embargo_minimum (integer), time_col (character).

`clustered` List. Clustered structure diagnostics with elements: detected (logical), icc (numeric), n_clusters (integer), cluster_sizes (numeric), design_effect (numeric), group_col (character).

`inflation_estimate` List. Estimated metric inflation from random CV with elements: auc_inflation (numeric, proportion), rmse_deflation (numeric), confidence (character: "low"/"medium"/"high"), basis (character).

`n_obs` Integer. Number of observations in the dataset.

`timestamp` POSIXct. When the diagnosis was performed.

`call` Language object. The original call that triggered diagnosis.

See Also

`borg_diagnose`, `borg_cv`

BorgRisk *BorgRisk S4 Class*

Description

Holds the result of `borg_inspect` or `borg_validate`: a structured assessment of evaluation risks detected in a workflow or object.

This class stores identified risks, their classification (hard violation vs soft inflation), affected data indices, and recommended remediation actions.

Usage

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

Arguments

`object`	A `BorgRisk` object to be printed.

Value

The BorgRisk object, returned invisibly. Called for the side effect of printing a risk assessment summary to the console.

Slots

`risks` A list of detected risk objects, each containing:

- **type** Character string: risk category (e.g., "preprocessing_leak")
- **severity** Character string: "hard_violation" or "soft_inflation"
- **description** Character string: human-readable description
- **affected_indices** Integer vector: row/column indices affected
- **source_object** Character string: name of the leaky object

`n_hard` Integer. Count of hard violations detected.

`n_soft` Integer. Count of soft inflation risks detected.

`is_valid` Logical. TRUE if no hard violations detected.

`train_indices` Integer vector. Row indices in training set.

`test_indices` Integer vector. Row indices in test set.

`timestamp` POSIXct. When the inspection was performed.

`call` Language object. The original call that triggered inspection.

See Also

borg_inspect, borg_validate, borg

Examples

```
# Create an empty BorgRisk object (no risks detected)
show(new("BorgRisk",
  risks = list(),
  n_hard = 0L,
  n_soft = 0L,
  is_valid = TRUE,
  train_indices = 1:80,
  test_indices = 81:100,
  timestamp = Sys.time(),
  call = quote(borg_inspect(x))
))
```

`borg_adversarial` *Adversarial Validation*

Description

Trains a binary classifier to distinguish training data from prediction locations. High classification accuracy (AUC > 0.7) indicates the prediction domain differs substantially from training, and spatial/blocked CV is essential. Low accuracy (AUC ~ 0.5) means random CV may suffice.

Usage

```
borg_adversarial(train, prediction, predictors = NULL, v = 5L)
```

Arguments

`train`	Data frame of training data.
`prediction`	Data frame of prediction locations (same predictor columns).
`predictors`	Character vector. If `NULL`, all shared numeric columns.
`v`	Integer. Number of CV folds for adversarial classifier. Default: 5.

Details

Uses logistic regression as the adversarial classifier (no external dependencies). The AUC is computed via cross-validation on the combined train+prediction dataset with a binary label (0=train, 1=prediction).

Interpretation:

- AUC < 0.6: Low dissimilarity. Random CV likely adequate.
- AUC 0.6-0.8: Moderate dissimilarity. Spatial CV recommended.
- AUC > 0.8: High dissimilarity. Spatial CV essential; check AOA.

Value

A list with class `"borg_adversarial"` containing:

auc Cross-validated AUC of the adversarial classifier

dissimilarity Dissimilarity score (0 to 100 percent)

recommendation Suggested CV strategy based on dissimilarity

importance Variable importance for distinguishing domains

Examples

```
set.seed(42)
train <- data.frame(a = rnorm(100), b = rnorm(100))
pred <- data.frame(a = rnorm(50, mean = 2), b = rnorm(50))
av <- borg_adversarial(train, pred)
av
```

borg_aoa *Area of Applicability*

Description

Determines where a spatial prediction model can be trusted, following Meyer & Pebesma (2021). Combines the dissimilarity index (DI) with a threshold derived from cross-validated DI values to produce a binary applicability mask.

Usage

```
borg_aoa(
  train,
  new,
  predictors = NULL,
  coords = NULL,
  weights = NULL,
  folds = NULL,
  threshold = NULL
)
```

Arguments

train	Data frame of training data.
new	Data frame of prediction locations (same predictor columns).
predictors	Character vector. Predictor column names.
coords	Character vector of length 2. Coordinate columns in new for spatial mapping. Optional.
weights	Numeric vector. Variable importance weights.
folds	Optional borg_cv object or fold list. If provided, the threshold is derived from cross-validated DI (more robust).
threshold	Numeric. Manual DI threshold override. If NULL, computed from training data (mean + sd of LOO-DI).

Value

A data frame with class "borg_aoa" containing:

di Dissimilarity index for each prediction point

aoa Logical. TRUE if inside AOA (DI <= threshold)

x, y Coordinates (if coords provided)

Has autoplot() method showing the AOA map.

References

Meyer, H., & Pebesma, E. (2021). Predicting into unknown space? Estimating the area of applicability of spatial prediction models. *Methods in Ecology and Evolution*, 12(9), 1620-1633. doi:10.1111/2041210X.13650

Examples

```
set.seed(42)
train <- data.frame(x = runif(80, 0, 50), y = runif(80, 0, 50),
                    a = rnorm(80), b = rnorm(80))
pred <- data.frame(x = runif(200, 0, 100), y = runif(200, 0, 100),
                   a = rnorm(200), b = rnorm(200))
aoa <- borg_aoa(train, pred, predictors = c("a", "b"), coords = c("x", "y"))
table(aoa$aoa)
```

`borg_assimilate` *Assimilate Leaky Evaluation Pipelines*

Description

`borg_assimilate()` attempts to automatically fix detected evaluation risks by restructuring the pipeline to eliminate information leakage.

Usage

```
borg_assimilate(workflow, risks = NULL, fix = "all")
```

Arguments

`workflow`	A list containing the evaluation workflow (same structure as `borg_validate`).
`risks`	Optional `BorgRisk` object from a previous inspection. If NULL, `borg_validate()` is called first.
`fix`	Character vector specifying which risk types to attempt to fix. Default: `"all"` attempts all rewritable violations. Other options: `"preprocessing"`, `"feature_engineering"`, `"thresholds"`.

Details

`borg_assimilate()` can automatically fix certain types of leakage:

Preprocessing on full data Refits preprocessing objects using only training indices

Feature engineering leaks Recomputes target encodings, embeddings, and derived features using train-only data

Threshold optimization Moves threshold selection to training/validation data

Some violations cannot be automatically fixed:

- Train-test index overlap (requires new split)
- Target leakage in original features (requires domain intervention)
- Temporal look-ahead in features (requires feature re-engineering)

Value

A list containing:

workflow The rewritten workflow (modified in place where possible)

fixed Character vector of risk types that were successfully fixed

unfixable Character vector of risk types that could not be fixed

report `BorgRisk` object from post-rewrite validation

See Also

`borg_validate` for validation without assimilation, `borg` for proactive enforcement.

Examples

```
# Attempt to fix a leaky workflow
workflow <- list(
  data = data.frame(x = rnorm(100), y = rnorm(100)),
  train_idx = 1:70,
  test_idx = 71:100
)
result <- borg_assimilate(workflow)

if (length(result$unfixable) > 0) {
  message("Some risks require manual intervention:")
  print(result$unfixable)
}
```

---

borg_auto_check *Enable/Disable BORG Auto-Check Mode*

Description

Configures BORG to automatically validate train/test splits when using supported ML frameworks. When enabled, BORG will intercept common modeling functions and validate indices before training proceeds.

Usage

```
borg_auto_check(enable = TRUE, strict = TRUE, verbose = FALSE)
```

Arguments

enable	Logical. If TRUE, enable auto-check mode. If FALSE, disable.
strict	Logical. If TRUE, throw errors on violations. If FALSE, warn.
verbose	Logical. If TRUE, print diagnostic messages.

Value

Invisibly returns the previous state of auto-check options.

Examples

```
# Enable auto-checking with strict mode
borg_auto_check(TRUE)

# Disable auto-checking
borg_auto_check(FALSE)

# Enable with warnings instead of errors
borg_auto_check(TRUE, strict = FALSE)
```

---

borg_best_subset *Best Subset Variable Selection with Blocked CV*

Description

Evaluates all 2^p combinations of predictor variables using blocked CV. Exhaustive search — only feasible for small p (< 15).

Usage

```
borg_best_subset(
  data,
  target,
  predictors,
  folds,
  metric = c("rmse", "mae", "rsq"),
  fit_fun = stats::lm,
  max_vars = NULL,
  verbose = FALSE
)
```

Arguments

data	Data frame.
target	Character. Response variable.
predictors	Character vector. Candidate predictors.
folds	A borg_cv object or fold list.
metric	Character. Default: "rmse".
fit_fun	Function. Default: lm.
max_vars	Integer. Maximum variables in a subset. Default: all.
verbose	Logical. Default: FALSE.

Value

A data frame with class "borg_bss": variables, n_vars, metric_value, rank.

---

borg_block_size *Optimize Spatial Block Size*

Description

Tests multiple block sizes and selects the one that minimizes residual spatial autocorrelation between CV folds. For each candidate size, generates spatial block folds and computes the mean Moran's I of residuals within test sets.

Usage

```
borg_block_size(
  data,
  coords,
  target,
  v = 5,
  n_sizes = 10,
  range = NULL,
  formula = NULL,
  verbose = FALSE
)
```

Arguments

data	Data frame with coordinate and target columns.
coords	Character vector of length 2. Coordinate column names.
target	Character. Target variable name.
v	Integer. Number of folds. Default: 5.
n_sizes	Integer. Number of candidate block sizes to test. Default: 10.

range	Numeric vector of length 2. Min and max block sizes to test. If NULL, automatically determined from the variogram range estimate (0.5x to 3x range).
formula	Optional model formula for computing residual autocorrelation. If NULL, uses the raw target values.
verbose	Logical. Default: FALSE.

Value

A list with class "borg_block_opt" containing:

optimal Optimal block size

results Data frame with columns: block_size, mean_morans_i, mean_test_size, n_empty_folds

range_estimate Variogram-based range estimate

Has an autoplot() method showing the optimization curve.

Examples

```
set.seed(42)
d <- data.frame(
  x = runif(200, 0, 100), y = runif(200, 0, 100),
  z = rnorm(200)
)
opt <- borg_block_size(d, coords = c("x", "y"), target = "z")
opt$optimal
```

---

borg_bootstrap *Block Bootstrap Confidence Intervals for CV Metrics*

Description

Estimates confidence intervals for cross-validation performance metrics using spatial block bootstrap. Standard bootstrap assumes independent observations; block bootstrap preserves the dependency structure detected by borg_diagnose().

Usage

```
borg_bootstrap(
  model,
  data,
  target,
  coords = NULL,
  formula = NULL,
  fit_fun = NULL,
  folds = NULL,
  metric = c("rmse", "mae", "rsq"),
```

```
  n_boot = 200,
  n_blocks = 10,
  conf_level = 0.95,
  seed = 42
)
```

Arguments

model	A fitted model with a predict() method, or a model-fitting function (see fit_fun).
data	Data frame with predictors and target.
target	Character. Target variable name.
coords	Character vector of length 2. Coordinate column names. If provided, uses spatial blocking. Otherwise, uses random blocks.
formula	Model formula. Required if model is a function.
fit_fun	Function. Model fitting function. If NULL, attempts to refit from model's call.
folds	A borg_cv object or fold list. If provided, bootstrap resamples within the fold structure.
metric	Character. Performance metric: "rmse" (default), "mae", "rsq".
n_boot	Integer. Number of bootstrap replicates. Default: 200.
n_blocks	Integer. Number of spatial blocks for resampling. Default: 10.
conf_level	Numeric. Confidence level. Default: 0.95.
seed	Integer. Random seed. Default: 42.

Details

Spatial block bootstrap: Data is partitioned into spatial blocks using k-means clustering on coordinates. Each bootstrap replicate resamples blocks (with replacement), then includes all observations from selected blocks. This preserves within-block spatial correlation while generating valid resamples.

CI methods: Returns bias-corrected and accelerated (BCa) intervals when possible, falling back to percentile intervals.

Value

A list with class "borg_bootstrap" containing:

estimate Point estimate of the metric

ci_lower Lower confidence bound

ci_upper Upper confidence bound

conf_level Confidence level

boot_distribution Numeric vector of bootstrap estimates

se Bootstrap standard error

bias Bootstrap bias estimate

metric Metric name

n_boot Number of bootstrap replicates

method "spatial_block" or "random_block"

Has `print()` and `autoplot()` methods.

References

Lahiri, S. N. (2003). *Resampling Methods for Dependent Data*. Springer.

Examples

```
set.seed(42)
d <- data.frame(x = runif(150), y = runif(150), a = rnorm(150))
d$z <- 2 * d$a + rnorm(150, sd = 0.5)
model <- lm(z ~ a, data = d)
boot <- borg_bootstrap(model, d, target = "z", coords = c("x", "y"),
                       n_boot = 50)
boot
```

`borg_cache` *Cache and Retrieve BORG Diagnoses*

Description

Caches `BorgDiagnosis` objects keyed by a hash of the input data, so expensive computations (variograms, distance matrices, autocorrelation tests) are not repeated across iterations.

Usage

```
borg_cache_get(data, coords = NULL, target = NULL, envir = .borg_cache_env)

borg_cache_set(
  data,
  diagnosis,
  coords = NULL,
  target = NULL,
  envir = .borg_cache_env
)

borg_cache_clear(envir = .borg_cache_env)

borg_cache_info(envir = .borg_cache_env)
```

Arguments

data	A data frame. Used to compute the cache key.
coords	Character vector. Included in the cache key to distinguish diagnoses of the same data with different coordinate columns.
target	Character. Included in the cache key.
envir	Environment for in-memory cache. Default: the package namespace cache.
diagnosis	A BorgDiagnosis object to cache (for borg_cache_set), or NULL.

Value

borg_cache_get A BorgDiagnosis or NULL if not cached.

borg_cache_set Invisible NULL. Stores the diagnosis.

borg_cache_clear Invisible NULL. Clears all cached diagnoses.

borg_cache_info A data frame with cache key, timestamp, and data dimensions for all cached entries.

Examples

```
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
diag <- borg_diagnose(d, coords = c("x", "y"), target = "z")

# Cache it
borg_cache_set(d, diag, coords = c("x", "y"), target = "z")

# Retrieve (fast, no recomputation)
cached <- borg_cache_get(d, coords = c("x", "y"), target = "z")
identical(diag, cached)  # TRUE

# Clear all
borg_cache_clear()
```

---

borg_calibration *Model Calibration Diagnostics*

Description

Assesses whether a model's predictions are well-calibrated. For classification, checks if predicted probabilities match observed frequencies. For regression, checks if predicted quantiles have correct coverage. Optionally uses spatial-aware binning to avoid autocorrelation artifacts in calibration curves.

Usage

```
borg_calibration(
  predicted,
  actual,
  type = NULL,
  n_bins = 10,
  coords = NULL,
  strategy = c("uniform", "quantile", "spatial")
)
```

Arguments

predicted	Numeric vector. Predicted values (probabilities for classification, point predictions for regression).
actual	Numeric or factor vector. Observed outcomes (0/1 for classification, continuous for regression).
type	Character. "classification" (default if actual is 0/1 or factor) or "regression".
n_bins	Integer. Number of bins for calibration curve. Default: 10.
coords	Data frame or matrix with coordinate columns. If provided, uses spatial-aware binning (ensures bins are not spatially clustered).
strategy	Character. Binning strategy: "uniform" (equal-width, default), "quantile" (equal-count), or "spatial" (spatial blocks as bins, requires coords).

Value

A list with class "borg_calibration" containing:

calibration_curve Data frame with columns: bin_midpoint, observed_freq, predicted_mean, n, ci_lower, ci_upper

ece Expected Calibration Error (weighted mean |observed - predicted|)

mce Maximum Calibration Error (worst bin)

brier_score Brier score (classification) or calibration MSE (regression)

type Model type

reliability_slope Slope of observed ~ predicted regression (1.0 = perfect calibration)

reliability_intercept Intercept (0.0 = no bias)

n_bins Number of bins used

assessment Character: "well_calibrated", "moderate", or "poorly_calibrated"

Has `print()` and `autoplot()` methods.

Examples

```
# Classification
set.seed(42)
probs <- runif(500)
outcomes <- rbinom(500, 1, probs^0.8)  # slightly miscalibrated
cal <- borg_calibration(probs, outcomes)
cal

# Regression
x <- rnorm(200)
y <- 2 * x + rnorm(200, sd = 0.5)
preds <- 2.1 * x  # slightly biased
cal_reg <- borg_calibration(preds, y, type = "regression")
cal_reg
```

---

borg_certificate *Create Validation Certificate*

Description

Generate a structured validation certificate documenting the BORG analysis for reproducibility and audit trails.

Usage

```
borg_certificate(diagnosis, data, comparison = NULL, cv = NULL)
```

Arguments

diagnosis	A BorgDiagnosis object.
data	The data frame that was analyzed.
comparison	Optional. A borg_comparison object with empirical inflation estimates.
cv	Optional. A borg_cv object with the CV folds used.

Value

A borg_certificate object containing:

- meta: Package version, R version, timestamp
- data: Data characteristics and hash
- diagnosis: Dependency type, severity, recommended CV
- cv_strategy: CV type and fold count
- inflation: Theoretical and empirical estimates

See Also

borg_export for writing certificates to file.

Examples

```
set.seed(42)
data <- data.frame(
  x = runif(100, 0, 100),
  y = runif(100, 0, 100),
  response = rnorm(100)
)
diagnosis <- borg_diagnose(data, coords = c("x", "y"), target = "response",
                           verbose = FALSE)
cert <- borg_certificate(diagnosis, data)
print(cert)
```

---

borg_check *Quick Leakage Check (Pipe-Friendly)*

Description

Single-verb entry point that runs the full BORG pipeline: diagnose dependencies, generate valid CV, validate the split, and return a tidy summary. Designed for use in pipelines.

Usage

```
borg_check(
  data,
  model = NULL,
  target = NULL,
  coords = NULL,
  time = NULL,
  groups = NULL,
  train_idx = NULL,
  test_idx = NULL,
  v = 5L,
  verbose = FALSE
)
```

Arguments

data	A data frame.
model	A fitted model object (optional). If provided, predictions are evaluated under blocked CV.
target	Character. Response variable name.

coords	Character vector of length 2. Coordinate column names (optional, for spatial data).
time	Character. Time column name (optional, for temporal data).
groups	Character. Grouping column name (optional).
train_idx	Integer vector. Training indices (optional, for validating an existing split).
test_idx	Integer vector. Test indices (optional).
v	Integer. Number of CV folds. Default: 5.
verbose	Logical. Default: FALSE.

Details

`borg_check()` is intentionally simple: one call, one result. For finer control, use `borg_diagnose()`, `borg_cv()`, and `borg_validate()` individually.

Value

A data frame with class `"borg_check"` containing one row per detected risk (or zero rows if clean), with columns:

risk_type Character. Type of risk detected.

severity Character. `"hard_inflation"`, `"soft_inflation"`, or `"info"`.

description Character. Plain-language description.

n_affected Integer. Number of observations affected.

source Character. Object/step that triggered the risk.

Also has attributes: `diagnosis`, `cv`, `risks` (the full objects for further inspection).

See Also

borg, borg_diagnose, borg_explain_risk

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))

# Pipe-friendly
result <- borg_check(d, target = "z", coords = c("x", "y"))
result
nrow(result)  # 0 if clean

# Check an existing split
result <- borg_check(d, target = "z", coords = c("x", "y"),
                     train_idx = 1:70, test_idx = 71:100)
```

`borg_check_coverage` *Check Geographic Representativeness of CV Folds*

Description

Evaluates whether each fold's test set covers a representative portion of the study area. Flags folds that are geographically isolated or biased toward one region.

Usage

```
borg_check_coverage(folds, data, coords, threshold = 0.2)
```

Arguments

`folds`	A `borg_cv` object or list of fold lists.
`data`	Data frame with coordinate columns.
`coords`	Character vector of length 2. Coordinate column names.
`threshold`	Numeric. Minimum proportion of geographic extent that each fold should cover (0-1). Default: 0.2 (each fold covers at least 20 percent of the x and y extent).

Value

A data frame with class `"borg_geo_strat"` containing per-fold coverage metrics:

fold Fold index

x_coverage Proportion of x-extent covered by test set

y_coverage Proportion of y-extent covered by test set

area_ratio Convex hull area ratio (test / total)

centroid_x, centroid_y Test set centroid

balanced Whether fold meets the threshold

Examples

```
set.seed(42)
d <- data.frame(x = runif(200, 0, 100), y = runif(200, 0, 100), z = rnorm(200))
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
strat <- borg_check_coverage(cv, d, coords = c("x", "y"))
strat
```

`borg_check_nested_cv` *Check Nested CV for Leakage*

Description

Validates a nested cross-validation setup for data leakage between outer and inner CV loops, and detects strategy mismatches where the inner CV uses random resampling despite data dependencies.

Usage

```
borg_check_nested_cv(
  inner_resamples,
  outer_train_idx,
  outer_test_idx,
  data,
  coords = NULL,
  time = NULL,
  groups = NULL
)
```

Arguments

`inner_resamples`	The inner resampling object. One of: • A `caret::train` object • A `tune_results` object from tidymodels • A character string naming the inner CV method (e.g. "cv", "vfold_cv")
`outer_train_idx`	Integer vector. Indices of the outer training set.
`outer_test_idx`	Integer vector. Indices of the outer test set.
`data`	Data frame used for modeling.
`coords`	Character vector of coordinate column names (for spatial check).
`time`	Character. Time column name (for temporal check).
`groups`	Character. Group column name (for clustered check).

Value

A `BorgRisk` object with any detected risks.

Examples

```
# Check if inner random CV is appropriate given grouped data
d <- data.frame(
  site = rep(1:20, each = 10),
  x = rnorm(200),
```

```
  y = rep(rnorm(20), each = 10) + rnorm(200, sd = 0.5)
)
result <- borg_check_nested_cv(
  inner_resamples = "cv",
  outer_train_idx = 1:160,
  outer_test_idx = 161:200,
  data = d,
  groups = "site"
)
```

`borg_check_residuals` *Check Residual Spatial Autocorrelation*

Description

After fitting a model, checks whether residuals still exhibit spatial autocorrelation. If they do, the model has not fully captured the spatial process and predictions may be biased.

Usage

```
borg_check_residuals(model, data = NULL, coords = NULL, alpha = 0.05)
```

Arguments

`model`	A fitted model with a `residuals()` method, or a numeric vector of residuals.
`data`	Data frame with coordinate columns (needed if `model` is a numeric vector).
`coords`	Character vector of length 2. Coordinate column names.
`alpha`	Numeric. Significance level for Moran's I test. Default: 0.05.

Value

A list with class `"borg_residual_check"` containing:

morans_i Moran's I of residuals

p_value P-value from Moran's I test

significant Logical. Whether residual autocorrelation is significant

variogram Residual variogram data frame (if computed)

assessment Character. "clean", "mild", or "strong"

Has an `autoplot()` method showing the residual variogram.

Examples

```
set.seed(42)
d <- data.frame(x = runif(100, 0, 100), y = runif(100, 0, 100))
d$z <- sin(d$x / 10) + rnorm(100, sd = 0.5)
model <- lm(z ~ x + y, data = d)
check <- borg_check_residuals(model, d, coords = c("x", "y"))
check
```

---

borg_compare_cv *Compare Random vs Blocked Cross-Validation*

Description

Runs both random and blocked cross-validation on the same data and model, providing empirical evidence of metric inflation from ignoring data dependencies.

Usage

```
borg_compare_cv(
  data,
  formula,
  model_fn = NULL,
  predict_fn = NULL,
  metric = NULL,
  diagnosis = NULL,
  coords = NULL,
  time = NULL,
  groups = NULL,
  target = NULL,
  v = 5,
  repeats = 10,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

data	A data frame containing predictors and response.
formula	A formula specifying the model (e.g., y ~ .).
model_fn	A function that fits a model. Should accept formula and data arguments and return a fitted model with a predict method. Default uses lm.
predict_fn	A function to generate predictions. Should accept model and newdata arguments. Default uses predict.
metric	A character string specifying the metric to compute. One of "rmse", "mae", "rsq", "auc", "accuracy". Default: "rmse" for regression, "accuracy" for classification.

`diagnosis`	A `BorgDiagnosis` object. If NULL, will be computed automatically using the provided structure hints.
`coords`	Character vector of length 2 specifying coordinate column names.
`time`	Character string specifying the time column name.
`groups`	Character string specifying the grouping column name.
`target`	Character string specifying the response variable name. If NULL, extracted from formula.
`v`	Integer. Number of CV folds. Default: 5.
`repeats`	Integer. Number of times to repeat CV. Default: 10 for stable estimates.
`seed`	Integer. Random seed for reproducibility.
`verbose`	Logical. Print progress messages. Default: TRUE.

Details

This function provides the "smoking gun" evidence for reviewers. It runs cross-validation twice on the same data:

1. **Random CV**: Standard k-fold CV ignoring data structure
2. **Blocked CV**: Structure-aware CV based on BORG diagnosis

The difference in metrics demonstrates empirically how much random CV inflates performance estimates when data dependencies exist.

For stable estimates, the comparison is repeated multiple times (default: 10) and a paired t-test assesses whether the difference is statistically significant.

Value

A `borg_comparison` object (S3 class) containing:

random_cv Data frame of metrics from random CV (one row per repeat)

blocked_cv Data frame of metrics from blocked CV (one row per repeat)

summary Summary statistics comparing the two approaches

inflation Estimated metric inflation from using random CV

diagnosis The BorgDiagnosis object used

p_value P-value from paired t-test comparing approaches

See Also

`borg_diagnose` for dependency detection, `borg_cv` for generating blocked CV folds.

Examples

```
# Spatial data example
set.seed(42)
n <- 200
spatial_data <- data.frame(
  x = runif(n, 0, 100),
  y = runif(n, 0, 100)
)
# Create spatially autocorrelated response
spatial_data$response <- spatial_data$x * 0.5 + rnorm(n, sd = 5)

# Compare CV approaches
comparison <- borg_compare_cv(
  spatial_data,
  formula = response ~ x + y,
  coords = c("x", "y"),
  repeats = 5  # Use more repeats in practice
)

print(comparison)
plot(comparison)
```

---

borg_compare_models *Compare Multiple Models with Spatial CV*

Description

Evaluates multiple models (or formulas) using the same blocked CV folds, producing a side-by-side comparison table.

Usage

```
borg_compare_models(
  data,
  folds,
  models,
  metric = c("rmse", "mae", "rsq", "auc", "tss", "kappa"),
  fit_fun = stats::lm
)
```

Arguments

data	Data frame.
folds	A borg_cv object or fold list.
models	Named list of either: • Formulas: list(lm = y ~ x, full = y ~ x + z)

- Fitted models: `list(lm = lm_fit, rf = rf_fit)`

`metric`	Character. Default: `"rmse"`.
`fit_fun`	Function. Used when `models` contains formulas. Default: `lm`.

Value

A data frame with class `"borg_model_comparison"` containing model name, mean metric, SD, and rank. Has `autoplot()` method.

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100),
                a = rnorm(100), b = rnorm(100))
d$z <- d$a * 2 + rnorm(100, sd = 0.5)
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
comp <- borg_compare_models(d, cv,
  models = list(simple = z ~ a, full = z ~ a + b))
comp
```

---

borg_conformal *Conformal Prediction with Spatial Dependence*

Description

Constructs distribution-free prediction intervals with finite-sample coverage guarantees, adjusted for spatial autocorrelation. Standard conformal prediction assumes exchangeability, which spatial data violates. This function uses spatially-blocked calibration residuals as nonconformity scores, producing intervals that maintain coverage even under dependence.

Usage

```
borg_conformal(
  model,
  data,
  new = NULL,
  target,
  coords = NULL,
  alpha = 0.1,
  method = c("split", "block_jackknife", "block_cv"),
  folds = NULL,
  n_blocks = 10,
  type = c("regression", "classification"),
  seed = 42
)
```

Arguments

model	A fitted model with a predict() method.
data	Data frame used to compute nonconformity scores (calibration set).
new	Data frame of new locations for prediction. If NULL, computes intervals for data via leave-one-block-out.
target	Character. Target variable name in data.
coords	Character vector of length 2. Coordinate column names. If provided, uses spatial blocking for calibration.
alpha	Numeric. Miscoverage level. Default: 0.1 (90% intervals).
method	Character. Conformal method: "split" (default) uses a held-out calibration set, "block_jackknife" uses leave-one-block-out, "block_cv" uses blocked cross-validation residuals.
folds	A borg_cv object or fold list. If provided, residuals are computed from these folds (respecting spatial structure). Overrides internal blocking.
n_blocks	Integer. Number of spatial blocks if folds is NULL and coords is provided. Default: 10.
type	Character. For classification: "regression" (default) or "classification". Classification uses adaptive prediction sets.
seed	Integer. Random seed. Default: 42.

Details

Why spatial conformal?: Standard split conformal prediction computes residuals on a random calibration set. Under spatial autocorrelation, nearby calibration points produce correlated residuals, leading to underestimated interval widths and actual coverage below the nominal level.

By using spatially-blocked calibration (where residuals come from predictions on spatially separated test folds), the effective sample size of nonconformity scores is honest, and coverage guarantees hold approximately even under dependence.

Methods:

"split" Splits data into training and calibration sets using spatial blocks. Fast, but uses only part of the data.

"block_jackknife" Leave-one-block-out: refit the model excluding each block, predict on the held-out block. More data-efficient but slower.

"block_cv" Use pre-computed blocked CV residuals from a borg_cv object. Requires folds.

Value

A data frame with class "borg_conformal" containing:

prediction Point prediction

lower Lower bound of prediction interval

upper Upper bound of prediction interval

width Interval width (upper - lower)

x and y Coordinates (if `coords` provided and `new` has them)

Attributes include `alpha`, `method`, `coverage_estimate`, and `quantile_score` (the nonconformity threshold).

References

Mao, H., Martin, R., & Reich, B. J. (2024). Valid prediction inference with spatial conformal methods. *arXiv preprint arXiv:2403.14058*.

Johnstone, C., & Cox, D. (2023). Conformal prediction with spatial data.

Vovk, V., Gammerman, A., & Shafer, G. (2005). *Algorithmic Learning in a Random World*. Springer.

Examples

```
set.seed(42)
d <- data.frame(x = runif(200), y = runif(200), a = rnorm(200))
d$z <- 2 * d$a + sin(d$x * 10) + rnorm(200, sd = 0.5)
model <- lm(z ~ a + x + y, data = d)

# Split conformal with spatial blocking
conf <- borg_conformal(model, d, target = "z", coords = c("x", "y"))
conf

# Predict on new locations
new <- data.frame(x = runif(50), y = runif(50), a = rnorm(50))
pred <- borg_conformal(model, d, new = new, target = "z",
                       coords = c("x", "y"))
```

---

borg_cv *Generate Valid Cross-Validation Scheme*

Description

Creates cross-validation folds that respect data dependency structure. When spatial, temporal, or clustered dependencies are detected, random CV is disabled and appropriate blocking strategies are enforced.

Usage

```
borg_cv(
  data,
  diagnosis = NULL,
  v = 5,
  coords = NULL,
  time = NULL,
  groups = NULL,
```

```
  target = NULL,
  env = NULL,
  dist_mat = NULL,
  prediction_points = NULL,
  block_size = NULL,
  embargo = NULL,
  buffer = NULL,
  strategy = NULL,
  repeats = 1L,
  output = c("list", "rsample", "caret", "mlr3"),
  allow_random = FALSE,
  verbose = FALSE
)
```

Arguments

data	A data frame to create CV folds for.
diagnosis	A BorgDiagnosis object from borg_diagnose. If NULL, diagnosis is performed automatically.
v	Integer. Number of folds. Default: 5.
coords	Character vector of length 2 specifying coordinate column names. Required for spatial blocking if diagnosis is NULL.
time	Character string specifying the time column name. Required for temporal blocking if diagnosis is NULL.
groups	Character string specifying the grouping column name. Required for group CV if diagnosis is NULL.
target	Character string specifying the response variable column name.
env	Environmental covariates for environmental blocking. A SpatRaster, data frame, or matrix with one row per observation. Required when strategy = "environmental_block" or "spatial_plus".
dist_mat	A distance matrix or dist object for custom blocking. Must be an n x n matrix matching nrow(data). Required when strategy = "custom_block".
prediction_points	Data frame, matrix, SpatVector, or sf object with coordinates of prediction locations. Required when strategy = "knndm".
block_size	Numeric. For spatial blocking, the minimum block size. If NULL, automatically determined from diagnosis. Should be larger than the autocorrelation range.
embargo	Integer. For temporal blocking, minimum gap between train and test. If NULL, automatically determined from diagnosis.
buffer	Numeric. Spatial buffer distance (in coordinate units). Training points within this distance of any test-fold point are removed to reduce autocorrelation leakage. Default: NULL (no buffer).
strategy	Character. Override the auto-detected CV strategy. Use "temporal_expanding" for expanding window (forward-chaining) or "temporal_sliding" for fixed-size sliding window. Default: NULL (auto-detect from diagnosis).

repeats	Integer. Number of times to repeat CV fold generation with different random seeds. Default: 1 (no repetition).
output	Character. Output format: "list" (default), "rsample", "caret", "mlr3".
allow_random	Logical. If TRUE, allows random CV even when dependencies detected. Default: FALSE. Setting to TRUE requires explicit acknowledgment.
verbose	Logical. If TRUE, print diagnostic messages. Default: FALSE.

Details

The Enforcement Principle: Unlike traditional CV helpers, borg_cv enforces valid evaluation:

- If spatial autocorrelation is detected, **random CV is disabled**
- If temporal autocorrelation is detected, **random CV is disabled**
- If clustered structure is detected, **random CV is disabled**
- To use random CV on dependent data, you must set allow_random = TRUE and provide justification (this is logged).

Spatial Blocking: When spatial dependencies are detected, data are partitioned into spatial blocks using k-means clustering on coordinates. Block size is set to exceed the estimated autocorrelation range. This ensures train and test sets are spatially separated.

Temporal Blocking: When temporal dependencies are detected, data are split chronologically with an embargo period between train and test sets. This prevents information from future observations leaking into training.

Group CV: When clustered structure is detected, entire groups (clusters) are held out together. No group appears in both train and test within a fold.

Value

Depending on output:

"list" A list with elements: folds (list of train/test index vectors), diagnosis (the BorgDiagnosis used), strategy (CV strategy name), params (parameters used).

"rsample" An rsample rset object compatible with tidymodels.

"caret" A trainControl object for caret.

"mlr3" An mlr3 Resampling object.

See Also

borg_diagnose, BorgDiagnosis

Examples

```
# Spatial data with autocorrelation
set.seed(42)
spatial_data <- data.frame(
  x = runif(200, 0, 100),
  y = runif(200, 0, 100),
```

```
  response = rnorm(200)
)

# Diagnose and create CV
cv <- borg_cv(spatial_data, coords = c("x", "y"), target = "response")
str(cv$folds)  # List of train/test indices

# Clustered data
clustered_data <- data.frame(
  site = rep(1:20, each = 10),
  value = rep(rnorm(20, sd = 2), each = 10) + rnorm(200, sd = 0.5)
)

cv <- borg_cv(clustered_data, groups = "site", target = "value")
cv$strategy  # "group_fold"

# Get rsample-compatible output for tidymodels

cv_rsample <- borg_cv(spatial_data, coords = c("x", "y"), output = "rsample")
```

`borg_debias` *Spatial+ Debiasing for Spatial Confounding*

Description

Implements the Spatial+ approach of Dupont et al. (2022) to address spatial confounding. Spatially structured predictors share variance with the spatial process, inflating their coefficients. Spatial+ removes the spatial component from each predictor by regressing it on smooth spatial coordinates, then uses the residuals as debiased predictors.

Usage

```
borg_debias(
  data,
  predictors = NULL,
  coords,
  target = NULL,
  method = c("gam_approx", "tps"),
  df = 6,
  keep_original = FALSE
)
```

Arguments

`data`	Data frame with predictors and coordinates.
`predictors`	Character vector. Predictor column names to debias. If `NULL`, uses all numeric columns except `coords` and `target`.

`coords`	Character vector of length 2. Coordinate column names.
`target`	Character. Target variable name (excluded from debiasing).
`method`	Character. Spatial smoothing method: `"gam_approx"` (default) uses polynomial basis (no mgcv dependency), `"tps"` uses thin plate spline approximation.
`df`	Integer. Degrees of freedom for the spatial smooth. Default: 6. Higher values capture more spatial structure but risk removing real signal.
`keep_original`	Logical. If TRUE, returns both original and debiased columns (debiased columns have suffix `_debiased`). Default: FALSE.

Details

When to use: Use when your predictors have spatial structure (e.g. climate variables that vary smoothly over space). If `borg_diagnose()` detects spatial autocorrelation in both residuals and predictors, spatial confounding is likely. Debiasing is especially important for inference (coefficient interpretation) rather than prediction.

How it works: For each predictor X_j:

1. Fit $X_j \sim f(s_1, s_2)$ where f is a smooth function of coordinates.
2. Replace X_j with the residuals $X_j - \hat{f}(s_1, s_2)$.

The residuals contain only the non-spatial variation in X_j.

Value

A list with class `"borg_debias"` containing:

data Debiased data frame (original columns replaced or augmented)

spatial_r2 Named numeric vector. R-squared of spatial smooth for each predictor (how much spatial structure was removed)

predictors Debiased predictor column names

method Smoothing method used

df Degrees of freedom

assessment Character: "minimal" (less than 10 percent spatial variance), "moderate" (10-40 percent), or "substantial" (over 40 percent)

Has `print()` and `autoplot()` methods.

References

Dupont, E., Wood, S. N., & Augustin, N. H. (2022). Spatial+: A novel approach to spatial confounding. *Biometrics*, 78(4), 1279-1290. doi:10.1111/biom.13656

Examples

```
set.seed(42)
d <- data.frame(
  x = runif(200, 0, 100), y = runif(200, 0, 100),
  temp = NA, elev = rnorm(200)
)
# temp has spatial structure
d$temp <- sin(d$x / 20) + cos(d$y / 20) + rnorm(200, sd = 0.3)
d$z <- 0.5 * d$temp + d$elev + rnorm(200, sd = 0.5)

db <- borg_debias(d, coords = c("x", "y"), target = "z")
db
```

---

borg_di *Dissimilarity Index*

Description

Computes the weighted Euclidean distance in feature space from each point to its nearest training observation. Points with high DI are dissimilar to the training data and predictions may be unreliable.

Usage

```
borg_di(train, new = NULL, train_idx = NULL, predictors = NULL, weights = NULL)
```

Arguments

`train`	Data frame of training predictors (or full data with `train_idx`).
`new`	Data frame of new/prediction locations with the same predictor columns. If `NULL`, computes DI for training data (leave-one-out).
`train_idx`	Integer vector. If provided, `train` is the full dataset and `train_idx` selects training rows.
`predictors`	Character vector. Predictor column names. If `NULL`, uses all shared numeric columns.
`weights`	Numeric vector. Variable importance weights (length = number of predictors). If `NULL`, all variables weighted equally. Typically from `borg_importance()`.

Value

A numeric vector of DI values (one per row of `new`, or per training row if `new` is NULL). Has class `"borg_di"` and attribute `"threshold"` (mean + sd of training DI, used as default AOA cutoff).

References

Meyer, H., & Pebesma, E. (2021). Predicting into unknown space? Estimating the area of applicability of spatial prediction models. *Methods in Ecology and Evolution*, 12(9), 1620-1633. doi:10.1111/2041210X.13650

Examples

```
set.seed(42)
train <- data.frame(a = rnorm(80), b = rnorm(80))
new <- data.frame(a = rnorm(20, mean = 3), b = rnorm(20))
di <- borg_di(train, new)
summary(di)
```

`borg_diagnose` *Diagnose Data Dependency Structure*

Description

Automatically detects spatial autocorrelation, temporal autocorrelation, and clustered structure in data. Returns a diagnosis object that specifies appropriate cross-validation strategies.

Usage

```
borg_diagnose(
  data,
  coords = NULL,
  time = NULL,
  groups = NULL,
  target = NULL,
  alpha = 0.05,
  verbose = FALSE
)
```

Arguments

`data`	A data frame, `sf` object, or `terra::SpatVector` to diagnose. For `sf` and `SpatVector` inputs, coordinates are extracted automatically and the `coords` argument is not needed.
`coords`	Character vector of length 2 specifying coordinate column names (e.g., `c("lon", "lat")` or `c("x", "y")`). If NULL, spatial autocorrelation is not tested.
`time`	Character string specifying the time column name. Can be Date, POSIXct, or numeric. If NULL, temporal autocorrelation is not tested.
`groups`	Character string specifying the grouping column name (e.g., "site_id", "patient_id"). If NULL, clustered structure is not tested.

target	Character string specifying the response variable column name. Used for more accurate autocorrelation diagnostics on residuals. Optional.
alpha	Numeric. Significance level for autocorrelation tests. Default: 0.05.
verbose	Logical. If TRUE, print diagnostic progress. Default: FALSE.

Details

Spatial Autocorrelation: Detected using Moran's I test on the target variable (or first numeric column). The autocorrelation range is estimated from the empirical variogram. Effective sample size is computed as $n_{eff} = n/DEFF$ where DEFF is the design effect.

Temporal Autocorrelation: Detected using the Ljung-Box test on the target variable. The decorrelation lag is the first lag where ACF drops below the significance threshold. Minimum embargo period is set to the decorrelation lag.

Clustered Structure: Detected by computing the intraclass correlation coefficient (ICC). An ICC > 0.05 indicates meaningful clustering. The design effect (DEFF) quantifies variance inflation: $DEFF = 1 + (m - 1) \times ICC$ where m is the average cluster size.

Value

A BorgDiagnosis object containing:

- Detected dependency type(s)
- Severity assessment
- Recommended CV strategy
- Detailed diagnostics for each dependency type
- Estimated metric inflation from using random CV

Examples

```
# Spatial data example
set.seed(42)
spatial_data <- data.frame(
  x = runif(100, 0, 100),
  y = runif(100, 0, 100),
  response = rnorm(100)
)
# Add spatial autocorrelation (nearby points are similar)
for (i in 2:100) {
  nearest <- which.min((spatial_data$x[1:(i-1)] - spatial_data$x[i])^2 +
                       (spatial_data$y[1:(i-1)] - spatial_data$y[i])^2)
  spatial_data$response[i] <- 0.7 * spatial_data$response[nearest] +
                              0.3 * rnorm(1)
}

diagnosis <- borg_diagnose(spatial_data, coords = c("x", "y"),
                           target = "response")
print(diagnosis)
```

```
# Clustered data example
clustered_data <- data.frame(
  site = rep(1:10, each = 20),
  value = rep(rnorm(10, sd = 2), each = 20) + rnorm(200, sd = 0.5)
)

diagnosis <- borg_diagnose(clustered_data, groups = "site", target = "value")
print(diagnosis)
```

`borg_disc_cv` *Leave-Disc-Out Cross-Validation*

Description

Spatial cross-validation with circular exclusion buffers around test points. For each fold, all training points within a specified radius of any test point are removed, creating a spatial gap that eliminates autocorrelation leakage. More principled than rectangular or k-means blocking for point data.

Usage

```
borg_disc_cv(
  data,
  coords,
  target = NULL,
  radius = NULL,
  v = 5,
  min_train = 0.5,
  seed = 42,
  output = c("list", "rsample"),
  verbose = FALSE
)
```

Arguments

`data`	A data frame with coordinate and predictor columns.
`coords`	Character vector of length 2. Coordinate column names.
`target`	Character. Target variable name (for diagnostics).
`radius`	Numeric. Exclusion buffer radius in coordinate units. If `NULL`, automatically set to the autocorrelation range estimated by `borg_diagnose()`.
`v`	Integer. Number of folds. Default: 5.
`min_train`	Numeric. Minimum fraction of data that must remain in training after exclusion. Folds that violate this are dropped. Default: 0.5.
`seed`	Integer. Random seed. Default: 42.
`output`	Character. Output format: `"list"` (default), `"rsample"`.
`verbose`	Logical. Print diagnostic messages. Default: FALSE.

Details

How it works:

1. Partition data into v spatial folds using k-means on coordinates (same as borg_cv).
2. For each fold, identify the test points.
3. Remove from training all points within radius of any test point. These excluded points are neither train nor test — they form a buffer zone.
4. If the remaining training set is too small (< min_train * n), the fold is dropped with a warning.

Choosing the radius: The radius should match the autocorrelation range of the spatial process. borg_diagnose() estimates this via variogram analysis. Setting radius = NULL uses this estimate automatically.

Value

A list with class "borg_disc_cv" containing:

folds List of train/test index vectors

radius Exclusion radius used

n_excluded Number of training points excluded per fold (in the buffer zone)

effective_training Fraction of data available for training per fold (after exclusion)

strategy "leave_disc_out"

params List of parameters used

Compatible with other BORG functions that accept fold lists.

Examples

```
set.seed(42)
d <- data.frame(x = runif(200, 0, 100), y = runif(200, 0, 100))
d$z <- sin(d$x / 10) + rnorm(200, sd = 0.5)
cv <- borg_disc_cv(d, coords = c("x", "y"), target = "z", radius = 15)
cv
```

---

borg_drift *Detect Distribution Shift Between Training and Deployment Data*

Description

Goes beyond Area of Applicability (AOA) by quantifying *how* the distribution changed and *which features* drifted. Combines univariate tests (Kolmogorov-Smirnov per feature) with a multivariate classifier two-sample test (train a model to distinguish train from deployment; AUC > 0.5 indicates shift).

Usage

```
borg_drift(
  train,
  new,
  predictors = NULL,
  alpha = 0.05,
  n_perm = 100,
  seed = 42
)
```

Arguments

train	Data frame of training data.
new	Data frame of deployment/prediction data.
predictors	Character vector. Predictor column names. If NULL, uses all shared numeric columns.
alpha	Numeric. Significance level for per-feature KS tests. Default: 0.05.
n_perm	Integer. Number of permutations for the classifier two-sample test p-value. Default: 100. Set to 0 to skip.
seed	Integer. Random seed. Default: 42.

Details

Univariate tests: For each feature, a two-sample Kolmogorov-Smirnov test detects distributional differences. Effect size is measured by Cohen's d (standardized mean difference). P-values are Bonferroni-corrected.

Multivariate classifier test: A logistic regression is trained to distinguish training from deployment observations. If the data distributions are identical, the classifier achieves AUC ~ 0.5. Higher AUC indicates multivariate shift that may not be captured by univariate tests alone. Statistical significance is assessed via permutation.

Value

A list with class "borg_drift" containing:

feature_drift Data frame with per-feature KS statistic, p-value, effect_size (Cohen's d), shifted (logical), and direction ("higher", "lower", "similar")

n_shifted Number of features with significant drift

classifier_auc AUC of train-vs-deployment classifier (0.5 = no shift, 1.0 = complete separation)

classifier_pvalue Permutation p-value for AUC

overall_severity Character: "none", "mild", "moderate", "severe"

summary One-sentence summary

Has `print()` and `autoplot()` methods.

References

Ginsberg, T., Liang, Z., & Krishnan, R. G. (2023). A learning based hypothesis test for harmful covariate shift. *ICLR*.

Lopez-Paz, D., & Oquab, M. (2017). Revisiting classifier two-sample tests. *ICLR*.

Examples

```
set.seed(42)
train <- data.frame(a = rnorm(200), b = rnorm(200), c = rnorm(200))
# Deployment: feature 'a' has shifted
deploy <- data.frame(a = rnorm(100, mean = 1), b = rnorm(100),
                     c = rnorm(100))
drift <- borg_drift(train, deploy)
drift
```

---

borg_ensemble *Ensemble Predictions from CV Fold Models*

Description

Combines predictions from models fitted on each CV fold into a weighted ensemble. Each fold's model predicts on the full dataset (or new data), and predictions are averaged with optional performance-based weighting.

Usage

```
borg_ensemble(
  data,
  folds,
  formula,
  newdata = NULL,
  fit_fun = stats::lm,
  weight_by = c("equal", "performance"),
  metric = c("rmse", "mae")
)
```

Arguments

`data`	Data frame used for training.
`folds`	A `borg_cv` object or fold list.
`formula`	Model formula.
`newdata`	Optional data frame for prediction. If `NULL`, predicts on `data`.
`fit_fun`	Function. Default: `lm`.
`weight_by`	Character. Weighting scheme: `"equal"` (default) or `"performance"` (inverse error weight).
`metric`	Character. Metric for performance weighting. Default: `"rmse"`.

Value

A list with class "borg_ensemble" containing:

prediction Numeric vector of ensemble predictions

uncertainty Per-observation SD across fold predictions

weights Fold weights used

n_models Number of contributing models

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
ens <- borg_ensemble(d, cv, z ~ x + y)
cor(d$z, ens$prediction)
```

---

borg_error_profile *Calibrate Dissimilarity Index to Prediction Error*

Description

Models the relationship between the Dissimilarity Index (DI) and prediction error using isotonic regression. Enables pixel-level uncertainty estimation: for any new prediction point, its DI can be mapped to an expected error.

Usage

```
borg_error_profile(
  data,
  folds,
  formula,
  predictors = NULL,
  metric = c("rmse", "mae"),
  fit_fun = stats::lm,
  n_bins = 10
)
```

Arguments

data	Data frame of training data.
folds	A borg_cv object or fold list.
formula	Model formula.
predictors	Character vector. Predictor columns for DI computation. If NULL, auto-detected.
metric	Character. "rmse" or "mae".
fit_fun	Function. Default: lm.
n_bins	Integer. Number of DI bins. Default: 10.

Value

A list with class "borg_error_profile" containing:

profile Data frame: di_bin, mean_di, mean_error, n_obs

raw Data frame: di, error (per observation)

iso_fit Isotonic regression fit for predicting error from DI

Has an autoplot() method and a predict() method.

References

Meyer, H., & Pebesma, E. (2021). Predicting into unknown space? Estimating the area of applicability of spatial prediction models. *Methods in Ecology and Evolution*, 12(9), 1620-1633. doi:10.1111/2041210X.13650

Examples

```
set.seed(42)
d <- data.frame(x = runif(150, 0, 100), y = runif(150, 0, 100),
                a = rnorm(150), b = rnorm(150))
d$z <- d$a * 2 + rnorm(150, sd = 0.5)
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
ep <- borg_error_profile(d, cv, z ~ a + b, predictors = c("a", "b"))
ep
```

---

borg_explain_risk *Explain Risks in Plain Language with Actionable Recommendations*

Description

Takes a BorgRisk or borg_result object and returns a prioritized, human-readable action plan. Each risk is translated into what went wrong, why it matters, how to fix it, and the expected inflation magnitude.

Usage

```
borg_explain_risk(x, style = c("console", "markdown", "list"), verbose = FALSE)
```

Arguments

x	A BorgRisk or borg_result object.
style	Character. Output style: "console" (default, prints to terminal), "markdown" (returns markdown string), or "list" (returns structured list).
verbose	Logical. If TRUE, include background explanations. Default: FALSE.

Value

Depending on `style`:

"console" Invisible `x`, prints explanation.

"markdown" Character string with markdown-formatted explanation.

"list" A list of per-risk explanation objects, each with fields: `risk_type`, `severity`, `plain_english`, `why_it_matters`, `how_to_fix`, `inflation_est`.

See Also

`borg`, `BorgRisk`, `borg_assimilate`

Examples

```
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
result <- borg(d, coords = c("x", "y"), target = "z",
               train_idx = 1:70, test_idx = 71:100)
borg_explain_risk(result)
```

`borg_export` *Export Validation Certificate*

Description

Write a BORG validation certificate to a YAML or JSON file for machine-readable documentation.

Usage

```
borg_export(diagnosis, data, file, comparison = NULL, cv = NULL)
```

Arguments

`diagnosis`	A `BorgDiagnosis` object.
`data`	The data frame that was analyzed.
`file`	Character. Output file path. Extension determines format (.yaml/.yml for YAML, .json for JSON).
`comparison`	Optional. A `borg_comparison` object.
`cv`	Optional. A `borg_cv` object.

Value

Invisibly returns the certificate object.

See Also

borg_certificate for creating certificates.

Examples

```
spatial_data <- data.frame(
  x = runif(100), y = runif(100), response = rnorm(100)
)
diagnosis <- borg_diagnose(spatial_data, coords = c("x", "y"), target = "response")
borg_export(diagnosis, spatial_data, file.path(tempdir(), "validation.yaml"))
borg_export(diagnosis, spatial_data, file.path(tempdir(), "validation.json"))
```

---

borg_extract *Extract Raster Values at Point Locations for BORG*

Description

Convenience function that extracts environmental raster values at species occurrence (or other point) locations and returns a BORG-ready data frame with coordinates attached.

Usage

```
borg_extract(
  raster,
  points,
  coords = NULL,
  na.rm = TRUE,
  coord_names = c("x", "y")
)
```

Arguments

raster	A terra::SpatRaster of environmental layers (predictors).
points	Occurrence locations. One of: • A terra::SpatVector of point geometries • An sf object with point geometries • A data.frame with coordinate columns (requires coords) • A two-column matrix of coordinates (x/lon, y/lat)
coords	Character vector of length 2 giving coordinate column names. Required when points is a data.frame. Ignored for sf/SpatVector.
na.rm	Logical. If TRUE (default), rows with any NA in extracted raster values are removed. Set to FALSE to keep all rows.
coord_names	Character vector of length 2. Column names for the coordinates in the output. Default: c("x", "y").

Details

This bridges the standard SDM workflow (rasters + points) with BORG's data.frame interface. The returned data frame can be passed directly to `borg()`, `borg_cv()`, or `borg_diagnose()`.

Requires the **terra** package. If `points` is an `sf` object, the **sf** package is also required.

The function performs the equivalent of:

```
env_data <- terra::extract(raster, points, ID = FALSE)
coords <- terra::crds(points)
cbind(coords, env_data)
```

but handles edge cases (NA removal, coordinate injection, CRS checks) and attaches metadata so downstream BORG functions can auto-detect spatial structure.

Value

A data frame with columns for coordinates (named by `coord_names`), one column per raster layer (environmental variables), and any additional columns from the input `points` data. The returned data frame has a `"borg_coords"` attribute storing the coordinate column names, so that `borg()` can auto-detect them.

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  # Create example raster and points
  r <- terra::rast(nrows = 50, ncols = 50,
                   xmin = 0, xmax = 100, ymin = 0, ymax = 100)
  terra::values(r) <- runif(terra::ncell(r))
  names(r) <- "bio1"

  pts <- terra::vect(
    cbind(x = runif(100, 0, 100), y = runif(100, 0, 100)),
    crs = terra::crs(r)
  )

  # Extract and run BORG
  d <- borg_extract(r, pts)
  result <- borg(d, coords = c("x", "y"), target = "bio1")
}
```

---

borg_extrapolation *Detect Model Extrapolation*

Description

Checks whether prediction locations fall outside the environmental envelope of training data. Uses per-variable range checks and multivariate Mahalanobis distance.

Usage

```
borg_extrapolation(train, new, predictors = NULL, coords = NULL)
```

Arguments

train	Data frame of training data.
new	Data frame of prediction data (same predictor columns).
predictors	Character vector. Predictor column names. If NULL, uses all shared numeric columns.
coords	Character vector of length 2. Coordinate columns in new for spatial mapping. Optional.

Value

A data frame with class "borg_extrapolation" containing:

mahal_dist Mahalanobis distance to training centroid

n_vars_outside Number of variables outside training range

vars_outside Comma-separated names of out-of-range variables

extrapolating Logical. TRUE if any variable is out of range

x, y Coordinates (if provided)

Examples

```
set.seed(42)
train <- data.frame(a = rnorm(80), b = rnorm(80))
new <- data.frame(a = c(rnorm(10), rnorm(10, mean = 5)),
                  b = c(rnorm(10), rnorm(10, mean = 5)))
ext <- borg_extrapolation(train, new)
table(ext$extrapolating)
```

---

borg_fairness *Detect Performance Disparities Across Subgroups Under Blocked CV*

Description

Compares model performance across spatial, temporal, or categorical subgroups under both random and blocked CV. Leakage can mask disparate performance — a model that looks uniformly good under random CV may show large subgroup-level gaps when evaluated honestly.

Usage

```
borg_fairness(
  data,
  model,
  target,
  group = NULL,
  coords = NULL,
  n_groups = 4L,
  folds_blocked = NULL,
  folds_random = NULL,
  metric = c("rmse", "mae", "rsq"),
  v = 5L,
  ...
)
```

Arguments

data	A data frame with predictors and response.
model	A fitted model object with a predict method.
target	Character. Response variable name.
group	Character. Column name defining subgroups (e.g., region, time period, site type). If NULL and coords is provided, subgroups are derived from spatial clusters.
coords	Character vector of length 2. Coordinate column names. Used for spatial subgroup clustering if group is NULL.
n_groups	Integer. Number of spatial clusters when auto-grouping. Default: 4.
folds_blocked	A list of train/test folds from borg_cv. If NULL, generated automatically.
folds_random	A list of random CV folds. If NULL, generated automatically.
metric	Character. Metric to compute: "rmse", "mae", or "rsq". Default: "rmse".
v	Integer. Number of folds if generating automatically. Default: 5.
...	Additional arguments passed to predict.

Details

This function evaluates whether leakage disproportionately affects certain data subgroups. A model may have uniform performance under random CV but show large disparities when spatial or temporal independence is enforced.

For example, in ecological modelling, a random-CV RMSE of 0.5 across all regions may hide the fact that spatially blocked CV yields RMSE of 0.3 in data-rich regions but 1.2 in undersampled regions.

Value

A list with class "borg_fairness" containing:

subgroup_metrics Data frame with columns: group, metric_random, metric_blocked, n_obs, disparity (blocked - random), and relative_disparity (percent change).

overall List with overall random and blocked metrics.

max_disparity The largest subgroup-level performance gap between random and blocked CV.

worst_group The subgroup with the worst blocked CV performance.

hidden_by_leakage Logical. TRUE if any subgroup's blocked performance is substantially worse than random CV suggested.

metric The metric used.

See Also

borg_cv, borg_compare_cv

Examples

```
set.seed(42)
d <- data.frame(x = runif(200), y = runif(200))
d$z <- sin(d$x * 3) + rnorm(200, sd = 0.3)
model <- lm(z ~ x + y, data = d)
fair <- borg_fairness(d, model, target = "z", coords = c("x", "y"))
fair
```

---

borg_fold_performance *Evaluate Per-Fold Model Performance*

Description

Fits a model on each fold's training set and evaluates on its test set, returning per-fold metrics with spatial centroids for geographic performance mapping.

Usage

```
borg_fold_performance(
  data,
  folds,
  formula,
  coords = NULL,
 metric = c("rmse", "mae", "rsq", "auc", "tss", "kappa", "sensitivity", "specificity",
    "accuracy"),
  fit_fun = stats::lm,
  parallel = FALSE
)
```

Arguments

`data`	Data frame with predictor and target columns.
`folds`	A `borg_cv` object, or a list of fold lists (each with `$train` and `$test` index vectors).
`formula`	A model formula (e.g. `y ~ x1 + x2`).
`coords`	Character vector of length 2. Coordinate column names for computing fold centroids. Optional.
`metric`	Character. Performance metric: `"rmse"` (default), `"mae"`, or `"rsq"`.
`fit_fun`	Function to fit a model. Default: `lm`. Must accept `formula` and `data` arguments and return an object with a `predict()` method.
`parallel`	Logical. If `TRUE` and the **future.apply** package is installed, fold evaluation runs in parallel. Default: `FALSE`.

Value

A data frame with columns:

fold Fold index

metric Metric name

value Metric value

n_train Training set size

n_test Test set size

centroid_x, centroid_y Spatial centroid of test set (if coords provided)

Has class `"borg_fold_perf"` with an `autoplot()` method.

Examples

```
set.seed(42)
d <- data.frame(
  x = runif(200, 0, 100), y = runif(200, 0, 100),
  z = rnorm(200)
)
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
perf <- borg_fold_performance(d, cv, z ~ x + y, coords = c("x", "y"))
perf
```

borg_fold_similarity *Per-Fold Environmental Similarity (MESS)*

Description

Computes the Multivariate Environmental Similarity Surface (MESS) for each CV fold, showing how representative each fold's test set is relative to the training set in environmental space.

Usage

```
borg_fold_similarity(data, folds, predictors = NULL)
```

Arguments

`data`	Data frame.
`folds`	A `borg_cv` object or fold list.
`predictors`	Character vector. If `NULL`, all numeric columns used.

Value

A data frame with per-fold similarity metrics.

References

Elith, J., Kearney, M., & Phillips, S. (2010). The art of modelling range-shifting species. *Methods in Ecology and Evolution*, 1(4), 330-342. doi:10.1111/j.2041210X.2010.00036.x

borg_forward_selection *Forward Feature Selection with Spatial/Blocked CV*

Description

Selects variables using blocked cross-validation instead of random CV, avoiding overfitting to spatial/temporal structure. At each step, adds the variable that most improves the CV metric.

Usage

```
borg_forward_selection(
  data,
  target,
  predictors = NULL,
  folds = NULL,
  coords = NULL,
  time = NULL,
```

```
  groups = NULL,
  metric = c("rmse", "mae", "rsq"),
  fit_fun = stats::lm,
  min_vars = 1L,
  verbose = FALSE
)
```

Arguments

data	Data frame.
target	Character. Response variable name.
predictors	Character vector. Candidate predictor names. If NULL, uses all numeric columns except target and coords.
folds	A borg_cv object or fold list. If NULL, generated automatically via borg_cv().
coords	Character vector of coordinate columns (for auto CV).
time	Character. Time column (for auto CV).
groups	Character. Group column (for auto CV).
metric	Character. "rmse" (default), "mae", or "rsq".
fit_fun	Function. Model fitting function. Default: lm.
min_vars	Integer. Minimum variables to include. Default: 1.
verbose	Logical. Default: FALSE.

Details

Similar to CAST::ffs() but uses BORG's CV infrastructure and supports any dependency structure (spatial, temporal, grouped).

Value

A list with class "borg_ffs" containing:

selected Character vector of selected variable names (in order)

history Data frame: step, variable_added, metric_value, n_vars

best_metric Best CV metric achieved

all_vars All candidate variables

Examples

```
set.seed(42)
d <- data.frame(
  x = runif(100), y = runif(100),
  important = rnorm(100), noise1 = rnorm(100), noise2 = rnorm(100)
)
d$z <- d$important * 2 + rnorm(100, sd = 0.5)
ffs <- borg_forward_selection(d, target = "z",
                              predictors = c("important", "noise1", "noise2"),
```

```
                                  coords = c("x", "y"))
ffs$selected
```

`borg_geodist` *Distance Distribution Diagnostics*

Description

Computes nearest-neighbor distance distributions in geographic space and/or feature space between training data, CV test sets, and optional prediction locations. Diagnostic for whether CV folds are representative of the prediction task.

Usage

```
borg_geodist(
  data,
  folds,
  prediction_points = NULL,
  coords = NULL,
  predictors = NULL,
  type = c("both", "geo", "feature")
)
```

Arguments

`data`	Data frame of training/modelling data.
`folds`	A `borg_cv` object or fold list.
`prediction_points`	Optional data frame of prediction locations.
`coords`	Character vector of length 2. Coordinate columns for geographic distance. If `NULL`, only feature-space distances are computed.
`predictors`	Character vector. Feature columns for feature-space distance. If `NULL`, uses all shared numeric columns excluding coords.
`type`	Character. `"geo"` (geographic only), `"feature"` (feature space only), or `"both"` (default).

Details

If the CV test-to-train distance distribution differs strongly from the prediction-to-train distribution, the CV does not mimic the real prediction scenario and performance estimates may be misleading.

Value

A list with class `"borg_geodist"` containing:

cv_distances Data frame of NN distances: test-to-train per fold

prediction_distances NN distances: prediction-to-train (if provided)

sample_distances NN distances: within training data (reference)

ks_statistic KS test statistic comparing CV vs prediction distances

Has an `autoplot()` method showing overlaid density curves.

References

Meyer, H., & Pebesma, E. (2022). Machine learning-based global maps of ecological variables and the challenge of assessing them. *Nature Communications*, 13, 2208. doi:10.1038/s41467022298389

Examples

```
set.seed(42)
d <- data.frame(x = runif(100, 0, 50), y = runif(100, 0, 50),
                a = rnorm(100), z = rnorm(100))
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
pred <- data.frame(x = runif(50, 0, 100), y = runif(50, 0, 100),
                   a = rnorm(50))
gd <- borg_geodist(d, cv, prediction_points = pred, coords = c("x", "y"))
gd
```

---

borg_global_validation

Global (Pooled) Cross-Validation Metrics

Description

Pools all held-out predictions across folds and computes a single performance metric, avoiding bias from unequal fold sizes.

Usage

```
borg_global_validation(
  data,
  folds,
  formula,
  metric = c("rmse", "mae", "rsq"),
  fit_fun = stats::lm
)
```

Arguments

data	Data frame.
folds	A borg_cv object or fold list.
formula	Model formula.
metric	Character. "rmse", "mae", or "rsq".
fit_fun	Function. Default: lm.

Value

A list with class "borg_global_validation" containing the pooled metric and per-fold metrics for comparison.

---

borg_group_vfold_cv *BORG-Guarded group_vfold_cv*

Description

A guarded version of rsample::group_vfold_cv() that validates group-based CV is appropriate for the data structure.

Usage

```
borg_group_vfold_cv(
  data,
  group,
  v = NULL,
  balance = c("groups", "observations"),
  coords = NULL,
  time = NULL,
  target = NULL,
  ...
)
```

Arguments

data	A data frame.
group	Character. Column name for grouping.
v	Integer. Number of folds. Default: number of groups.
balance	Character. How to balance folds: "groups" or "observations".
coords	Character vector. Coordinate columns for spatial check.
time	Character. Time column for temporal check.
target	Character. Target variable for dependency detection.
...	Additional arguments passed to rsample::group_vfold_cv().

Value

An `rset` object from rsample.

Examples

```
if (requireNamespace("rsample", quietly = TRUE)) {
  # Clustered data - group CV is appropriate
  data <- data.frame(
    site = rep(1:20, each = 5),
    x = rnorm(100),
    y = rnorm(100)
  )
  folds <- borg_group_vfold_cv(data, group = "site", v = 5)
}
```

---

borg_importance *Block-Permutation Variable Importance*

Description

Computes permutation importance that respects spatial structure. Instead of permuting individual rows (which breaks spatial autocorrelation and inflates importance), permutes values within spatial blocks.

Usage

```
borg_importance(
  model,
  data,
  target,
  coords = NULL,
  predictors = NULL,
  n_blocks = 10,
  n_rep = 10,
  metric = c("rmse", "mae"),
  seed = 42
)
```

Arguments

`model`	A fitted model with a `predict()` method.
`data`	Data frame with predictor columns.
`target`	Character. Target variable name.
`coords`	Character vector of length 2. Coordinate column names. If `NULL`, falls back to standard row permutation.

predictors	Character vector. Variables to assess. If NULL, uses all columns except target and coords.
n_blocks	Integer. Number of spatial blocks for permutation. Default: 10.
n_rep	Integer. Number of permutation repeats per variable. Default: 10.
metric	Character. "rmse" (default) or "mae".
seed	Integer. Random seed. Default: 42.

Value

A data frame with class "borg_importance" containing:

variable Predictor name

importance Mean increase in error after permutation

importance_sd SD across repeats

rank Importance rank (1 = most important)

Has an autoplot() method.

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100),
                a = rnorm(100), b = rnorm(100))
d$z <- d$a * 2 + rnorm(100, sd = 0.5)
model <- lm(z ~ a + b, data = d)
imp <- borg_importance(model, d, target = "z", coords = c("x", "y"))
imp
```

---

borg_initial_split *BORG-Guarded initial_split*

Description

A guarded version of rsample::initial_split() that checks for temporal ordering when time structure is specified.

Usage

```
borg_initial_split(
  data,
  prop = 3/4,
  strata = NULL,
  time = NULL,
  coords = NULL,
  groups = NULL,
  target = NULL,
  ...
)
```

Arguments

data	A data frame.
prop	Numeric. Proportion of data for training. Default: 0.75.
strata	Character. Column name for stratification.
time	Character. Time column - if provided, ensures chronological split.
coords	Character vector. Coordinate columns for spatial check.
groups	Character. Group column for clustered check.
target	Character. Target variable.
...	Additional arguments passed to `rsample::initial_split()`.

Details

When `time` is specified, this function ensures the split respects temporal ordering (training data comes before test data). For spatial data, it warns if random splitting may cause issues.

Value

An `rsplit` object.

Examples

```
if (requireNamespace("rsample", quietly = TRUE)) {
  # Temporal data - ensures chronological split
  ts_data <- data.frame(
    date = seq(as.Date("2020-01-01"), by = "day", length.out = 100),
    value = cumsum(rnorm(100))
  )
  split <- borg_initial_split(ts_data, prop = 0.8, time = "date")
}
```

---

borg_inspect *Inspect R Objects for Evaluation Risks*

Description

`borg_inspect()` examines R objects for signals of information reuse that would invalidate model evaluation. It returns a structured assessment of detected risks.

Usage

```
borg_inspect(
  object,
  train_idx = NULL,
  test_idx = NULL,
  data = NULL,
  target = NULL,
  coords = NULL,
  ...
)
```

Arguments

`object`	An R object to inspect. Supported types include: • Preprocessing: `preProcess`, `recipe`, `prcomp` • CV objects: `trainControl`, `rsplit`, `vfold_cv` • Model objects: `train`, `lm`, `glm` • Data frames with train/test split information
`train_idx`	Integer vector of training row indices. Required for data-level inspection.
`test_idx`	Integer vector of test row indices. Required for data-level inspection.
`data`	Optional data frame. Required when inspecting preprocessing objects to compare parameters against train-only statistics.
`target`	Optional name of the target/outcome column. If provided, checks for target leakage (features highly correlated with target).
`coords`	Optional character vector of coordinate column names. If provided, checks spatial separation between train and test.
`...`	Additional arguments passed to type-specific inspectors.

Details

`borg_inspect()` dispatches to type-specific inspectors based on the class of the input object. Each inspector looks for specific leakage patterns:

Preprocessing objects Checks if parameters (mean, sd, loadings) were computed on data that includes test indices

CV objects Validates that train/test indices do not overlap and that grouping structure is respected

Feature engineering Checks if encodings, embeddings, or derived features used test data during computation

Value

A `BorgRisk` object containing:

risks List of detected risk objects

n_hard Count of hard violations

n_soft Count of soft inflation warnings

is_valid TRUE if no hard violations detected

See Also

borg_validate for complete workflow validation, borg for automated enforcement during evaluation.

Examples

```
# Inspect a preprocessing object
data(mtcars)
train_idx <- 1:25
test_idx <- 26:32

# BAD: preProcess fitted on full data (will detect leak)
pp_bad <- scale(mtcars[, -1])

# GOOD: preProcess fitted on train only
pp_good <- scale(mtcars[train_idx, -1])
```

---

borg_leaflet *Interactive Leaflet Map of Spatial CV Folds*

Description

Creates an interactive leaflet map showing train/test point assignments across CV folds. Requires geographic coordinates (lat/lon).

Usage

```
borg_leaflet(object, data, coords, fold = 1)
```

Arguments

object	A borg_result or borg_cv object with spatial folds.
data	Data frame with coordinate columns. Required.
coords	Character vector of length 2. Coordinate column names (longitude first, latitude second).
fold	Integer or "all". Which fold to display. Default: 1. Use "all" for a layer-group selector across all folds.

Details

Requires the **leaflet** package. Coordinates must be in WGS84 (longitude/latitude). Points are color-coded: blue = train, red = test. Click points for popups with index and fold assignment.

Value

A leaflet htmlwidget.

Examples

```
if (requireNamespace("leaflet", quietly = TRUE)) {
  set.seed(42)
  d <- data.frame(
    lon = runif(100, 10, 20),
    lat = runif(100, 45, 55),
    z = rnorm(100)
  )
  result <- borg(d, coords = c("lon", "lat"), target = "z")
  borg_leaflet(result, data = d, coords = c("lon", "lat"))
}
```

---

borg_literature_check *Scan Methods Text for Common Leakage Patterns*

Description

Analyzes a methods section (as text) for descriptions of evaluation practices that commonly lead to data leakage. Useful for reviewing papers, teaching, or auditing your own methods description.

Usage

```
borg_literature_check(
  text,
  context = c("auto", "spatial", "temporal", "general"),
  strict = FALSE
)
```

Arguments

`text`	Character string. The methods section text to analyze. Can be a single string or character vector (lines).
`context`	Character. What kind of data the methods describe. One of `"spatial"`, `"temporal"`, `"general"`, or `"auto"` (default, tries to detect from text).
`strict`	Logical. If TRUE, flags borderline practices too. Default: FALSE.

Details

The scanner looks for textual patterns that describe evaluation practices known to cause leakage. It does NOT parse actual code or data — it analyzes the *description* of methods.

Patterns checked:

Random CV on spatial data Any mention of random/stratified k-fold CV when spatial coordinates are mentioned.

Preprocessing before splitting Normalization, PCA, or feature selection described before train/test splitting.

No spatial blocking Spatial data evaluated without spatial CV or buffer zones.

Feature selection on full data Variable selection or importance computed before CV.

Temporal look-ahead Random splits on time series or panel data.

No embargo Temporal CV without gap/embargo between train and test.

Value

A list with class `"borg_lit_check"` containing:

flags Data frame with columns: pattern, severity (`"high"`, `"medium"`, `"low"`), matched_text, explanation, and recommendation.

n_flags Total number of issues found.

detected_context The data context detected or provided.

summary One-line summary.

See Also

`borg_explain_risk`, `borg_report`

Examples

```
methods_text <- "We used 10-fold cross-validation to evaluate
  species distribution models. Predictors were normalized and
  reduced via PCA before model fitting. Occurrence records
  were collected across 50 sites in Europe."

result <- borg_literature_check(methods_text)
result
```

---

borg_local_moran *Local Moran's I for Residuals*

Description

Computes local Moran's I statistics to identify spatial clusters of high/low residuals (hotspots and coldspots).

Usage

```
borg_local_moran(residuals, x, y, k = 8L)
```

Arguments

residuals	Numeric vector of model residuals.
x	Numeric. X-coordinates.
y	Numeric. Y-coordinates.
k	Integer. Number of nearest neighbors for spatial weights. Default: 8.

Value

A data frame with columns: x, y, local_i, p_value, cluster_type.

---

borg_metrics *Available Classification Metrics*

Description

Returns the valid classification metric names.

Usage

```
borg_metrics()
```

Value

Character vector of available classification metrics.

Examples

```
borg_metrics()
```

---

borg_multiscale *Multi-Scale Performance Assessment*

Description

Evaluates model accuracy at increasing spatial aggregation scales. Predictions and observations are averaged within grid cells of increasing size, and performance is computed at each scale. Reveals scale-dependent error patterns.

Usage

```
borg_multiscale(
  data,
  predictions,
  target,
  coords,
  scales = NULL,
  metric = c("rmse", "mae", "rsq")
)
```

Arguments

data	Data frame with coordinates and target variable.
predictions	Numeric vector of predicted values (same length as nrow(data)).
target	Character. Target variable name.
coords	Character vector of length 2. Coordinate columns.
scales	Numeric vector. Grid cell sizes to evaluate. If NULL, auto-generated from 5 percent to 50 percent of extent in 8 steps.
metric	Character. "rmse" (default), "mae", or "rsq".

Value

A data frame with class "borg_multiscale" containing:

scale Grid cell size

metric_value Performance at this scale

n_cells Number of occupied grid cells

Has an autoplot() method.

References

Riemann, R., Wilson, B. T., Lister, A., & Parks, S. (2010). An effective assessment protocol for continuous geospatial datasets of forest characteristics using USFS Forest Inventory and Analysis (FIA) data. *Remote Sensing of Environment*, 114(10), 2337-2352. doi:10.1016/j.rse.2010.05.010

Examples

```
set.seed(42)
d <- data.frame(x = runif(200, 0, 100), y = runif(200, 0, 100),
                z = rnorm(200))
preds <- d$z + rnorm(200, sd = 0.5)
ms <- borg_multiscale(d, preds, target = "z", coords = c("x", "y"))
ms
```

`borg_null_test` *Null Model Significance Test*

Description

Tests whether model performance is significantly better than random expectation by permuting the response variable and re-evaluating with the same CV scheme. Computes z-scores and p-values.

Usage

```
borg_null_test(
  data,
  folds,
  formula,
  metric = c("rmse", "mae", "rsq"),
  fit_fun = stats::lm,
  n_null = 99L,
  seed = 42L,
  verbose = FALSE
)
```

Arguments

`data`	Data frame.
`folds`	A `borg_cv` object or fold list.
`formula`	Model formula.
`metric`	Character. `"rmse"` (default), `"mae"`, or `"rsq"`.
`fit_fun`	Function. Default: `lm`.
`n_null`	Integer. Number of null permutations. Default: 99.
`seed`	Integer. Random seed. Default: 42.
`verbose`	Logical. Default: FALSE.

Value

A list with class `"borg_null_test"` containing:

empirical Empirical CV metric value

null_distribution Numeric vector of null metric values

z_score Z-score of empirical vs null

p_value One-sided p-value

significant Logical. Whether model is significantly better

Has an `autoplot()` method showing null distribution with empirical value.

References

Kass, J. M., Muscarella, R., Galante, P. J., Bohl, C. L., Pinilla-Buitrago, G. E., Boria, R. A., Soley-Guardia, M., & Anderson, R. P. (2021). ENMeval 2.0: Redesigned for customizable and reproducible modeling of species' niches and distributions. *Methods in Ecology and Evolution*, 12(9), 1602-1608. doi:10.1111/2041210X.13628

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
d$z <- d$x * 3 + rnorm(100)
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
nt <- borg_null_test(d, cv, z ~ x + y, n_null = 19)
nt
```

---

borg_options *Get Current BORG Options*

Description

Returns the current state of BORG configuration options.

Usage

```
borg_options()
```

Value

A named list of current BORG options.

Examples

```
borg_options()
```

---

borg_pipeline *Validate an Entire Modeling Pipeline*

Description

Walks a tidymodels `workflow()` or `caret::train()` object and validates every step — preprocessing, feature selection, tuning, and model fitting — for information leakage.

Usage

```
borg_pipeline(pipeline, train_idx, test_idx, data = NULL, ...)
```

Arguments

`pipeline`	A modeling pipeline object. Supported types: • A tidymodels `workflow` object (fitted or unfitted) • A `caret::train` object • A list with named components (recipe, model, tune_results, etc.)
`train_idx`	Integer vector of training row indices.
`test_idx`	Integer vector of test row indices.
`data`	Optional data frame. Required for parameter-level checks.
`...`	Additional arguments passed to inspectors.

Details

`borg_pipeline()` decomposes a pipeline into stages and inspects each:

1. **Preprocessing**: Recipe steps, preProcess, PCA, scaling
2. **Feature selection**: Variable importance, filtering
3. **Hyperparameter tuning**: Inner CV resamples
4. **Model fitting**: Training data scope, row counts
5. **Post-processing**: Threshold optimization, calibration

Each stage gets its own BorgRisk assessment. The overall result aggregates all risks across stages.

Value

An object of class `"borg_pipeline"` containing:

stages Named list of per-stage BorgRisk results

overall Aggregated BorgRisk for the full pipeline

n_stages Number of stages inspected

leaking_stages Character vector of stage names with hard violations

See Also

`borg_validate`, `borg_inspect`

Examples

```
if (requireNamespace("caret", quietly = TRUE)) {
  ctrl <- caret::trainControl(method = "cv", number = 5)
  model <- caret::train(mpg ~ ., data = mtcars[1:25, ], method = "lm",
                        trControl = ctrl, preProcess = c("center", "scale"))
  result <- borg_pipeline(model, train_idx = 1:25, test_idx = 26:32,
                          data = mtcars)
  print(result)
}
```

borg_power *Estimate Statistical Power After Blocking*

Description

Computes how much statistical power is lost when switching from random to blocked cross-validation. Reports effective sample size, minimum detectable effect size, and whether the dataset is large enough.

Usage

```
borg_power(
  data,
  diagnosis = NULL,
  coords = NULL,
  time = NULL,
  groups = NULL,
  target = NULL,
  alpha = 0.05,
  power = 0.8,
  effect_size = NULL,
  verbose = FALSE
)
```

Arguments

data	A data frame.
diagnosis	A BorgDiagnosis object. If NULL, computed automatically from the other arguments.
coords	Character vector of length 2 for spatial coordinates.
time	Character string for the time column.
groups	Character string for the grouping column.
target	Character string for the response variable.
alpha	Significance level. Default: 0.05.
power	Target power. Default: 0.80.
effect_size	Numeric. Expected effect size (Cohen's d for continuous, OR for binary). If NULL, reports minimum detectable effect size instead.
verbose	Logical. Print progress messages. Default: FALSE.

Details

When data have spatial, temporal, or clustered dependencies, blocked CV reduces the effective sample size. This function quantifies that reduction using the design effect (DEFF):

$$n_{eff} = n/DEFF$$

The design effect is computed from:

- **Spatial**: Moran's I and the ratio of autocorrelation range to study extent (Griffith, 2005)
- **Temporal**: ACF lag-1 autocorrelation ($DEFF \approx (1+\rho)/(1-\rho)$)
- **Clustered**: ICC and mean cluster size ($DEFF = 1 + (m-1) \times ICC$)

For mixed dependencies, design effects are combined multiplicatively.

Value

An object of class `"borg_power"` containing:

n_actual Total number of observations

n_effective Effective sample size after accounting for dependencies

design_effect Variance inflation factor from dependencies

power_random Statistical power under random CV

power_blocked Statistical power under blocked CV

power_loss Absolute power loss (power_random - power_blocked)

min_detectable_effect Minimum detectable effect at target power

min_detectable_effect_random Same, under random CV (for comparison)

sufficient Logical. Is the dataset large enough at target power?

recommendation Character. Human-readable recommendation.

diagnosis The BorgDiagnosis used

See Also

borg_diagnose, borg_cv

Examples

```
# Clustered data
clustered_data <- data.frame(
  site = rep(1:20, each = 10),
  value = rep(rnorm(20, sd = 2), each = 10) + rnorm(200, sd = 0.5)
)

pw <- borg_power(clustered_data, groups = "site", target = "value")
print(pw)
```

`borg_prediction_map` *Spatial Prediction Uncertainty Map*

Description

Collects predictions from all CV folds at each test location, computes per-location mean and standard deviation, and provides a spatial uncertainty map.

Usage

```
borg_prediction_map(data, folds, formula, coords, fit_fun = stats::lm)
```

Arguments

`data`	Data frame with predictor and coordinate columns.
`folds`	A `borg_cv` object or fold list.
`formula`	Model formula.
`coords`	Character vector of length 2. Coordinate column names.
`fit_fun`	Function. Default: `lm`.

Value

A data frame with class `"borg_pred_map"` containing:

x, y Coordinates

pred_mean Mean prediction across folds where this obs was in test set

pred_sd SD of predictions (uncertainty)

actual Observed target value

residual_mean Mean residual

n_folds_tested Number of folds where this obs appeared in test set

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
pm <- borg_prediction_map(d, cv, z ~ x + y, coords = c("x", "y"))
head(pm)
```

`borg_predict_raster` *Predict onto a SpatRaster with AOA Masking*

Description

Generates spatial predictions from a fitted model onto a `terra::SpatRaster`, computes the dissimilarity index and area of applicability, and returns a multi-layer raster with prediction, DI, and AOA mask layers.

Usage

```
borg_predict_raster(
  model,
  raster,
  train_data,
  predictors = NULL,
  weights = NULL,
  threshold = NULL,
  type = "response"
)
```

Arguments

`model`	A fitted model with a `predict()` method.
`raster`	A `terra::SpatRaster` with predictor layers matching the model's training variables.
`train_data`	Data frame of training data used to fit the model.
`predictors`	Character vector. Predictor column names. If `NULL`, uses raster layer names.
`weights`	Numeric vector. Variable importance weights for DI.
`threshold`	Numeric. Manual AOA threshold. If `NULL`, computed from training data.
`type`	Character. Prediction type passed to `predict()`. Default: `"response"`.

Details

Requires the **terra** package.

Value

A `terra::SpatRaster` with three layers:

prediction Model predictions

di Dissimilarity index

aoa Area of applicability (1 = inside, 0 = outside)

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  set.seed(42)
  r <- terra::rast(nrows = 20, ncols = 20, xmin = 0, xmax = 100,
                   ymin = 0, ymax = 100, nlyrs = 2)
  terra::values(r) <- cbind(rnorm(400), rnorm(400))
  names(r) <- c("bio1", "bio2")
  train <- data.frame(bio1 = rnorm(50), bio2 = rnorm(50))
  train$y <- train$bio1 * 2 + rnorm(50, sd = 0.5)
  model <- lm(y ~ bio1 + bio2, data = train)
  result <- borg_predict_raster(model, r, train, predictors = c("bio1", "bio2"))
  names(result)
}
```

---

borg_register_hooks *Register BORG Hooks*

Description

Registers BORG validation hooks that automatically check data dependencies when using common ML framework functions. This is an experimental feature.

Usage

```
borg_register_hooks(
  frameworks = c("rsample", "caret", "mlr3"),
  action = c("error", "warn", "message")
)
```

Arguments

`frameworks`	Character vector. Which frameworks to hook into. Options: "rsample", "caret", "mlr3". Default: all available.
`action`	Character. What to do when dependencies detected: "error" (block), "warn" (warn but proceed), "message" (info only).

Details

This function uses R's trace mechanism to add BORG checks to framework functions. The hooks are session-specific and do not persist.

To remove hooks, use `borg_unregister_hooks()`.

Value

Invisible NULL. Called for side effect.

Examples

```
if (requireNamespace("rsample", quietly = TRUE)) {
  # Register hooks for rsample
  borg_register_hooks("rsample")

  # Now vfold_cv() will check for dependencies
  spatial_data <- data.frame(
    lon = runif(50), lat = runif(50), response = rnorm(50)
  )
  options(borg.check_data = spatial_data)
  options(borg.check_coords = c("lon", "lat"))

  # Remove hooks
  borg_unregister_hooks()
}
```

---

borg_repeated_cv *Repeated Blocked Cross-Validation*

Description

Repeats borg_cv multiple times with different random seeds and aggregates the results. Repeated CV provides better variance estimates and more stable performance metrics than a single CV run, particularly important for spatial/temporal blocking where fold assignment depends on the random seed.

Usage

```
borg_repeated_cv(
  data,
  repeats = 10L,
  v = 5L,
  seeds = NULL,
  aggregate = TRUE,
  ...
)
```

Arguments

data	A data frame.
repeats	Integer. Number of repetitions. Default: 10.
v	Integer. Number of folds per repetition. Default: 5.
seeds	Integer vector of length repeats, or NULL (auto-generated). Controls fold assignment for each repetition.

aggregate	Logical. If TRUE (default), returns aggregated metrics. If FALSE, returns all individual fold results.
...	Additional arguments passed to borg_cv.

Details

Each repetition calls borg_cv() with a different seed, producing different fold assignments. The diagnosis is computed once and reused across repetitions for consistency.

For aggregation with a model, use borg_fold_performance() on each repetition's folds and combine the results.

Value

A list with class "borg_repeated_cv" containing:

folds List of length repeats, each element a list of v train/test fold pairs.

repeats Number of repetitions.

v Number of folds.

strategy CV strategy used.

seeds Seeds used for each repetition.

diagnosis The BorgDiagnosis from the first run.

Has print() and autoplot() methods.

See Also

borg_cv, borg_fold_performance

Examples

```
set.seed(42)
d <- data.frame(
  x = runif(200), y = runif(200), z = rnorm(200)
)
rcv <- borg_repeated_cv(d, repeats = 3, v = 5,
                        coords = c("x", "y"), target = "z")
rcv
length(rcv$folds)  # 3 repetitions
length(rcv$folds[[1]])  # 5 folds each
```

borg_report *Generate BORG HTML Diagnostic Report*

Description

Creates a self-contained HTML report with embedded plots summarizing the full BORG analysis: diagnosis, variogram, CV folds, performance, and risk assessment.

Usage

```
borg_report(
  object,
  file = "borg_report.html",
  title = "BORG Diagnostic Report",
  open = interactive()
)
```

Arguments

object	A borg_workflow or borg_result object.
file	Character. Output file path. Default: "borg_report.html".
title	Character. Report title.
open	Logical. Open in browser after generation. Default: TRUE in interactive sessions.

Details

Plots are embedded as base64-encoded PNGs. Requires **ggplot2**. No rmarkdown or pandoc dependency.

Value

Invisible path to the generated HTML file.

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
wf <- borg_workflow(d, z ~ x + y, coords = c("x", "y"))
borg_report(wf, file = tempfile(fileext = ".html"), open = FALSE)
```

borg_rset *Convert BORG Folds to an rsample rset Object*

Description

Creates a proper `rset` object from BORG cross-validation folds, enabling direct use with `tune::tune_grid()`, `tune::fit_resamples()`, and other tidymodels infrastructure without manual conversion.

Usage

```
borg_rset(data = NULL, folds = NULL, cv_obj = NULL)
```

Arguments

`data`	A data frame. Required in all cases because `borg_cv` objects do not store the original data.
`folds`	A list of lists, each with `train` and `test` integer index vectors. Typically from `borg_cv()$folds`. Not needed when `cv_obj` is provided.
`cv_obj`	A `borg_cv` object. If provided, `folds` are extracted from it.

Details

The returned object works directly with `tune::tune_grid()`, `tune::fit_resamples()`, `rsample::assessment()`, and `rsample::analysis()`. The folds preserve BORG's spatial/temporal blocking structure.

Value

An `rset` object (inheriting from `tbl_df`) compatible with the tidymodels ecosystem. Each row has an `rsplit` column containing train/test index information and an `id` column.

See Also

`borg_cv`, `borg_trainControl`

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
rset <- borg_rset(data = d, cv_obj = cv)
class(rset)  # "borg_rset" "rset" "tbl_df" ...
```

`borg_sample_design` *Suggest Sampling Locations to Improve AOA*

Description

Identifies locations in the prediction domain where new training data would most reduce the dissimilarity index, thereby expanding the area of applicability.

Usage

```
borg_sample_design(
  train,
  prediction,
  predictors = NULL,
  coords = NULL,
  n = 10L,
  weights = NULL
)
```

Arguments

`train`	Data frame of existing training data.
`prediction`	Data frame of prediction locations (with coordinates).
`predictors`	Character vector. Predictor columns for DI computation.
`coords`	Character vector of length 2. Coordinate columns in `prediction`.
`n`	Integer. Number of suggested sampling locations. Default: 10.
`weights`	Numeric vector. Variable importance weights for DI.

Value

A data frame with class `"borg_sample_design"` containing the top-n prediction locations ranked by DI (highest first), with columns: x, y, di, rank.

Examples

```
set.seed(42)
train <- data.frame(x = runif(50, 0, 50), y = runif(50, 0, 50),
                    a = rnorm(50))
pred <- data.frame(x = runif(200, 0, 100), y = runif(200, 0, 100),
                   a = rnorm(200))
design <- borg_sample_design(train, pred, predictors = "a",
                             coords = c("x", "y"), n = 5)
design
```

`borg_shap` *Spatial SHAP Values*

Description

Approximates SHAP values using block-conditional expectations instead of naive row permutation. Standard SHAP marginalizes over all observations, which creates impossible feature combinations when data is spatially structured. Spatial SHAP marginalizes within spatial blocks, preserving realistic covariate relationships.

Usage

```
borg_shap(
  model,
  data,
  target,
  coords = NULL,
  predictors = NULL,
  n_blocks = 10,
  n_samples = 50,
  explain_idx = NULL,
  seed = 42
)
```

Arguments

`model`	A fitted model with a `predict()` method.
`data`	Data frame with predictors.
`target`	Character. Target variable name (excluded from SHAP).
`coords`	Character vector of length 2. Coordinate column names. If provided, uses spatial blocking for marginal expectations.
`predictors`	Character vector. Variables to compute SHAP for. If `NULL`, uses all numeric columns except target and coords.
`n_blocks`	Integer. Number of spatial blocks for marginal expectations. Default: 10.
`n_samples`	Integer. Number of background samples per block for marginal expectations. Default: 50.
`explain_idx`	Integer vector. Row indices to explain. If `NULL`, explains all rows (can be slow).
`seed`	Integer. Random seed. Default: 42.

Details

Algorithm: For each observation x_i and feature j:

1. Identify the spatial block containing x_i.
2. Sample background points from *other* blocks.

3. Compute marginal contribution: replace feature j with values from background points, keeping all other features fixed.
4. Average the change in prediction.

This approximates the Shapley value while respecting spatial structure.

Why not standard SHAP?: Standard kernel SHAP or marginal SHAP samples replacement values uniformly from the dataset. For spatial data, this creates combinations that never occur in reality (e.g., tropical temperature with arctic precipitation), biasing the SHAP values.

Value

A list with class `"borg_shap"` containing:

shap_values Matrix (n_explain x n_predictors) of SHAP values

baseline Mean prediction (expected value)

feature_importance Named vector of mean |SHAP| per feature

predictors Feature names

method "spatial_block" or "standard"

Has `print()` and `autoplot()` methods.

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100),
                a = rnorm(100), b = rnorm(100))
d$z <- 3 * d$a - d$b + rnorm(100, sd = 0.5)
model <- lm(z ~ a + b, data = d)
shap <- borg_shap(model, d, target = "z", coords = c("x", "y"),
                  explain_idx = 1:20)
shap
```

---

borg_simulate *Generate Synthetic Data with Known Leakage*

Description

Creates datasets with controlled spatial autocorrelation, temporal dependence, or target leakage. The true inflation magnitude is known, enabling benchmarking of leakage detection methods and quantifying how much performance metrics inflate under different CV strategies.

Usage

```
borg_simulate(
  n = 500L,
 type = c("spatial", "temporal", "target_leak", "preprocessing_leak", "combined",
    "independent"),
  n_predictors = 5L,
  signal_strength = 0.3,
  autocorrelation = 0.5,
  leak_strength = 0.9,
  grid_size = NULL,
  seed = 42L
)
```

Arguments

n	Integer. Number of observations. Default: 500.
type	Character. Type of dependency to simulate. One of "spatial", "temporal", "target_leak", "preprocessing_leak", "combined", or "independent" (control). Default: "spatial".
n_predictors	Integer. Number of predictor variables. Default: 5.
signal_strength	Numeric in [0, 1]. Fraction of variance explained by predictors. Default: 0.3.
autocorrelation	Numeric in [0, 1]. Strength of spatial or temporal autocorrelation. 0 = none, 1 = very strong. Default: 0.5.
leak_strength	Numeric in [0, 1]. For target leakage scenarios, how strongly the leaked feature correlates with the response. Default: 0.9.
grid_size	Integer. For spatial data, side length of the spatial grid (total points = n, placed on a grid of approximately this resolution). Default: NULL (auto).
seed	Integer. Random seed for reproducibility. Default: 42.

Details

Simulation types:

"spatial" Predictors and response have Gaussian spatial autocorrelation on a 2D grid. Random CV inflates R2 relative to spatial block CV.

"temporal" Predictors and response are AR(1) time series. Random CV inflates metrics relative to temporal CV.

"target_leak" One predictor is a noisy copy of the response (post-hoc information).

"preprocessing_leak" Predictors are normalized on the full dataset, not within each fold. The data itself is independent but leakage occurs through shared statistics.

"combined" Spatial autocorrelation plus one target-leaked predictor.

"independent" No dependencies. Control scenario where random and blocked CV should yield similar results.

Value

A list with class "borg_simulation" containing:

data The generated data frame with predictors, response, and (if spatial/temporal) coordinate/time columns.

true_r2 The true R-squared if the model could perfectly recover the signal (upper bound for honest evaluation).

type The dependency type used.

params List of all generation parameters.

leaked_vars Character vector of intentionally leaked variable names (for target_leak/preprocessing_leak types).

coords Character vector of coordinate column names (if spatial).

time_col Name of time column (if temporal).

See Also

borg_cv, borg_compare_cv

Examples

```
# Spatial leakage benchmark
sim <- borg_simulate(n = 300, type = "spatial", autocorrelation = 0.7)
str(sim$data)
sim$true_r2

# Compare random vs spatial CV

cv_spatial <- borg_cv(sim$data, coords = sim$coords, target = "y")
```

---

borg_spatial_cv *Spatial Block Cross-Validation (rsample-compatible)*

Description

Creates spatial block CV folds that plug directly into tune::tune_grid() and tune::fit_resamples().

Usage

```
borg_spatial_cv(
  data,
  coords,
  target = NULL,
  v = 5,
  buffer = NULL,
  repeats = 1L,
  ...
)
```

Arguments

data	A data frame, sf object, or terra::SpatVector.
coords	Character vector of length 2. Coordinate column names.
target	Character. Target variable for autocorrelation diagnosis.
v	Integer. Number of folds. Default: 5.
buffer	Numeric. Optional spatial buffer distance for exclusion.
repeats	Integer. Number of repeated fold sets. Default: 1.
...	Additional arguments passed to borg_cv().

Value

A borg_rset object (subclass of rset) compatible with tidymodels.

Examples

```
if (requireNamespace("rsample", quietly = TRUE)) {
  set.seed(42)
  d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
  folds <- borg_spatial_cv(d, coords = c("x", "y"), target = "z")
}
```

---

borg_spatial_loo *Buffered Leave-One-Out Cross-Validation*

Description

For each observation, holds it out as the test set and removes all training points within a buffer distance. This ensures spatial independence between train and test in each iteration, following the approach recommended by Roberts et al. (2017).

Usage

```
borg_spatial_loo(
  data,
  coords,
  buffer = NULL,
  target = NULL,
  diagnosis = NULL,
  max_iter = NULL,
  seed = 42L,
  verbose = FALSE
)
```

Arguments

data	A data frame.
coords	Character vector of length 2. Coordinate column names.
buffer	Numeric. Exclusion buffer distance (in coordinate units). Training points within this distance of the test point are removed. If NULL, uses the estimated autocorrelation range from borg_diagnose().
target	Character. Response variable name (used for diagnosis if buffer is NULL).
diagnosis	A BorgDiagnosis object. If NULL and buffer is NULL, runs borg_diagnose() to estimate the autocorrelation range.
max_iter	Integer or NULL. Maximum number of LOO iterations. If NULL, uses all n observations. For large datasets, set this to subsample. Default: NULL.
seed	Integer. Random seed for subsampling. Default: 42.
verbose	Logical. Print progress. Default: FALSE.

Value

A list with class "borg_spatial_loo" containing:

folds List of length n (or max_iter), each with train and test integer index vectors.

buffer Buffer distance used.

n_excluded Integer vector: number of training points excluded per iteration.

effective_train_size Integer vector: training set size per iteration after exclusion.

References

Roberts, D.R., et al. (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, 40(8), 913-929. doi:10.1111/ecog.02881

See Also

borg_cv, borg_thin

Examples

```
set.seed(42)
d <- data.frame(x = runif(50), y = runif(50), z = rnorm(50))
loo <- borg_spatial_loo(d, coords = c("x", "y"), buffer = 0.2)
length(loo$folds)  # 50 iterations
mean(loo$n_excluded)  # avg points excluded per iteration
```

`borg_stability` *Analyze CV Fold Stability Across Repeats*

Description

Quantifies how much CV results and fold assignments change across repeated runs. Uses a `borg_cv` object with `repeats > 1`.

Usage

```
borg_stability(
  object,
  data,
  formula,
  metric = c("rmse", "mae", "rsq"),
  fit_fun = stats::lm
)
```

Arguments

`object`	A `borg_cv` object with `repeats > 1` (i.e., `object$all_folds` is not NULL).
`data`	Data frame for computing per-repeat performance.
`formula`	Model formula.
`metric`	Character. `"rmse"` (default), `"mae"`, or `"rsq"`.
`fit_fun`	Function. Model fitting function. Default: `lm`.

Value

A list with class `"borg_stability"` containing:

metric_stability Data frame: repeat, mean_metric, sd_metric

metric_cv Coefficient of variation of mean metric across repeats

assignment_stability Mean Jaccard similarity of test sets between repeats

summary Character assessment: "stable", "moderate", or "unstable"

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
cv <- borg_cv(d, coords = c("x", "y"), target = "z", repeats = 5)
stab <- borg_stability(cv, d, z ~ x + y)
stab
```

borg_stability_map *Prediction Stability Map*

Description

Creates a continuous spatial map of prediction variance across cross-validation folds. Unlike `borg_stability()` which returns summary statistics, this function produces per-location stability scores showing where predictions are consistent versus volatile across different train/test partitions.

Usage

```
borg_stability_map(
  model,
  data,
  new = NULL,
  target,
  coords,
  formula = NULL,
  fit_fun = NULL,
  folds = NULL,
  v = 10,
  seed = 42
)
```

Arguments

`model`	A fitted model with a `predict()` method, or a model-fitting function.
`data`	Data frame with predictors, target, and coordinates.
`new`	Data frame of prediction locations. If `NULL`, uses `data`.
`target`	Character. Target variable name.
`coords`	Character vector of length 2. Coordinate column names.
`formula`	Model formula. Required if `model` is a function.
`fit_fun`	Function. Model fitting function. If `NULL`, attempts to refit from model's call.
`folds`	A `borg_cv` or `borg_disc_cv` object, or a fold list. If `NULL`, generates spatial CV folds automatically.
`v`	Integer. Number of folds if generating automatically. Default: 10.
`seed`	Integer. Random seed. Default: 42.

Details

How it works: For each CV fold:

1. Refit the model on the training set.
2. Predict on all locations (not just test set).
3. Store per-location predictions.

Then compute per-location statistics across folds. Locations with high `pred_sd` are unstable — the prediction changes substantially depending on which data is included in training.

Interpretation: High instability at a location can indicate:

- The location is in a data-sparse region (near AOA boundary)
- The model is overfitting to nearby training points
- The underlying relationship changes spatially (non-stationarity)

Value

A data frame with class `"borg_stability_map"` containing:

pred_mean Mean prediction across folds

pred_sd Standard deviation of predictions across folds

pred_cv Coefficient of variation (|sd/mean|)

pred_range Range of predictions (max - min)

n_folds_predicted Number of folds where this point was in the test set (or could be predicted)

x, y Coordinates

Has `print()` and `autoplot()` methods.

Examples

```
set.seed(42)
d <- data.frame(x = runif(100, 0, 50), y = runif(100, 0, 50))
d$z <- sin(d$x / 5) + rnorm(100, sd = 0.5)
model <- lm(z ~ x + y, data = d)
sm <- borg_stability_map(model, d, target = "z", coords = c("x", "y"),
                         v = 5)
sm
```

---

borg_temporal_cv *Temporal Block Cross-Validation (rsample-compatible)*

Description

Creates temporal block CV folds that plug directly into `tune::tune_grid()` and `tune::fit_resamples()`.

Usage

```
borg_temporal_cv(data, time, target = NULL, v = 5, embargo = NULL, ...)
```

Arguments

data	A data frame.
time	Character. Time column name.
target	Character. Target variable.
v	Integer. Number of folds. Default: 5.
embargo	Numeric. Time gap between train and test sets.
...	Additional arguments passed to borg_cv().

Value

A borg_rset object (subclass of rset) compatible with tidymodels.

Examples

```
if (requireNamespace("rsample", quietly = TRUE)) {
  d <- data.frame(time = 1:100, x = rnorm(100), y = cumsum(rnorm(100)))
  folds <- borg_temporal_cv(d, time = "time", target = "y")
}
```

---

borg_thin *Spatially Thin Occurrence Data*

Description

Reduces spatial clustering by ensuring a minimum distance between all retained observations. Uses iterative nearest-neighbor removal: at each step, the point with the smallest nearest-neighbor distance is removed until all pairwise distances exceed min_dist.

Usage

```
borg_thin(data, coords = NULL, min_dist = NULL, verbose = FALSE)
```

Arguments

data	A data frame, sf object, or terra::SpatVector.
coords	Character vector of length 2. Coordinate column names. Required for data frames; ignored for sf/SpatVector.
min_dist	Numeric. Minimum distance between retained points. Units match the coordinate system (degrees for lat/lon, meters for projected CRS). If NULL, uses 10 percent of the nearest-neighbor interquartile range as a default.
verbose	Logical. Print thinning progress. Default: FALSE.

Details

Spatial thinning is standard practice in species distribution modelling to reduce sampling bias. Clustered occurrences inflate apparent model performance and bias spatial CV fold sizes.

For geographic coordinates (lat/lon), distances are computed using the Haversine formula (meters). For projected coordinates, Euclidean distance is used.

Value

The input data filtered to retained rows. Has attribute `"thinned_idx"` with the original row indices of kept observations, and `"n_removed"` with the count of removed points.

Examples

```
# Thin clustered points to 5-unit minimum spacing
set.seed(42)
d <- data.frame(
  x = c(rnorm(50, 0, 1), rnorm(50, 10, 1)),
  y = c(rnorm(50, 0, 1), rnorm(50, 10, 1)),
  species = "A"
)
d_thin <- borg_thin(d, coords = c("x", "y"), min_dist = 1)
nrow(d_thin)  # fewer rows
```

---

borg_to_biomod2 *Convert BORG Folds to biomod2 Format*

Description

Converts BORG CV folds to the data split table format expected by biomod2's `BIOMOD_Modeling()` function.

Usage

```
borg_to_biomod2(borg_cv)
```

Arguments

`borg_cv`	A `borg_cv` object.

Value

A matrix where each column is a CV run and each row is an observation. Values are `TRUE` (calibration) or `FALSE` (validation), matching biomod2's `DataSplitTable` format.

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
split_table <- borg_to_biomod2(cv)
dim(split_table)
```

---

borg_to_enmeval *Convert BORG Folds to ENMeval Partition Format*

Description

Converts BORG CV folds to the partition format expected by ENMeval's `ENMevaluate()` function (a vector of fold assignments).

Usage

```
borg_to_enmeval(borg_cv)
```

Arguments

borg_cv	A borg_cv object.

Value

A named list with:

occs.grp Integer vector of fold assignments for occurrence points

bg.grp Integer vector for background points (all assigned to fold 0)

Examples

```
set.seed(42)
d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
cv <- borg_cv(d, coords = c("x", "y"), target = "z")
parts <- borg_to_enmeval(cv)
table(parts$occs.grp)
```

`borg_to_mlr3` *mlr3 Resampling for BORG Cross-Validation*

Description

Creates a native `mlr3::Resampling` object from BORG cross-validation folds, enabling direct use with mlr3 benchmarking infrastructure.

Usage

```
borg_to_mlr3(data = NULL, folds = NULL, cv_obj = NULL, id = "borg")
```

Arguments

`data`	A data frame.
`folds`	A list of lists, each with `train` and `test` integer index vectors.
`cv_obj`	A `borg_cv` object. If provided, `data` and `folds` are extracted from it.
`id`	Character. Resampling identifier. Default: `"borg"`.

Details

The returned resampling is pre-instantiated — it contains fixed train/test splits that respect BORG's spatial/temporal blocking. Calling `$instantiate()` is not needed (and will not overwrite the existing splits).

Value

An `mlr3::Resampling` R6 object that can be used directly with `mlr3::resample()`, `mlr3::benchmark()`, and other mlr3 infrastructure.

See Also

`borg_cv`, `borg_rset`

Examples

```
if (requireNamespace("mlr3", quietly = TRUE)) {
  d <- data.frame(x = runif(100), y = runif(100), z = rnorm(100))
  cv <- borg_cv(d, coords = c("x", "y"), target = "z")
  resampling <- borg_to_mlr3(cv_obj = cv)
  resampling$iters  # number of folds
}
```

borg_trainControl *BORG-Guarded trainControl*

Description

A guarded version of `caret::trainControl()` that validates CV settings against data dependencies.

Usage

```
borg_trainControl(
  data,
  method = "cv",
  number = 10,
  coords = NULL,
  time = NULL,
  groups = NULL,
  target = NULL,
  allow_override = FALSE,
  ...
)
```

Arguments

`data`	A data frame. Required for dependency checking.
`method`	Character. Resampling method.
`number`	Integer. Number of folds or iterations.
`coords`	Character vector. Coordinate columns for spatial check.
`time`	Character. Time column for temporal check.
`groups`	Character. Group column for clustered check.
`target`	Character. Target variable.
`allow_override`	Logical. Allow random CV despite dependencies.
`...`	Additional arguments passed to `caret::trainControl()`.

Value

A `trainControl` object, potentially modified for blocked CV.

Examples

```
if (requireNamespace("caret", quietly = TRUE)) {
  spatial_data <- data.frame(
    lon = runif(50), lat = runif(50), response = rnorm(50)
  )
  ctrl <- borg_trainControl(
```

```
    data = spatial_data,
    method = "cv",
    number = 5,
    coords = c("lon", "lat")
  )
}
```

`borg_transferability` *Assess Geographic Transferability*

Description

Evaluates how well a model transfers across geographic regions by splitting data into spatial zones, training on each zone, and testing on all others. Quantifies performance decay with geographic distance.

Usage

```
borg_transferability(
  data,
  formula,
  coords,
  n_regions = 4L,
  metric = c("rmse", "mae", "rsq"),
  fit_fun = stats::lm
)
```

Arguments

`data`	Data frame with coordinates and predictors.
`formula`	Model formula.
`coords`	Character vector of length 2. Coordinate columns.
`n_regions`	Integer. Number of geographic regions to create. Default: 4.
`metric`	Character. Default: `"rmse"`.
`fit_fun`	Function. Default: `lm`.

Value

A list with class `"borg_transferability"` containing:

matrix Performance matrix (n_regions x n_regions): entry (i, j) = metric when training on region i, testing on region j

distance_matrix Geographic distance between region centroids

decay Data frame: distance, metric_value for plotting decay

mean_transfer Mean cross-region performance

mean_within Mean within-region performance

Examples

```
set.seed(42)
d <- data.frame(x = runif(200, 0, 100), y = runif(200, 0, 100),
                a = rnorm(200))
d$z <- d$a * 2 + sin(d$x / 20) + rnorm(200, sd = 0.5)
tf <- borg_transferability(d, z ~ a, coords = c("x", "y"), n_regions = 4)
tf
```

---

borg_unregister_hooks *Unregister BORG Hooks*

Description

Removes BORG validation hooks from framework functions.

Usage

```
borg_unregister_hooks()
```

Value

Invisible NULL.

---

borg_validate *Validate Complete Evaluation Workflow*

Description

`borg_validate()` performs post-hoc validation of an entire evaluation workflow, checking all components for information leakage.

Usage

```
borg_validate(workflow, strict = FALSE)
```

Arguments

`workflow`	A list containing the evaluation workflow components: **data** The full dataset **train_idx** Integer vector of training indices **test_idx** Integer vector of test indices **preprocess** Optional preprocessing object(s) **model** The fitted model object **predictions** Model predictions on test data **metrics** Computed evaluation metrics
`strict`	Logical. If TRUE, any hard violation causes an error. Default: FALSE (returns report only).

Details

`borg_validate()` inspects each component of an evaluation workflow:

1. **Split validation**: Checks train/test index isolation
2. **Preprocessing audit**: Traces preprocessing parameters to verify train-only origin
3. **Feature audit**: Checks for target leakage and proxy features
4. **Model audit**: Validates that model used only training data
5. **Threshold audit**: Checks if any thresholds were optimized on test data

Value

A `BorgRisk` object containing a comprehensive assessment of the workflow.

See Also

`borg` for proactive enforcement, `borg_inspect` for single-object inspection.

Examples

```
# Validate an existing workflow
data <- data.frame(x = rnorm(100), y = rnorm(100))
result <- borg_validate(list(
  data = data,
  train_idx = 1:70,
  test_idx = 71:100
))

# Check validity
if (!result@is_valid) {
  print(result)  # Shows detailed risk report
}
```

---

borg_vfold_cv *BORG-Guarded vfold_cv*

Description

A guarded version of `rsample::vfold_cv()` that checks for data dependencies before creating folds. If spatial, temporal, or clustered dependencies are detected, random CV is blocked.

Usage

```
borg_vfold_cv(
  data,
  v = 10,
  repeats = 1,
  strata = NULL,
  coords = NULL,
  time = NULL,
  groups = NULL,
  target = NULL,
  allow_override = FALSE,
  auto_block = FALSE,
  ...
)
```

Arguments

data	A data frame.
v	Integer. Number of folds. Default: 10.
repeats	Integer. Number of repeats. Default: 1.
strata	Character. Column name for stratification.
coords	Character vector of length 2. Coordinate columns for spatial check.
time	Character. Time column for temporal check.
groups	Character. Group column for clustered check.
target	Character. Target variable for dependency detection.
allow_override	Logical. If TRUE, allow random CV with explicit confirmation. Default: FALSE.
auto_block	Logical. If TRUE, automatically switch to blocked CV when dependencies detected. If FALSE, throw error. Default: FALSE.
...	Additional arguments passed to `rsample::vfold_cv()`.

Value

If no dependencies detected or `allow_override = TRUE`, returns an `rset` object from rsample. If dependencies detected and `auto_block = TRUE`, returns BORG-generated blocked CV folds.

See Also

borg_cv for direct blocked CV generation.

Examples

```
if (requireNamespace("rsample", quietly = TRUE)) {
  # Safe: no dependencies
  data <- data.frame(x = rnorm(100), y = rnorm(100))
  folds <- borg_vfold_cv(data, v = 5)
```

```
  # Use auto_block to automatically switch to spatial CV:
  spatial_data <- data.frame(
    lon = runif(100, -10, 10),
    lat = runif(100, -10, 10),
    response = rnorm(100)
  )
  folds <- borg_vfold_cv(spatial_data, coords = c("lon", "lat"),
                         target = "response", auto_block = TRUE)
}
```

`borg_willmott` *Willmott's Index of Agreement*

Description

Computes Willmott's d (original, refined d1, and modified dr) for spatial model assessment.

Usage

```
borg_willmott(actual, predicted)
```

Arguments

`actual`	Numeric vector of observed values.
`predicted`	Numeric vector of predicted values.

Value

A list with `d` (original), `d1` (refined), and `dr` (modified).

References

Willmott, C. J. (1981). On the validation of models. *Physical Geography*, 2(2), 184-194. doi:10.1080/02723646.1981.10642213

`borg_workflow` *Create a BORG Validation Workflow*

Description

Wraps the full model evaluation pipeline into a single trackable object: diagnose data dependencies, generate appropriate CV folds, fit a model per fold, and validate each split for leakage.

Usage

```
borg_workflow(
  data,
  formula,
  coords = NULL,
  time = NULL,
  groups = NULL,
  v = 5,
  fit_fun = stats::lm,
  metric = c("rmse", "mae", "rsq"),
  buffer = NULL,
  parallel = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

`data`	Data frame with predictor and target columns.
`formula`	Model formula (e.g. `y ~ x1 + x2`).
`coords`	Character vector of coordinate column names.
`time`	Character. Time column name.
`groups`	Character. Group column name.
`v`	Integer. Number of folds. Default: 5.
`fit_fun`	Function to fit a model. Default: `lm`. Must accept `formula` and `data` and return an object with `predict()`.
`metric`	Character. Performance metric. Default: `"rmse"`.
`buffer`	Numeric. Optional spatial buffer.
`parallel`	Logical. If `TRUE` and **future.apply** is installed, fold fitting and evaluation run in parallel. Default: `FALSE`.
`verbose`	Logical. Default: FALSE.
`...`	Additional arguments passed to `borg_cv()`.

Value

A `borg_workflow` object (list) containing:

diagnosis BorgDiagnosis object

cv borg_cv object with folds

models List of fitted models (one per fold)

predictions List of prediction vectors (one per fold)

performance borg_fold_perf data frame

risks List of BorgRisk objects (one per fold)

data Original data

formula Model formula

params Workflow parameters

Examples

```
set.seed(42)
d <- data.frame(
  x = runif(100, 0, 100), y = runif(100, 0, 100),
  z = rnorm(100)
)
wf <- borg_workflow(d, z ~ x + y, coords = c("x", "y"))
wf
```

`cv_leakage_report` *Generate CV Leakage Report*

Description

Generates a detailed report of cross-validation leakage issues.

Usage

```
cv_leakage_report(cv_object, train_idx, test_idx)
```

Arguments

`cv_object`	A cross-validation object (trainControl, vfold_cv, etc.).
`train_idx`	Integer vector of training indices.
`test_idx`	Integer vector of test indices.

Value

A list with detailed CV leakage information.

Examples

```
# Using caret trainControl
if (requireNamespace("caret", quietly = TRUE)) {
  folds <- list(Fold1 = 1:10, Fold2 = 11:20, Fold3 = 21:25)
  ctrl <- caret::trainControl(method = "cv", index = folds)
  report <- cv_leakage_report(ctrl, train_idx = 1:25, test_idx = 26:32)
  print(report)
}
```

`plot.BorgRisk` *Plot BORG Objects*

Description

S3 plot method for BORG risk assessment objects.

Usage

```
## S3 method for class 'BorgRisk'
plot(x, title = NULL, max_risks = 10, ...)
```

Arguments

`x`	A `BorgRisk` object from `borg_inspect()` or `borg()`.
`title`	Optional custom plot title.
`max_risks`	Maximum number of risks to display. Default: 10.
`...`	Additional arguments (currently unused).

Details

Displays a visual summary of detected risks:

- Hard violations shown in red
- Soft inflation risks shown in yellow/orange
- Green "OK" when no risks detected

Value

Invisibly returns NULL. Called for plotting side effect.

Examples

```
# No risks
data <- data.frame(x = 1:100, y = 101:200)
result <- borg_inspect(data, train_idx = 1:70, test_idx = 71:100)
plot(result)

# With overlap violation
result_bad <- borg_inspect(data, train_idx = 1:60, test_idx = 51:100)
plot(result_bad)
```

---

plot.borg_comparison *Plot CV Comparison Results*

Description

Creates a visualization comparing random vs blocked CV performance.

Usage

```
## S3 method for class 'borg_comparison'
plot(x, type = c("boxplot", "density", "paired"), ...)
```

Arguments

x	A borg_comparison object from borg_compare_cv.
type	Character. Plot type: "boxplot" (default), "density", or "paired".
...	Additional arguments passed to plotting functions.

Value

The borg_comparison object x, returned invisibly. Called for the side effect of producing a plot.

---

plot.borg_result *Plot BORG Result Objects*

Description

S3 plot method for borg_result objects from borg().

Usage

```
## S3 method for class 'borg_result'
plot(
  x,
  type = c("split", "risk", "temporal", "groups"),
  fold = 1,
  time = NULL,
  groups = NULL,
  title = NULL,
  ...
)
```

Arguments

x	A borg_result object from borg().
type	Character. Plot type: "split" (default), "risk", "temporal", or "groups".
fold	Integer. Which fold to plot (for split visualization). Default: 1.
time	Column name or values for temporal plots.
groups	Column name or values for group plots.
title	Optional custom plot title.
...	Additional arguments passed to internal plot functions.

Value

Invisibly returns NULL. Called for plotting side effect.

Examples

```
set.seed(42)
data <- data.frame(
  x = runif(100, 0, 100),
  y = runif(100, 0, 100),
  response = rnorm(100)
)
result <- borg(data, coords = c("x", "y"), target = "response")
plot(result)  # Split visualization for first fold
```

---

print.borg_cv_report *Print CV Leakage Report*

Description

Print CV Leakage Report

Usage

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

Arguments

x	A borg_cv_report object.
...	Additional arguments (ignored).

Value

The borg_cv_report object x, returned invisibly. Called for the side effect of printing a human-readable leakage summary to the console.

---

summary.BorgDiagnosis *Summarize BORG Diagnosis*

Description

Generate a methods section summary for publication from a BorgDiagnosis object.

Usage

```
## S3 method for class 'BorgDiagnosis'
summary(
  object,
  comparison = NULL,
  v = 5,
  style = c("apa", "nature", "ecology"),
  include_citation = TRUE,
  ...
)
```

Arguments

object	A BorgDiagnosis object.
comparison	Optional. A borg_comparison object from borg_compare_cv() to include empirical inflation estimates.
v	Integer. Number of CV folds used. Default: 5.
style	Character. Citation style: "apa" (default), "nature", "ecology".
include_citation	Logical. Include BORG package citation. Default: TRUE.
...	Additional arguments (currently unused).

Value

Character string with methods section text (invisibly). Also prints the text to the console.

Examples

```
set.seed(42)
data <- data.frame(
  x = runif(100, 0, 100),
  y = runif(100, 0, 100),
  response = rnorm(100)
)
diagnosis <- borg_diagnose(data, coords = c("x", "y"), target = "response",
                           verbose = FALSE)
summary(diagnosis)
```

---

summary.BorgRisk *Summarize BORG Risk Assessment*

Description

Print a summary of detected risks.

Usage

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

Arguments

object	A BorgRisk object from borg_inspect().
...	Additional arguments (currently unused).

Value

The object invisibly.

Examples

```
data <- data.frame(x = 1:100, y = 101:200)
risk <- borg_inspect(data, train_idx = 1:60, test_idx = 51:100)
summary(risk)
```

summary.borg_cv *Summarize BORG Cross-Validation*

Description

Summarize BORG Cross-Validation

Usage

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

Arguments

object	A borg_cv object from borg_cv.
...	Additional arguments (currently unused).

Value

A list with strategy, fold count, and fold size statistics (invisibly).

summary.borg_pipeline *Summarize BORG Pipeline Validation*

Description

Summarize BORG Pipeline Validation

Usage

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

Arguments

object	A borg_pipeline object from borg_pipeline.
...	Additional arguments (currently unused).

Value

A list with per-stage risk counts (invisibly).

summary.borg_power *Summarize BORG Power Analysis*

Description

Summarize BORG Power Analysis

Usage

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

Arguments

object	A borg_power object from borg_power.
...	Additional arguments (currently unused).

Value

A list with key power metrics (invisibly).

summary.borg_result *Summarize BORG Result*

Description

Generate a methods section summary for publication from a borg_result object.

Usage

```
## S3 method for class 'borg_result'
summary(
  object,
  comparison = NULL,
  v = 5,
  style = c("apa", "nature", "ecology"),
  include_citation = TRUE,
  ...
)
```

Arguments

object	A borg_result object from borg().
comparison	Optional. A borg_comparison object.
v	Integer. Number of CV folds. Default: 5.
style	Character. Citation style.
include_citation	Logical. Include BORG citation.
...	Additional arguments (currently unused).

Value

Character string with methods text (invisibly).

Examples

```
set.seed(42)
data <- data.frame(
  x = runif(100, 0, 100),
  y = runif(100, 0, 100),
  response = rnorm(100)
)
result <- borg(data, coords = c("x", "y"), target = "response")
summary(result)
```

Index