Package 'BORG'

March 20, 2026

Title Bounded Outcome Risk Guard for Model Evaluation

Version 0.2.5

Description Automatically detects and enforces valid model evaluation by identifying information reuse between training and evaluation data. Guards against data leakage, look-ahead bias, and invalid cross-validation schemes that inflate performance estimates. Supports temporal, spatial, and grouped evaluation structures. Based on evaluation principles described in Roberts et al. (2017) <doi:10.1111/ecog.02881>, Kaufman et al. (2012) <doi:10.1145/2382577.2382579>, and Kapoor & Narayanan (2023) <doi:10.1016/j.patter.2023.100804>.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

RoxygenNote 7.3.3

LinkingTo Rcpp

Imports Rcpp, methods, stats, utils

Suggests caret, rsample, tidyselect, recipes, mlr3, sf, ranger, parsnip, workflows, xgboost, lightgbm, 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 (>= 3.5)

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 2026-03-20 12:30:09 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)
```

---

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,
  v = 5,
  train_idx = NULL,
```

```
  test_idx = NULL,
  output = c("list", "rsample", "caret", "mlr3"),
  ...
)
```

Arguments

`data`	A data frame to diagnose and create CV folds for.
`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.
`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.
`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_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_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_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_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,
  block_size = NULL,
  embargo = NULL,
```

```
  strategy = NULL,
  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.
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.
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).
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_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 to diagnose.
`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_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_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_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_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_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_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_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)
}
```

---

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