Package 'gp3ml'

July 30, 2026

Type Package

Title Governance-First Predictive Modelling for 'Gazepoint' Research

Version 0.1.0

Description Provides governance-first infrastructure for leakage-resistant predictive modelling and validation using 'Gazepoint'-derived research data. Supports explicit task and role declarations, feature-provenance manifests, group-aware holdout splitting and repeated resampling, fold-local preprocessing, governed model engines, discrimination and calibration metrics, bootstrap uncertainty, external-validation reports, model cards, and reproducibility reports. Intended only for explicitly observed, non-sensitive outcomes and declared scientific purposes. Use is prohibited for person identification, biometric authentication, health or protected-attribute inference, and direct or indirect inference of emotion, stress, personality, deception, cognition, comprehension, intent, or other mental states.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Suggests jsonlite, keras3, knitr, nnet, pkgdown, ranger, rmarkdown, testthat (>= 3.0.0), xgboost

RoxygenNote 8.0.0

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

URL https://stefanosbalaskas.github.io/gp3ml/, https://github.com/stefanosbalaskas/gp3ml

BugReports https://github.com/stefanosbalaskas/gp3ml/issues

NeedsCompilation no

Author Stefanos Balaskas [aut, cre] (ORCID: <https://orcid.org/0000-0003-2444-9796>)

Maintainer Stefanos Balaskas <s.balaskas@ac.upatras.gr>

Repository CRAN

Date/Publication 2026-07-30 17:30:08 UTC

Contents

`apply_gazepoint_calibrator` *Apply a fitted probability calibrator*

Description

Apply a fitted probability calibrator

Usage

```
apply_gazepoint_calibrator(calibrator, probability)
```

Arguments

`calibrator`	A fitted `gp3ml_calibrator` object.
`probability`	Uncalibrated probabilities to transform.

Value

A numeric vector of calibrated probabilities, clipped to the open unit interval.

Examples

```
truth <- factor(
  rep(c("pass", "review"), 6),
  levels = c("pass", "review")
)
probability <- c(
  0.20, 0.70, 0.60, 0.55, 0.30, 0.80,
  0.65, 0.45, 0.40, 0.75, 0.50, 0.60
)
calibrator <- fit_gazepoint_calibrator(
  truth = truth,
  probability = probability,
  positive = "review"
)
apply_gazepoint_calibrator(
```

```
  calibrator,
  probability
)
```

`assert_gp3ml_use_case` *Assert that a task is within the permitted gp3ml scope*

Description

Assert that a task is within the permitted gp3ml scope

Usage

```
assert_gp3ml_use_case(task, data = NULL)
```

Arguments

`task`	A `gp3ml_task` object.
`data`	Optional data frame used to validate task columns.

Value

Invisibly returns `TRUE` when the task is permitted; otherwise, the function stops with an error.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
```

```
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
assert_gp3ml_use_case(task, example_data)
```

`assess_gazepoint_calibration` *Calibration assessment with bootstrap uncertainty*

Description

Calibration assessment with bootstrap uncertainty

Usage

```
assess_gazepoint_calibration(
  truth,
  probability,
  positive = NULL,
  bins = 10L,
  bootstrap = 200L,
  conf_level = 0.95,
  seed = 1L
)
```

Arguments

`truth`	Observed binary outcome values.
`probability`	Predicted positive-class probabilities.
`positive`	Label representing the positive class.
`bins`	Number of reliability bins.
`bootstrap`	Number of bootstrap replicates.
`conf_level`	Confidence level for percentile intervals.
`seed`	Deterministic random seed.

Value

A `gp3ml_calibration_assessment` object containing calibration summaries, reliability-bin results, bootstrap intervals, and assessment settings.

Examples

```
truth <- factor(
  rep(rep(c("pass", "review"), 5), 10),
  levels = c("pass", "review")
)
probability <- rep(
  seq(0.10, 0.90, length.out = 10),
  each = 10
)

assessment <- assess_gazepoint_calibration(
  truth = truth,
  probability = probability,
  positive = "review",
  bins = 5L,
  bootstrap = 10L,
  seed = 101L
)
assessment
```

---

audit_gazepoint_group_folds

Aggregate leakage audits across group-aware folds

Description

Aggregate leakage audits across group-aware folds

Usage

```
audit_gazepoint_group_folds(x)
```

Arguments

x	A gazepoint_group_folds object.

Value

An object of class gazepoint_group_folds_audit.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
```

```
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
```

```
  seed = 101L
)
audit <- audit_gazepoint_group_folds(folds)
audit
```

`audit_gazepoint_ml_leakage` *Audit leakage between predictive-analysis partitions*

Description

Audits already-defined analysis and assessment partitions for common forms of leakage and for incompatibility with a declared generalization target. The function does not create data splits, preprocess variables, select features, or fit predictive models.

Usage

```
audit_gazepoint_ml_leakage(
  analysis,
  assessment,
  outcome,
  predictors,
  participant_id = NULL,
  trial_id = NULL,
  stimulus_id = NULL,
  generalization_target = c("new_trials_known_participants", "new_participants",
    "new_stimuli", "new_participants_and_new_stimuli"),
  target_derived = character(),
  post_outcome = character()
)
```

Arguments

`analysis`	A data frame containing the analysis or training partition.
`assessment`	A data frame containing the assessment or test partition.
`outcome`	A single column name identifying the outcome.
`predictors`	A character vector identifying intended predictor columns.
`participant_id`	An optional participant-identifier column.
`trial_id`	An optional trial-identifier column.
`stimulus_id`	An optional stimulus-identifier column.
`generalization_target`	The predictive generalization target. One of `"new_trials_known_participants"`, `"new_participants"`, `"new_stimuli"`, or `"new_participants_and_new_stimuli"`.
`target_derived`	Character vector of columns known to have been derived directly from the outcome.
`post_outcome`	Character vector of columns measured or constructed after the outcome became available.

Details

The overall status is "fail" when at least one failing check is present, "review" when no failing checks are present but at least one review item is present, and "pass" otherwise.

When participant_id is supplied, trial overlap is evaluated using composite participant-trial units. This permits trial labels such as "T01" to be reused by different participants without being treated as leakage. Without participant_id, trial_id is assumed to be globally unique.

The audit can identify structural leakage visible in the supplied partitions and declared variable roles. It cannot prove that preprocessing or feature selection was estimated inside resampling folds. Those operations require separate provenance and resampling safeguards.

The function does not determine whether an outcome is scientifically or ethically appropriate. All uses remain subject to the package governance and prohibited-use statements.

Value

An object of class gazepoint_ml_leakage_audit. The object contains an overall status, partition summary, complete check table, and machine-readable table of non-passing issues.

Examples

```
analysis <- data.frame(
  participant_id = c("P01", "P01", "P02", "P02"),
  trial_id = c("T01", "T02", "T03", "T04"),
  stimulus_id = c("S01", "S02", "S03", "S04"),
  outcome = c(0, 1, 0, 1),
  fixation_duration = c(210, 240, 225, 260),
  pupil_change = c(0.10, 0.16, 0.12, 0.18)
)

assessment <- data.frame(
  participant_id = c("P03", "P03", "P04", "P04"),
  trial_id = c("T05", "T06", "T07", "T08"),
  stimulus_id = c("S05", "S06", "S07", "S08"),
  outcome = c(1, 0, 1, 0),
  fixation_duration = c(275, 230, 290, 245),
  pupil_change = c(0.21, 0.11, 0.24, 0.14)
)

audit_gazepoint_ml_leakage(
  analysis = analysis,
  assessment = assessment,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants"
)
```

`bake_gazepoint_preprocessor` *Apply a fitted preprocessing engine*

Description

Apply a fitted preprocessing engine

Usage

```
bake_gazepoint_preprocessor(preprocessor, new_data)
```

Arguments

`preprocessor`	A fitted `gp3ml_preprocessor` object.
`new_data`	Data to transform using the fitted parameters.

Value

A numeric model matrix transformed using only the parameters stored in the fitted preprocessor.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
preprocessor <- fit_gazepoint_preprocessor(
  data = example_data,
  predictors = c(
    "fixation_duration",
    "pupil_change",
    "condition"
  )
)
```

```
baked <- bake_gazepoint_preprocessor(
  preprocessor,
  example_data
)
dim(baked)
```

`bootstrap_gazepoint_metrics` *Bootstrap uncertainty intervals for performance metrics*

Description

Bootstrap uncertainty intervals for performance metrics

Usage

```
bootstrap_gazepoint_metrics(
  task,
  truth,
  prediction = NULL,
  probability = NULL,
  threshold = 0.5,
  bootstrap = 1000L,
  conf_level = 0.95,
  seed = 1L
)
```

Arguments

`task`	A governed `gp3ml_task` object.
`truth`	Observed outcome values.
`prediction`	Predicted classes or numeric values.
`probability`	Predicted positive-class probabilities.
`threshold`	Probability threshold for classification.
`bootstrap`	Number of bootstrap replicates.
`conf_level`	Confidence level for percentile intervals.
`seed`	Deterministic random seed.

Value

A `gp3ml_metric_uncertainty` object containing point estimates, percentile intervals, bootstrap draws, resampling settings, and the governed task.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
probability <- seq(
  0.20,
  0.80,
  length.out = nrow(example_data)
)
predicted <- factor(
  ifelse(probability >= 0.5, "review", "pass"),
  levels = levels(example_data$quality_status)
)
uncertainty <- bootstrap_gazepoint_metrics(
  task = task,
  truth = example_data$quality_status,
  prediction = predicted,
  probability = probability,
  bootstrap = 10L,
  seed = 101L
)
uncertainty
```

`create_external_validation_report` *Create an external-validation report object*

Description

Create an external-validation report object

Usage

```
create_external_validation_report(
  validation,
  development_metrics = NULL,
  limitations = character()
)
```

Arguments

`validation`	A `gp3ml_external_validation` object.
`development_metrics`	Optional development-sample metrics.
`limitations`	Character vector describing report limitations.

Value

A `gp3ml_external_validation_report` object containing the validation result, optional development metrics, limitations, and prohibited-use information.

Examples

```
training_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
training_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = training_data,
  outcome = "quality_status",
```

```
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
model <- train_gazepoint_classifier(
  data = training_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  engine = "glm",
  seed = 101L
)
external_data <- training_data
external_data$participant_id <- rep(
  sprintf("E%02d", 1:12),
  each = 2
)
external_data$trial_id <- sprintf("ET%02d", 1:24)
external_data$fixation_duration <-
  external_data$fixation_duration + 4
external_data$pupil_change <- cos(seq_len(24) / 4)
validation <- evaluate_external_validation(
  model = model,
  external_data = external_data,
  label = "synthetic_external",
  bootstrap = 10L,
  seed = 101L
)
report <- create_external_validation_report(
  validation = validation,
  limitations = "Synthetic external-validation example."
)
report
```

`create_gazepoint_feature_manifest` *Create a Gazepoint feature-provenance manifest*

Description

Creates a structured provenance manifest for intended predictive features. Each row records where a feature originated, when it became available, whether it is outcome-derived or post-outcome, and where any data-dependent preprocessing was estimated.

Usage

```
create_gazepoint_feature_manifest(
```

```
  features,
  scientific_source = NA_character_,
  source_table = NA_character_,
  transformation = "none",
  availability_stage = "unknown",
  prediction_time_available = NA,
  outcome_derived = FALSE,
  post_outcome = FALSE,
  identifier = FALSE,
  preprocessing_scope = "unknown",
  fold_local_required = NA,
  reviewer_notes = ""
)
```

Arguments

`features`	Character vector of unique feature names.
`scientific_source`	Scientific or measurement source for each feature.
`source_table`	Source export, table, or object for each feature.
`transformation`	Description of the transformation used to construct each feature.
`availability_stage`	Availability stage for each feature. One of `"pre_exposure"`, `"during_exposure"`, `"post_exposure_pre_outcome"`, `"at_prediction"`, `"post_outcome"`, or `"unknown"`.
`prediction_time_available`	Logical vector indicating whether each feature is available at the intended prediction time.
`outcome_derived`	Logical vector indicating whether each feature was derived directly or indirectly from the outcome.
`post_outcome`	Logical vector indicating whether each feature was measured or constructed after the outcome became available.
`identifier`	Logical vector indicating whether each feature is an identifier or row-location variable.
`preprocessing_scope`	Scope in which any data-dependent preprocessing was estimated. One of `"none"`, `"global"`, `"analysis_partition"`, `"resampling_fold"`, or `"unknown"`.
`fold_local_required`	Logical vector indicating whether preprocessing for each feature must be estimated separately inside each resampling fold.
`reviewer_notes`	Optional reviewer-facing notes.

Details

Each row is treated as an intended predictor. Consequently, outcome-derived, post-outcome, unavailable, and identifier features are treated as failing conditions by `validate_gazepoint_feature_manifest()`.

The manifest records declared provenance. It does not independently prove that preprocessing was estimated within the stated scope.

Value

A data frame of class `gazepoint_feature_manifest`.

Examples

```
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint all-gaze export"
  ),
  source_table = c("fixations", "all_gaze"),
  transformation = c(
    "Trial-level mean",
    "Baseline-adjusted change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = c("none", "resampling_fold"),
  fold_local_required = c(FALSE, TRUE)
)

manifest
```

---

create_gazepoint_group_folds

Create deterministic group-aware Gazepoint resampling folds

Description

Creates repeated grouped assessment folds that preserve the grouping structure implied by an explicit generalization target. A passing feature-provenance manifest is required, and every analysis-assessment pair is evaluated using the leakage audit.

Usage

```
create_gazepoint_group_folds(
  data,
  outcome,
  predictors,
  feature_manifest,
  generalization_target,
  participant_id = NULL,
  trial_id = NULL,
  stimulus_id = NULL,
  v = 5L,
  repeats = 1L,
```

```
  seed = 1L,
  source_row_id = ".gp3ml_source_row"
)
```

Arguments

`data`	A data frame containing the outcome, predictors, and grouping identifiers.
`outcome`	Name of the outcome column.
`predictors`	Character vector naming intended predictors.
`feature_manifest`	A feature manifest containing all intended predictors.
`generalization_target`	One of `"new_trials_known_participants"`, `"new_participants"`, `"new_stimuli"`, or `"new_participants_and_new_stimuli"`.
`participant_id`	Optional participant-identifier column.
`trial_id`	Optional trial-identifier column.
`stimulus_id`	Optional stimulus-identifier column.
`v`	Number of group folds. For simultaneous participant and stimulus generalization, a length-two vector specifies participant and stimulus fold counts.
`repeats`	Number of repeated fold assignments.
`seed`	Integer random seed. The caller's random-number state is restored.
`source_row_id`	Name of the source-row identifier added to returned partitions.

Details

For new trials among known participants, participant-trial units are assigned separately within each participant. For simultaneous participant and stimulus generalization, crossed participant-stimulus assessment blocks are created; cross-block rows are excluded from that fold. Each source row appears in assessment exactly once per repeat.

This function does not perform preprocessing, feature selection, tuning, nested resampling, or model fitting.

Value

An object of class `gazepoint_group_folds`.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
```

```
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
```

```
folds
```

`create_gazepoint_model_card` *Create a governance-focused model card*

Description

Create a governance-focused model card

Usage

```
create_gazepoint_model_card(
  model,
  intended_use,
  evaluation = NULL,
  calibration = NULL,
  feature_manifest = NULL,
  external_validation = NULL,
  limitations = character(),
  ethical_review = NULL
)
```

Arguments

`model`	A fitted `gp3ml_model` object.
`intended_use`	Explicit description of the intended research use.
`evaluation`	Optional performance-evaluation object.
`calibration`	Optional calibration-assessment object.
`feature_manifest`	Optional feature-provenance manifest.
`external_validation`	Optional external-validation result.
`limitations`	Character vector describing model limitations.
`ethical_review`	Optional ethical-review information.

Value

A `gp3ml_model_card` object containing task, model, governance, evaluation, calibration, provenance, external-validation, and limitation metadata.

Examples

```
training_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
training_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = training_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
model <- train_gazepoint_classifier(
  data = training_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  engine = "glm",
  seed = 101L
)
card <- create_gazepoint_model_card(
  model = model,
  intended_use = paste(
    "Support manual review of predefined",
    "recording-quality status"
  ),
  limitations = "Synthetic example for documentation."
)
card
```

---

create_gazepoint_reproducibility_report

Create a reproducibility report

Description

Create a reproducibility report

Usage

```
create_gazepoint_reproducibility_report(
  objects = list(),
  data = NULL,
  seeds = list(),
  notes = character(),
  project_path = getwd()
)
```

Arguments

objects	Named objects to fingerprint.
data	Optional data frame to fingerprint.
seeds	Named list of deterministic seeds.
notes	Optional reproducibility notes.
project_path	Project directory recorded in the report.

Value

A gp3ml_reproducibility_report object containing runtime information, object and data fingerprints, seeds, Git metadata, notes, and prohibited uses.

Examples

```
example_data <- data.frame(
  trial_id = sprintf("T%02d", 1:6),
  fixation_duration = c(190, 205, 198, 214, 202, 220),
  stringsAsFactors = FALSE
)
report <- create_gazepoint_reproducibility_report(
  objects = list(
    fixation_values = example_data$fixation_duration
  ),
  data = example_data,
  seeds = list(example = 101L),
  notes = "Synthetic documentation example.",
  project_path = tempdir()
)
report
```

declare_gazepoint_task *Declare a governed Gazepoint prediction task*

Description

Declare a governed Gazepoint prediction task

Usage

```
declare_gazepoint_task(
  data,
  outcome,
  purpose,
  task_type = c("classification", "regression"),
  unit_id,
  participant_id = NULL,
  stimulus_id = NULL,
  generalization_target = c("new_trials_known_participants", "new_participants",
    "new_stimuli", "new_participants_and_new_stimuli", "external_validation"),
  positive = NULL,
  observed_outcome = TRUE,
  sensitive_outcome = FALSE
)
```

Arguments

data	A data frame containing the outcome and task identifiers.
outcome	Name of the explicitly observed outcome column.
purpose	One explicit scientific-purpose statement.
task_type	Either classification or regression.
unit_id	Column identifying the prediction unit.
participant_id	Optional participant-identifier column.
stimulus_id	Optional stimulus-identifier column.
generalization_target	The intended target of generalization.
positive	Positive outcome level for binary classification.
observed_outcome	Whether the outcome was directly observed.
sensitive_outcome	Whether the outcome is sensitive or prohibited.

Value

A governed gp3ml_task object describing the outcome, scientific purpose, prediction unit, grouping roles, task type, and generalization target.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
task
```

---

diagnose_gazepoint_group_folds

Diagnose group-aware Gazepoint resampling folds

Description

Creates fold-size, repeat-level, grouping, assessment-coverage, outcome-balance, and exclusion diagnostics for an existing `gazepoint_group_folds` object.

Usage

```
diagnose_gazepoint_group_folds(x, imbalance_review = 1.5, imbalance_fail = 2)
```

Arguments

`x`	A `gazepoint_group_folds` object.

`imbalance_review` Fold-size ratio above which diagnostics receive a `review` status.

`imbalance_fail` Fold-size ratio above which diagnostics receive a `fail` status.

Value

An object of class `gazepoint_fold_diagnostics`.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
```

```
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
diagnostics <- diagnose_gazepoint_group_folds(folds)
diagnostics
```

`evaluate_external_validation` *Evaluate an independent external-validation dataset*

Description

Evaluate an independent external-validation dataset

Usage

```
evaluate_external_validation(
  model,
  external_data,
  label = "external",
  threshold = model$threshold,
  bootstrap = 200L,
  seed = 1L
)
```

Arguments

`model`	A fitted `gp3ml_model` object.
`external_data`	Independent external-validation data.
`label`	Label identifying the validation dataset.
`threshold`	Classification probability threshold.

bootstrap	Number of calibration bootstrap replicates.
seed	Deterministic random seed.

Value

A gp3ml_external_validation object containing external predictions, performance metrics, calibration results where applicable, predictor-shift diagnostics, a dataset fingerprint, and task metadata.

Examples

```
training_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
training_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = training_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
model <- train_gazepoint_classifier(
  data = training_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  engine = "glm",
  seed = 101L
)
external_data <- training_data
external_data$participant_id <- rep(
  sprintf("E%02d", 1:12),
  each = 2
)
external_data$trial_id <- sprintf("ET%02d", 1:24)
external_data$fixation_duration <-
```

```
  external_data$fixation_duration + 4
external_data$pupil_change <- cos(seq_len(24) / 4)
validation <- evaluate_external_validation(
  model = model,
  external_data = external_data,
  label = "synthetic_external",
  bootstrap = 10L,
  seed = 101L
)
validation
```

`fit_gazepoint_calibrator` *Fit a probability calibrator*

Description

Fit a probability calibrator

Usage

```
fit_gazepoint_calibrator(
  truth,
  probability,
  positive = NULL,
  method = c("platt", "isotonic")
)
```

Arguments

`truth`	Observed binary outcome values.
`probability`	Uncalibrated positive-class probabilities.
`positive`	Label representing the positive class.
`method`	Calibration method: Platt scaling or isotonic regression.

Value

A fitted `gp3ml_calibrator` object containing the calibration method, fitted model, and outcome labels.

Examples

```
truth <- factor(
  rep(c("pass", "review"), 6),
  levels = c("pass", "review")
)
probability <- c(
  0.20, 0.70, 0.60, 0.55, 0.30, 0.80,
```

```
  0.65, 0.45, 0.40, 0.75, 0.50, 0.60
)
calibrator <- fit_gazepoint_calibrator(
  truth = truth,
  probability = probability,
  positive = "review",
  method = "platt"
)
calibrator
```

`fit_gazepoint_deep_model` *Fit an optional governed deep-learning model through keras3*

Description

Fit an optional governed deep-learning model through keras3

Usage

```
fit_gazepoint_deep_model(
  data,
  task,
  predictors = NULL,
  preprocessor = NULL,
  hidden_units = c(64L, 32L),
  dropout = 0.2,
  epochs = 50L,
  batch_size = 32L,
  validation_split = 0.2,
  optimizer = "adam",
  seed = 1L,
  verbose = 0L
)
```

Arguments

`data`	Analysis data used to fit the network.
`task`	A governed `gp3ml_task` object.
`predictors`	Optional character vector of predictor columns.
`preprocessor`	Optional fitted preprocessing object.
`hidden_units`	Integer vector of hidden-layer sizes.
`dropout`	Dropout proportion applied after hidden layers.
`epochs`	Number of training epochs.
`batch_size`	Training batch size.

validation_split
: Proportion reserved for internal validation.

optimizer
: Keras optimizer name or object.

seed
: Deterministic random seed.

verbose
: Keras training verbosity.

Value

A governed gp3ml_model object containing the fitted keras3 model, training history, preprocessing object, task contract, and training metadata.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
deep_model <- fit_gazepoint_deep_model(
  data = example_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  hidden_units = 4L,
  dropout = 0,
  epochs = 1L,
  batch_size = 8L,
  validation_split = 0,
```

```
  seed = 101L,
  verbose = 0L
)
deep_model
```

`fit_gazepoint_model` *Fit a governed Gazepoint model*

Description

Fit a governed Gazepoint model

Usage

```
fit_gazepoint_model(
  data,
  task,
  predictors = NULL,
  engine = NULL,
  preprocessor = NULL,
  preprocessor_args = list(),
  engine_args = list(),
  seed = 1L,
  threshold = 0.5
)
```

Arguments

`data`	Analysis data used to fit the model.
`task`	A governed `gp3ml_task` object.
`predictors`	Optional character vector of predictor columns.
`engine`	Engine name or controlled custom-engine object.
`preprocessor`	Optional fitted preprocessing object.
`preprocessor_args`	Arguments passed to preprocessing fitting.
`engine_args`	Arguments passed to the model engine.
`seed`	Deterministic random seed.
`threshold`	Classification probability threshold.

Value

A governed `gp3ml_model` object containing the fitted engine, preprocessing object, task contract, predictors, and training metadata.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
model <- fit_gazepoint_model(
  data = example_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  engine = "glm",
  seed = 101L
)
model
```

---

fit_gazepoint_preprocessor

Fit a fold-local preprocessing engine

Description

Fit a fold-local preprocessing engine

Usage

```
fit_gazepoint_preprocessor(
  data,
  predictors,
  numeric_imputation = c("median", "mean"),
  center = TRUE,
  scale = TRUE,
  novel_level = c("other", "error"),
  remove_zero_variance = TRUE
)
```

Arguments

`data`	Analysis data used to estimate preprocessing parameters.
`predictors`	Character vector naming predictor columns.
`numeric_imputation`	Numeric imputation method.
`center`	Whether numeric model columns should be centered.
`scale`	Whether numeric model columns should be scaled.
`novel_level`	How novel categorical levels should be handled.
`remove_zero_variance`	Whether zero-variance columns are removed.

Value

A fitted `gp3ml_preprocessor` object containing analysis-partition imputation values, factor levels, model columns, centering values, and scaling values.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
preprocessor <- fit_gazepoint_preprocessor(
```

```
  data = example_data,
  predictors = c(
    "fixation_duration",
    "pupil_change",
    "condition"
  )
)
preprocessor
```

`gazepoint_classification_metrics` *Binary classification metrics*

Description

Binary classification metrics

Usage

```
gazepoint_classification_metrics(
  truth,
  probability,
  predicted = NULL,
  positive = NULL,
  threshold = 0.5
)
```

Arguments

`truth`	Observed binary outcome values.
`probability`	Predicted positive-class probabilities.
`predicted`	Optional predicted classes.
`positive`	Label representing the positive class.
`threshold`	Probability threshold used for class predictions.

Value

A one-row data frame containing the sample size, threshold, class-performance measures, discrimination metrics, Brier score, and log loss.

Examples

```
truth <- factor(
  rep(c("pass", "review"), 6),
  levels = c("pass", "review")
)
probability <- c(
```

```
  0.20, 0.70, 0.60, 0.55, 0.30, 0.80,
  0.65, 0.45, 0.40, 0.75, 0.50, 0.60
)
predicted <- factor(
  ifelse(probability >= 0.5, "review", "pass"),
  levels = levels(truth)
)
gazepoint_classification_metrics(
  truth = truth,
  probability = probability,
  predicted = predicted,
  positive = "review"
)
```

`gazepoint_performance_metrics` *Task-aware performance metrics*

Description

Task-aware performance metrics

Usage

```
gazepoint_performance_metrics(
  task,
  truth,
  prediction = NULL,
  probability = NULL,
  threshold = 0.5
)
```

Arguments

`task`	A governed `gp3ml_task` object.
`truth`	Observed outcome values.
`prediction`	Predicted classes or numeric values.
`probability`	Predicted positive-class probabilities.
`threshold`	Probability threshold for classification.

Value

A one-row data frame of classification or regression metrics selected according to the governed task type.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
probability <- seq(
  0.20,
  0.80,
  length.out = nrow(example_data)
)
predicted <- factor(
  ifelse(probability >= 0.5, "review", "pass"),
  levels = levels(example_data$quality_status)
)
gazepoint_performance_metrics(
  task = task,
  truth = example_data$quality_status,
  prediction = predicted,
  probability = probability
)
```

---

gazepoint_regression_metrics

Regression metrics

Description

Regression metrics

Usage

```
gazepoint_regression_metrics(truth, prediction)
```

Arguments

truth	Observed numeric outcome values.
prediction	Predicted numeric outcome values.

Value

A one-row data frame containing the sample size, RMSE, MAE, R-squared value, and prediction correlation.

Examples

```
truth <- c(1.0, 2.0, 3.0, 4.0, 5.0)
prediction <- c(1.1, 1.8, 3.2, 3.9, 4.8)
gazepoint_regression_metrics(truth, prediction)
```

---

gp3ml_available_engines

List available model engines

Description

List available model engines

Usage

```
gp3ml_available_engines()
```

Value

A data frame listing supported model-engine names and whether each optional engine is currently available.

Examples

```
gp3ml_available_engines()
```

gp3ml_prohibited_uses *Prohibited gp3ml uses*

Description

Prohibited gp3ml uses

Usage

```
gp3ml_prohibited_uses()
```

Value

A character vector of prohibited use descriptions.

Examples

```
gp3ml_prohibited_uses()
```

integrate_black_box_model *Integrate a controlled black-box model engine*

Description

Integrate a controlled black-box model engine

Usage

```
integrate_black_box_model(
  name,
  fit_fun,
  predict_fun,
  supports = c("classification", "regression"),
  probability = TRUE,
  metadata = list(),
  safety_declaration
)
```

Arguments

name	Unique name for the custom engine.
fit_fun	Function that fits the custom engine.
predict_fun	Function that generates predictions.
supports	Task types supported by the engine.
probability	Whether classification probabilities are supported.
metadata	Optional engine metadata.
safety_declaration	Named logical safety declarations.

Value

A controlled gp3ml_engine object containing the custom fit and prediction functions, supported task types, metadata, and explicit safety declarations.

Examples

```
custom_fit <- function(x, y, task, args) {
  training_data <- data.frame(
    .outcome = y,
    x,
    check.names = FALSE
  )
  stats::glm(
    .outcome ~ .,
    data = training_data,
    family = stats::binomial()
  )
}
custom_predict <- function(fit, newdata, type, task, ...) {
  as.numeric(stats::predict(
    fit,
    newdata = as.data.frame(newdata),
    type = "response"
  ))
}
engine <- integrate_black_box_model(
  name = "custom_glm",
  fit_fun = custom_fit,
  predict_fun = custom_predict,
  supports = "classification",
  probability = TRUE,
  safety_declaration = list(
    prohibited_uses_acknowledged = TRUE,
    prediction_time_inputs_only = TRUE,
    group_aware_evaluation_required = TRUE
  )
)
engine
```

`predict.gp3ml_model` *Predict from a gp3ml model*

Description

Predict from a gp3ml model

Usage

```
## S3 method for class 'gp3ml_model'
predict(
  object,
  newdata,
  type = c("response", "probability", "class", "link"),
  ...
)
```

Arguments

`object`	A fitted `gp3ml_model` object.
`newdata`	New data containing the required predictors.
`type`	Requested prediction type.
`...`	Additional arguments passed to custom prediction methods.

Value

For classification with `type = "class"`, a factor of predicted classes. Otherwise, a numeric vector of probabilities, link-scale values, or regression predictions.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
```

```
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
model <- train_gazepoint_classifier(
  data = example_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  engine = "glm",
  seed = 101L
)
probability <- predict(
  model,
  example_data,
  type = "probability"
)
predicted_class <- predict(
  model,
  example_data,
  type = "class"
)
head(probability)
head(predicted_class)
```

`print.gazepoint_feature_manifest_validation` *Print feature-manifest validation*

Description

Print feature-manifest validation

Usage

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

Arguments

`x`	An object returned by `validate_gazepoint_feature_manifest()`.
`...`	Additional arguments, currently unused.

Value

x, invisibly.

Examples

```
manifest <- create_gazepoint_feature_manifest(
  features = "fixation_duration",
  scientific_source = "Gazepoint fixation export",
  source_table = "fixations",
  transformation = "Trial-level mean",
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
validation <- validate_gazepoint_feature_manifest(manifest)
print(validation)
```

---

print.gazepoint_fold_diagnostics

Print Gazepoint fold diagnostics

Description

Print Gazepoint fold diagnostics

Usage

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

Arguments

x	A gazepoint_fold_diagnostics object.
...	Additional arguments, currently unused.

Value

x, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
```

```
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
```

```
  repeats = 1L,
  seed = 101L
)
diagnostics <- diagnose_gazepoint_group_folds(folds)
print(diagnostics)
```

`print.gazepoint_fold_diagnostics_validation` *Print Gazepoint fold-diagnostics validation*

Description

Print Gazepoint fold-diagnostics validation

Usage

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

Arguments

`x`	A `gazepoint_fold_diagnostics_validation` object.
`...`	Additional arguments, currently unused.

Value

`x`, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
```

```
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
diagnostics <- diagnose_gazepoint_group_folds(folds)
validation <- validate_gazepoint_fold_diagnostics(
  diagnostics
)
print(validation)
```

print.gazepoint_group_folds *Print group-aware Gazepoint resampling folds*

Description

Print group-aware Gazepoint resampling folds

Usage

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

Arguments

x	A gazepoint_group_folds object.
...	Additional arguments, currently unused.

Value

x, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
```

```
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
print(folds)
```

`print.gazepoint_group_folds_audit` *Print aggregated group-fold leakage auditing*

Description

Print aggregated group-fold leakage auditing

Usage

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

Arguments

x	A gazepoint_group_folds_audit object.
...	Additional arguments, currently unused.

Value

x, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
```

```
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
audit <- audit_gazepoint_group_folds(folds)
print(audit)
```

`print.gazepoint_group_folds_validation` *Print group-aware fold validation*

Description

Print group-aware fold validation

Usage

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

Arguments

`x`	A `gazepoint_group_folds_validation` object.
`...`	Additional arguments, currently unused.

Value

`x`, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
```

```
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
validation <- validate_gazepoint_group_folds(folds)
print(validation)
```

`print.gazepoint_ml_leakage_audit` *Print a Gazepoint ML leakage audit*

Description

Print a Gazepoint ML leakage audit

Usage

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

Arguments

`x`	An object returned by `audit_gazepoint_ml_leakage()`.
`...`	Additional arguments, currently unused.

Value

`x`, invisibly.

Examples

```
analysis <- data.frame(
  participant_id = c("P01", "P01", "P02", "P02"),
  trial_id = c("T01", "T02", "T03", "T04"),
  stimulus_id = c("S01", "S02", "S03", "S04"),
  outcome = c(0, 1, 0, 1),
  fixation_duration = c(210, 240, 225, 260),
  pupil_change = c(0.10, 0.16, 0.12, 0.18)
)
assessment <- data.frame(
  participant_id = c("P03", "P03", "P04", "P04"),
  trial_id = c("T05", "T06", "T07", "T08"),
```

```
  stimulus_id = c("S05", "S06", "S07", "S08"),
  outcome = c(1, 0, 1, 0),
  fixation_duration = c(275, 230, 290, 245),
  pupil_change = c(0.21, 0.11, 0.24, 0.14)
)
audit <- audit_gazepoint_ml_leakage(
  analysis = analysis,
  assessment = assessment,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants"
)
print(audit)
```

`print.gazepoint_ml_split` *Print a group-aware Gazepoint split*

Description

Print a group-aware Gazepoint split

Usage

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

Arguments

`x`	A `gazepoint_ml_split` object.
`...`	Additional arguments, currently unused.

Value

`x`, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
```

```
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
split <- split_gazepoint_ml_data(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  assessment_prop = 1 / 3,
  seed = 101L
)
```

```
print(split)
```

---

print.gazepoint_ml_split_validation
Print group-aware split validation

Description

Print group-aware split validation

Usage

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

Arguments

x	An object returned by validate_gazepoint_ml_split().
...	Additional arguments, currently unused.

Value

x, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
```

```
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
split <- split_gazepoint_ml_data(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  assessment_prop = 1 / 3,
  seed = 101L
)
validation <- validate_gazepoint_ml_split(split)
print(validation)
```

`split_gazepoint_ml_data` *Create a deterministic group-aware Gazepoint holdout split*

Description

Creates analysis and assessment partitions that preserve the grouping unit implied by an explicit generalization target.

Usage

```
split_gazepoint_ml_data(
  data,
  outcome,
  predictors,
  feature_manifest,
  generalization_target,
  participant_id = NULL,
  trial_id = NULL,
  stimulus_id = NULL,
  assessment_prop = 0.2,
  seed = 1L,
  source_row_id = ".gp3ml_source_row"
)
```

Arguments

`data`	Data frame containing the outcome, predictors, and grouping identifiers.
`outcome`	Name of the outcome column.
`predictors`	Character vector of predictor-column names.
`feature_manifest`	Feature manifest containing the predictors.
`generalization_target`	Declared predictive-generalization target.
`participant_id`	Optional participant-identifier column.
`trial_id`	Optional trial-identifier column.
`stimulus_id`	Optional stimulus-identifier column.
`assessment_prop`	Requested assessment proportion.
`seed`	Integer random seed.
`source_row_id`	Name of the source-row identifier added to the returned partitions.

Details

For simultaneous participant and stimulus generalization, cross-block rows are placed in the excluded partition.

This function does not perform preprocessing, feature selection, resampling, or model fitting.

Value

An object of class `gazepoint_ml_split`.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
split <- split_gazepoint_ml_data(
  data = example_data,
```

```
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  assessment_prop = 1 / 3,
  seed = 101L
)
split
```

`train_gazepoint_classifier` *Generic governed binary-classifier training wrapper*

Description

Generic governed binary-classifier training wrapper

Usage

```
train_gazepoint_classifier(data, task, predictors = NULL, engine = "glm", ...)
```

Arguments

`data`	Analysis data used to train the classifier.
`task`	A governed binary-classification task.
`predictors`	Optional character vector of predictor columns.
`engine`	Classification engine name or custom engine.
`...`	Additional arguments passed to `fit_gazepoint_model()`.

Value

A governed classification `gp3ml_model` object returned by `fit_gazepoint_model()`.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
```

```
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
model <- train_gazepoint_classifier(
  data = example_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  engine = "glm",
  seed = 101L
)
model
```

`validate_gazepoint_feature_manifest` *Validate a Gazepoint feature-provenance manifest*

Description

Validates the schema and declared scientific safeguards in a feature manifest. Schema errors stop execution. Substantive concerns are returned as structured `pass`, `review`, or `fail` checks.

Usage

```
validate_gazepoint_feature_manifest(x)
```

Arguments

`x`	A feature manifest created by `create_gazepoint_feature_manifest()` or a compatible data frame.

Details

A manifest fails when an intended predictor is declared as outcome-derived, post-outcome, unavailable at prediction time, or an identifier. It also fails when fold-local estimation is required but preprocessing is declared outside the resampling fold.

Unknown or incomplete provenance is returned for review rather than treated as evidence that a safeguard was satisfied.

Value

An object of class `gazepoint_feature_manifest_validation` containing the overall status, complete checks, non-passing issues, and validated manifest.

Examples

```
manifest <- create_gazepoint_feature_manifest(
  features = "fixation_duration",
  scientific_source = "Gazepoint fixation export",
  source_table = "fixations",
  transformation = "Trial-level mean",
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)

validate_gazepoint_feature_manifest(manifest)
```

---

validate_gazepoint_fold_diagnostics

Validate Gazepoint fold diagnostics

Description

Validate Gazepoint fold diagnostics

Usage

```
validate_gazepoint_fold_diagnostics(x)
```

Arguments

`x`	A `gazepoint_fold_diagnostics` object.

Value

An object of class `gazepoint_fold_diagnostics_validation`.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
```

```
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
diagnostics <- diagnose_gazepoint_group_folds(folds)
validation <- validate_gazepoint_fold_diagnostics(
  diagnostics
)
validation
```

`validate_gazepoint_group_folds` *Validate group-aware Gazepoint resampling folds*

Description

Validate group-aware Gazepoint resampling folds

Usage

```
validate_gazepoint_group_folds(x)
```

Arguments

`x`	A `gazepoint_group_folds` object.

Value

An object of class `gazepoint_group_folds_validation`.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
```

```
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
validation <- validate_gazepoint_group_folds(folds)
validation
```

validate_gazepoint_ml_roles *Validate outcome, predictor, identifier, and grouping roles*

Description

Validate outcome, predictor, identifier, and grouping roles

Usage

```
validate_gazepoint_ml_roles(data, task, predictors, feature_manifest = NULL)
```

Arguments

data	A data frame containing outcome, predictors, and identifiers.
task	A governed gp3ml_task object.
predictors	Character vector naming intended predictors.
feature_manifest	Optional Gazepoint feature-provenance manifest.

Value

A gp3ml_role_validation object containing the overall status, complete check table, non-passing issues, and optional feature-manifest validation.

Examples

```
example_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  condition = rep(c("A", "B"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
example_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = example_data,
  outcome = "quality_status",
```

```
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint all-gaze export"
  ),
  source_table = c("fixations", "all_gaze"),
  transformation = c(
    "Trial-level mean",
    "Baseline-adjusted change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = c("none", "resampling_fold"),
  fold_local_required = c(FALSE, TRUE)
)

validate_gazepoint_ml_roles(
  data = example_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest
)
```

`validate_gazepoint_ml_split` *Validate a group-aware Gazepoint holdout split*

Description

Validate a group-aware Gazepoint holdout split

Usage

```
validate_gazepoint_ml_split(x)
```

Arguments

x	An object returned by `split_gazepoint_ml_data()`.

Value

An object of class `gazepoint_ml_split_validation`.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
split <- split_gazepoint_ml_data(
  data = example_data,
```

```
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  assessment_prop = 1 / 3,
  seed = 101L
)
validation <- validate_gazepoint_ml_split(split)
validation
```

`write_external_validation_report` *Write an external-validation report*

Description

Write an external-validation report

Usage

```
write_external_validation_report(report, path, overwrite = FALSE)
```

Arguments

`report`	A `gp3ml_external_validation_report` object.
`path`	Destination Markdown file path.
`overwrite`	Whether an existing file may be replaced.

Value

The destination path, returned invisibly after the Markdown report is written.

Examples

```
training_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
training_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
```

```
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = training_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
  generalization_target = "new_participants",
  positive = "review"
)
model <- train_gazepoint_classifier(
  data = training_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  engine = "glm",
  seed = 101L
)
external_data <- training_data
external_data$participant_id <- rep(
  sprintf("E%02d", 1:12),
  each = 2
)
external_data$trial_id <- sprintf("ET%02d", 1:24)
external_data$fixation_duration <-
  external_data$fixation_duration + 4
external_data$pupil_change <- cos(seq_len(24) / 4)
validation <- evaluate_external_validation(
  model = model,
  external_data = external_data,
  label = "synthetic_external",
  bootstrap = 10L,
  seed = 101L
)
report <- create_external_validation_report(validation)
output <- tempfile(fileext = ".md")
write_external_validation_report(report, output)
file.exists(output)
unlink(output)
```

---

write_gazepoint_feature_manifest_csv

Write a Gazepoint feature manifest or validation table to CSV

Description

Writes a feature manifest or one table from a validated manifest to a UTF-8 CSV file. Existing files are not replaced unless explicitly permitted.

Usage

```
write_gazepoint_feature_manifest_csv(
  x,
  file,
  table = c("manifest", "issues", "checks"),
  overwrite = FALSE,
  na = ""
)
```

Arguments

x	A gazepoint_feature_manifest, compatible data frame, or object returned by validate_gazepoint_feature_manifest().
file	A single output path ending in .csv.
table	Table to export. One of "manifest", "issues", or "checks". Plain manifest inputs support only "manifest".
overwrite	Logical. When FALSE, the default, an existing file causes an error.
na	Character value used for missing values.

Value

The normalized output path, invisibly.

Examples

```
manifest <- create_gazepoint_feature_manifest(
  features = "fixation_duration",
  scientific_source = "Gazepoint fixation export",
  source_table = "fixations",
  transformation = "Trial-level mean",
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)

output <- tempfile(fileext = ".csv")

write_gazepoint_feature_manifest_csv(
  manifest,
  output
)

unlink(output)
```

`write_gazepoint_fold_diagnostics_csv` *Write Gazepoint fold diagnostics to CSV files*

Description

Write Gazepoint fold diagnostics to CSV files

Usage

```
write_gazepoint_fold_diagnostics_csv(
  x,
  directory,
  prefix = "gazepoint_fold_diagnostics",
  tables = c("fold_metrics", "repeat_metrics", "outcome_balance", "group_balance",
   "assessment_coverage", "exclusion_summary", "validation_checks", "validation_issues"),
  overwrite = FALSE,
  na = ""
)
```

Arguments

`x`	A `gazepoint_fold_diagnostics` object.
`directory`	Output directory.
`prefix`	File-name prefix.
`tables`	Diagnostic tables to export.
`overwrite`	Whether existing files may be overwritten.
`na`	String used for missing values.

Value

A named character vector of written file paths, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
```

```
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
diagnostics <- diagnose_gazepoint_group_folds(folds)
output_directory <- tempfile()
paths <- write_gazepoint_fold_diagnostics_csv(
```

```
  x = diagnostics,
  directory = output_directory,
  tables = c("fold_metrics", "repeat_metrics")
)
paths
unlink(output_directory, recursive = TRUE)
```

`write_gazepoint_group_folds_csv` *Write group-aware resampling tables to CSV*

Description

Write group-aware resampling tables to CSV

Usage

```
write_gazepoint_group_folds_csv(
  x,
  directory,
  prefix = "gazepoint_group_folds",
  tables = c("assignments", "fold_summary", "group_counts", "group_mapping",
    "validation_checks", "validation_issues", "audit_summary", "audit_checks",
    "audit_issues"),
  include_fold_data = FALSE,
  overwrite = FALSE,
  na = ""
)
```

Arguments

`x`	A `gazepoint_group_folds` object.
`directory`	Output directory.
`prefix`	Non-empty filename prefix.
`tables`	Character vector selecting summary tables.
`include_fold_data`	Logical. Whether every materialized fold partition should also be written.
`overwrite`	Logical. Whether existing files may be replaced.
`na`	Character representation of missing values.

Value

A named character vector of normalized output paths, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
folds <- create_gazepoint_group_folds(
  data = example_data,
```

```
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  v = 3L,
  repeats = 1L,
  seed = 101L
)
output_directory <- tempfile()
paths <- write_gazepoint_group_folds_csv(
  x = folds,
  directory = output_directory,
  tables = c("fold_summary", "group_counts")
)
paths
unlink(output_directory, recursive = TRUE)
```

`write_gazepoint_ml_leakage_audit_csv` *Write a Gazepoint ML leakage-audit table to CSV*

Description

Writes one machine-readable table from a leakage-audit object to a UTF-8 CSV file. Existing files are not replaced unless explicitly permitted.

Usage

```
write_gazepoint_ml_leakage_audit_csv(
  x,
  file,
  table = c("issues", "checks", "partition_summary"),
  overwrite = FALSE,
  na = ""
)
```

Arguments

`x`	An object returned by `audit_gazepoint_ml_leakage()`.
`file`	A single output file path ending in `.csv`.
`table`	The audit table to export. One of `"issues"`, `"checks"`, or `"partition_summary"`.
`overwrite`	Logical. When `FALSE`, the default, an existing output file causes an error.
`na`	Character value used for missing values in the CSV file.

Value

The normalized output path, invisibly.

Examples

```
analysis <- data.frame(
  participant_id = c("P01", "P02"),
  trial_id = c("T01", "T02"),
  outcome = c(0, 1),
  feature = c(1.2, 1.8)
)

assessment <- data.frame(
  participant_id = c("P03", "P04"),
  trial_id = c("T03", "T04"),
  outcome = c(1, 0),
  feature = c(2.1, 2.4)
)

audit <- audit_gazepoint_ml_leakage(
  analysis = analysis,
  assessment = assessment,
  outcome = "outcome",
  predictors = "feature",
  participant_id = "participant_id",
  trial_id = "trial_id",
  generalization_target = "new_participants"
)

output <- tempfile(fileext = ".csv")

write_gazepoint_ml_leakage_audit_csv(
  audit,
  output,
  table = "checks"
)

unlink(output)
```

---

write_gazepoint_ml_split_csv

Write group-aware split tables to CSV

Description

Write group-aware split tables to CSV

Usage

```
write_gazepoint_ml_split_csv(
  x,
  directory,
  prefix = "gazepoint_ml_split",
  tables = c("analysis", "assessment", "excluded", "assignment", "summary",
    "group_counts", "checks", "issues"),
  overwrite = FALSE,
  na = ""
)
```

Arguments

x	A gazepoint_ml_split object.
directory	Output directory.
prefix	Filename prefix.
tables	Tables to export.
overwrite	Whether existing files may be replaced.
na	Character representation of missing values.

Value

A named character vector of normalized file paths, invisibly.

Examples

```
example_data <- expand.grid(
  participant_id = sprintf("P%02d", 1:6),
  stimulus_id = sprintf("S%02d", 1:4),
  repetition = 1:2,
  KEEP.OUT.ATTRS = FALSE,
  stringsAsFactors = FALSE
)
example_data$trial_id <- paste0(
  example_data$stimulus_id,
  "_T",
  example_data$repetition
)
participant_number <- as.integer(
  sub("P", "", example_data$participant_id)
)
stimulus_number <- as.integer(
  sub("S", "", example_data$stimulus_id)
)
example_data$outcome <- factor(
  ifelse(
    (participant_number + stimulus_number) %% 2L == 0L,
    "review",
    "pass"
```

```
  ),
  levels = c("pass", "review")
)
row_index <- seq_len(nrow(example_data))
example_data$fixation_duration <- 180 + row_index
example_data$pupil_change <- round(
  sin(row_index / 7),
  4
)
example_data$repetition <- NULL
manifest <- create_gazepoint_feature_manifest(
  features = c("fixation_duration", "pupil_change"),
  scientific_source = c(
    "Gazepoint fixation export",
    "Gazepoint pupil export"
  ),
  source_table = c("fixations", "pupil"),
  transformation = c(
    "Trial-level mean",
    "Trial-level change"
  ),
  availability_stage = "during_exposure",
  prediction_time_available = TRUE,
  preprocessing_scope = "none",
  fold_local_required = FALSE
)
split <- split_gazepoint_ml_data(
  data = example_data,
  outcome = "outcome",
  predictors = c("fixation_duration", "pupil_change"),
  feature_manifest = manifest,
  generalization_target = "new_participants",
  participant_id = "participant_id",
  trial_id = "trial_id",
  stimulus_id = "stimulus_id",
  assessment_prop = 1 / 3,
  seed = 101L
)
output_directory <- tempfile()
paths <- write_gazepoint_ml_split_csv(
  x = split,
  directory = output_directory,
  tables = c("summary", "group_counts")
)
paths
unlink(output_directory, recursive = TRUE)
```

---

write_gazepoint_model_card

Write a model card

Description

Write a model card

Usage

```
write_gazepoint_model_card(
  card,
  path,
  format = c("markdown", "json"),
  overwrite = FALSE
)
```

Arguments

card	A gp3ml_model_card object.
path	Destination file path.
format	Output format: Markdown or JSON.
overwrite	Whether an existing file may be replaced.

Value

The destination path, returned invisibly after the model card is written.

Examples

```
training_data <- data.frame(
  participant_id = rep(sprintf("P%02d", 1:12), each = 2),
  trial_id = sprintf("T%02d", 1:24),
  stimulus_id = rep(c("S01", "S02"), 12),
  fixation_duration = 180 + seq_len(24),
  pupil_change = sin(seq_len(24) / 3),
  stringsAsFactors = FALSE
)
training_data$quality_status <- factor(
  c(
    "pass", "review", "pass", "review", "review", "pass",
    "review", "pass", "pass", "review", "review", "pass",
    "review", "pass", "review", "pass", "pass", "review",
    "pass", "review", "review", "pass", "pass", "review"
  ),
  levels = c("pass", "review")
)
task <- declare_gazepoint_task(
  data = training_data,
  outcome = "quality_status",
  purpose = "Predict predefined recording-quality review status",
  task_type = "classification",
  unit_id = "trial_id",
  participant_id = "participant_id",
  stimulus_id = "stimulus_id",
```

```
  generalization_target = "new_participants",
  positive = "review"
)
model <- train_gazepoint_classifier(
  data = training_data,
  task = task,
  predictors = c("fixation_duration", "pupil_change"),
  engine = "glm",
  seed = 101L
)
card <- create_gazepoint_model_card(
  model = model,
  intended_use = paste(
    "Support manual review of predefined",
    "recording-quality status"
  ),
  limitations = "Synthetic example for documentation."
)
output <- tempfile(fileext = ".md")
write_gazepoint_model_card(
  card = card,
  path = output,
  format = "markdown"
)
file.exists(output)
unlink(output)
```

`write_gazepoint_reproducibility_report` *Write a reproducibility report*

Description

Write a reproducibility report

Usage

```
write_gazepoint_reproducibility_report(report, path, overwrite = FALSE)
```

Arguments

`report`	A `gp3ml_reproducibility_report` object.
`path`	Destination Markdown file path.
`overwrite`	Whether an existing file may be replaced.

Value

The destination path, returned invisibly after the reproducibility report is written.

Examples

```
example_data <- data.frame(
  trial_id = sprintf("T%02d", 1:6),
  fixation_duration = c(190, 205, 198, 214, 202, 220),
  stringsAsFactors = FALSE
)
report <- create_gazepoint_reproducibility_report(
  objects = list(
    fixation_values = example_data$fixation_duration
  ),
  data = example_data,
  seeds = list(example = 101L),
  notes = "Synthetic documentation example.",
  project_path = tempdir()
)
output <- tempfile(fileext = ".md")
write_gazepoint_reproducibility_report(report, output)
file.exists(output)
unlink(output)
```

Index