Package: uniml (via r-universe)

November 18, 2026

Type Package

Title Unified Predictive Modeling Toolkit

Version 0.5.1

Author Steve Shi [aut, cre]

Maintainer Steve Shi <19943405117@163.com>

Description A unified toolkit for binary and multiclass classification modelling that chains preprocessing, feature selection, model training and tuning, evaluation, visualization, and SHAP interpretability behind one set of objects, while keeping a strict separation between the data that fits a model and the data that judges it. Models, tasks, resampling, and tuning are 'mlr3' objects, multiple imputation is pooled with Rubin's rules, and results are exported in a layout that can be reopened outside R.

License MIT + file LICENSE

URL https://github.com/zhengyangszy/uniml

BugReports https://github.com/zhengyangszy/uniml/issues

Encoding UTF-8

Depends R (>= 4.1)

Imports mlr3, mlr3learners, mlr3pipelines, mlr3filters, mlr3fselect, mlr3tuning, mlr3measures, paradox, mlr3misc, future, future.apply, data.table, ggplot2, pROC, glmnet, Boruta, ranger, mice, shapviz, kernelshap

Suggests mlr3mbo (>= 1.0.0), DiceKriging, rgenoud, missRanger, missForest, xgboost, e1071, kknn, nnet, rpart, PRROC, ResourceSelection, dcurves, smotefamily, ROSE, correlation, car, ggsci, patchwork, igraph, ggraph, ggpubr, haven, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

VignetteBuilder knitr

NeedsCompilation no

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

Date/Publication 2026-11-17 17:50:02 UTC

RemoteUrl https://github.com/cran/uniml

RemoteRef HEAD

RemoteSha 986a5edb5664ec4d9f2d71291ab24a08950dbe69

Contents

apply_balance . . . 4
apply_encode . . . 5
apply_imputer . . . 6
apply_preprocessor . . . 6
apply_scale . . . 7
apply_step . . . 8
as_uniml_binary . . . 9
as_uniml_data . . . 9
as_uniml_multiclass . . . 10
assemble_result . . . 11
balance_classes . . . 12
calc_ci . . . 13
calc_metrics.uniml_multiclass . . . 15
calc_nri_idi . . . 17
calibrate_probabilities.matrix . . . 18
calibrate_probabilities.uniml_multiclass . . . 20
calibration_metrics . . . 22
combine_selections . . . 23
dca . . . 23
dca.numeric . . . 24
delong_test . . . 25
detect_var_types . . . 26
diagnose_missing . . . 27
encode_categorical . . . 27
explain_shap.Learner . . . 28
export_results . . . 30
extract_thresholds . . . 31
find_threshold . . . 32
fit_balance . . . 33
fit_encode . . . 34
fit_imputer . . . 35
fit_preprocessor . . . 36
fit_scale . . . 37
get_task_type . . . 38
hl_test . . . 38
impute_data . . . 39
list_task_types . . . 40

make_split . . . 40
make_task . . . 41
multiclass_calibration . . . 42
new_uniml_result . . . 43
plot.uniml_result . . . 44
plot_calibration.uniml_multiclass . . . 44
plot_confusion.uniml_multiclass . . . 46
plot_dca.uniml_multiclass . . . 48
plot_pr_curve.uniml_result . . . 49
plot_roc.uniml_multiclass . . . 51
plot_shap_beeswarm.uniml_result . . . 53
plot_shap_dependence.uniml_result . . . 54
plot_shap_force.uniml_result . . . 56
plot_shap_importance.uniml_result . . . 57
plot_shap_waterfall.uniml_result . . . 59
pool_mi . . . 61
predict_uniml . . . 62
print.uniml_calibration . . . 62
print.uniml_data . . . 63
print.uniml_preprocessor . . . 64
print.uniml_result . . . 64
print.uniml_shap . . . 65
print.uniml_spec . . . 65
print.uniml_split . . . 66
print.uniml_task_type . . . 66
rank_models . . . 67
register_learner . . . 68
register_task_type . . . 68
register_workflow_adapter . . . 69
remove_collinear . . . 70
resolve_learners . . . 71
run_workflow . . . 72
save_result . . . 73
scale_numeric . . . 74
select_boruta . . . 75
select_features . . . 76
select_lasso . . . 77
select_multivariate . . . 78
select_rf_importance . . . 79
select_rfe . . . 80
select_univariate . . . 81
set_evaluation . . . 82
set_explain . . . 83
set_feature_selection . . . 84
set_labels . . . 85
set_leakage_policy . . . 86
set_learners . . . 86
set_output . . . 87

set_preprocess 88
set_tuning 89
summary.uniml_result 90
task_type_spec 91
theme_uniml 92
train_model 93
tripod_ai_check 94
tune_model 95
uniml_spec 96
validate_uniml_data 97
validate_uniml_preprocessor 98
validate_uniml_result 98
validate_uniml_spec 99
validate_uniml_split 99
validate_uniml_task_type 100
workflow_adapter 101
workflow_engine 101

Index **103**

`apply_balance` *Apply a fitted class-balancing step*

Description

Balancing never transforms validation or test data, so this method always returns the fitted data unchanged.

Usage

```
apply_balance(object, newdata, ...)

## Default S3 method:
apply_balance(object, newdata, ...)

## S3 method for class 'uniml_step_balance'
apply_balance(object, newdata, ...)
```

Arguments

`object`	A fitted `uniml_step_balance` object.
`newdata`	A data frame or `uniml_data` object.
`...`	Reserved for future use.

Value

The data as supplied by `newdata`, unchanged.

Examples

```
train <- data.frame(x = c(1, 2, 3, 4), y = c(0, 0, 0, 1))
step <- fit_balance(train, outcome = "y")
apply_balance(step, train)
```

`apply_encode` *Apply a fitted encoding step*

Description

Apply a fitted encoding step

Usage

```
apply_encode(object, newdata, ...)

## Default S3 method:
apply_encode(object, newdata, ...)

## S3 method for class 'uniml_step_encode'
apply_encode(object, newdata, ...)
```

Arguments

`object`	A fitted `uniml_step_encode` object.
`newdata`	A data frame or `uniml_data` object.
`...`	Reserved for future use.

Value

The encoded data, matching the class of `newdata`.

Examples

```
train <- data.frame(x = factor(c("a", "b", "a")), y = c(0, 1, 0))
step <- fit_encode(train, outcome = "y")
apply_encode(step, train)
```

apply_imputer *Apply an imputation step*

Description

S3 generic dispatching on the fitted step; applied by `apply_preprocessor()`.

Usage

```
apply_imputer(object, newdata, ...)

## Default S3 method:
apply_imputer(object, newdata, ...)

## S3 method for class 'uniml_step_impute'
apply_imputer(object, newdata, ...)
```

Arguments

`object`	A fitted `uniml_step_impute` object.
`newdata`	A data frame or `uniml_data` object.
`...`	Reserved for future use.

Value

The imputed data, matching the class of `newdata`.

Examples

```
train <- data.frame(x = c(1, NA, 3), y = c(0, 1, 0))
step <- fit_imputer(train, outcome = "y")
apply_imputer(step, train)
```

apply_preprocessor *Apply a fitted preprocessing pipeline*

Description

Apply a fitted preprocessing pipeline

Usage

```
apply_preprocessor(preprocessor, newdata, ...)
```

Arguments

preprocessor	A fitted uniml_preprocessor object.
newdata	A data frame or uniml_data object to transform.
...	Reserved for future use.

Value

The transformed data, matching the class of newdata.

Examples

```
train <- data.frame(x = c(1, NA, 3), y = c(0, 1, 0))
pre <- fit_preprocessor(train, outcome = "y")
apply_preprocessor(pre, train)
```

---

apply_scale *Apply a fitted scaling step*

Description

Apply a fitted scaling step

Usage

```
apply_scale(object, newdata, ...)

## Default S3 method:
apply_scale(object, newdata, ...)

## S3 method for class 'uniml_step_scale'
apply_scale(object, newdata, ...)
```

Arguments

object	A fitted uniml_step_scale object.
newdata	A data frame or uniml_data object.
...	Reserved for future use.

Value

The scaled data, matching the class of newdata.

Examples

```
train <- data.frame(x = c(1, 2, 3), y = c(0, 1, 0))
step <- fit_scale(train, outcome = "y")
apply_scale(step, train)
```

apply_step *Apply a single preprocessing step*

Description

Apply a single preprocessing step

Usage

```
apply_step(step, newdata, ...)

## Default S3 method:
apply_step(step, newdata, ...)

## S3 method for class 'uniml_step_impute'
apply_step(step, newdata, ...)

## S3 method for class 'uniml_step_scale'
apply_step(step, newdata, ...)

## S3 method for class 'uniml_step_encode'
apply_step(step, newdata, ...)

## S3 method for class 'uniml_step_balance'
apply_step(step, newdata, ...)
```

Arguments

step	A uniml_step object.
newdata	A data frame or uniml_data object.
...	Method-specific arguments.

Value

The transformed data.

Examples

```
train <- data.frame(x = c(1, NA, 3), y = c(0, 1, 0))
pre <- fit_preprocessor(train, outcome = "y")
apply_step(pre$steps$impute, train)
```

as_uniml_binary *Convert an mlr3 prediction into a* uniml_binary *object*

Description

The plotting and evaluation helpers accept a uniml_binary object. This converter wraps an mlr3::PredictionClassif so that a prediction produced by a plain mlr3 learner can be plugged into the same functions as a workflow result.

Usage

```
as_uniml_binary(x, positive = NULL)
```

Arguments

x	An mlr3::PredictionClassif with predict_type = "prob".
positive	Positive class label. Defaults to the positive class of the mlr3 task the prediction was made for, or to the first probability column when the prediction does not carry its task.

Value

A uniml_binary object accepted by the uniml plotting functions.

Examples

```
data <- data.frame(
  y = factor(rep(c("no", "yes"), 20), levels = c("no", "yes")),
  x1 = stats::rnorm(40),
  x2 = stats::rnorm(40)
)
spec <- uniml_spec(task_type = "binary", outcome = "y", positive = "yes")
task <- make_task(data, spec)
learner <- mlr3::lrn("classif.rpart", predict_type = "prob")
learner$train(task)
binary <- as_uniml_binary(learner$predict(task))
names(binary)
```

as_uniml_data *Convert data to a uniml data object*

Description

Convert data to a uniml data object

Usage

```
as_uniml_data(data, spec)
```

Arguments

data	A data frame.
spec	A uniml_spec object.

Value

A uniml_data object.

Examples

```
spec <- uniml_spec("binary", "y", "yes", id = "id")
data <- data.frame(
  id = 1:4,
  y = factor(c("no", "no", "yes", "yes"), levels = c("no", "yes")),
  x1 = c(0.1, 0.4, 0.6, 0.9)
)
as_uniml_data(data, spec)
```

---

as_uniml_multiclass *Convert an mlr3 prediction into a* uniml_multiclass *object*

Description

The multiclass metric and plotting helpers accept a uniml_multiclass object: a probability matrix with one column per class plus the observed labels.

Usage

```
as_uniml_multiclass(x, truth = NULL)
```

Arguments

x	An mlr3::PredictionClassif with predict_type = "prob", or a numeric probability matrix.
truth	Observed class labels. Required when x is a matrix.

Value

A uniml_multiclass object with prob, truth, and classes.

References

Hand DJ, Till RJ (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45, 171-186. doi:10.1023/A:1010920819831

Examples

```
prob <- rbind(
  c(a = 0.7, b = 0.2, c = 0.1),
  c(a = 0.2, b = 0.6, c = 0.2),
  c(a = 0.1, b = 0.3, c = 0.6)
)
multiclass <- as_uniml_multiclass(prob, truth = c("a", "b", "c"))
calc_metrics(multiclass)$accuracy
```

---

assemble_result *Assemble a complete uniml result*

Description

Combines the outputs of the modelling stages into a `uniml_result`. Missing stages are preserved as `NULL`; ranking is attempted whenever metric results are available.

Usage

```
assemble_result(
  data_info = NULL,
  preprocess = NULL,
  selection = NULL,
  tuning = NULL,
  cv = NULL,
  metrics = NULL,
  ci = NULL,
  calibration = NULL,
  tests = NULL,
  dca = NULL,
  ranking = NULL,
  shap = NULL,
  thresholds = NULL,
  plots = NULL,
  models = NULL,
  meta = list(),
  select_on = "auto",
  best_model_metric = "composite"
)
```

Arguments

`data_info`	Optional data information, usually a `uniml_data` object.
`preprocess`	Optional fitted preprocessor.
`selection`	Optional feature-selection result.
`tuning`	Optional tuning result or list of tuning results.

cv	Optional cross-validation result.
metrics	Optional metric data frame.
ci	Optional confidence-interval table.
calibration	Optional calibration result or list.
tests	Optional post-model test results.
dca	Optional decision-curve table.
ranking	Optional precomputed ranking.
shap	Optional SHAP results.
thresholds	Optional threshold tables.
plots	Optional plot object list.
models	Optional final model list.
meta	Optional metadata list.
select_on	Comparison set used for ranking.
best_model_metric	Metric used for ranking.

Value

A uniml_result.

Examples

```
result <- assemble_result(
  data_info = list(n_train = 100L, outcome = "y"),
  metrics = data.frame(
    model = c("lr", "cart"),
    set = "validation",
    probability = "raw",
    auc = c(0.82, 0.78)
  )
)
result
```

---

balance_classes *Balance class frequencies*

Description

Convenience wrapper that applies the fixed preprocessing order and returns the balanced training data.

Usage

```
balance_classes(
  data,
  method = c("none", "rose", "smote", "both"),
  outcome = NULL,
  ...
)
```

Arguments

data	A data frame or uniml_data object.
method	Balancing method.
outcome	Outcome column name.
...	Arguments passed to fit_preprocessor().

Value

The balanced data, matching the class of data.

References

Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002). SMOTE: synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16, 321-357. doi:10.1613/jair.953

Menardi G, Torelli N (2014). Training and assessing classification rules with imbalanced data. *Data Mining and Knowledge Discovery*, 28(1), 92-122. doi:10.1007/s10618-012-0295-5

van den Goorbergh R, van Smeden M, Timmerman D, Van Calster B (2022). The harm of class imbalance corrections for risk prediction models: illustration and simulation using logistic regression. *Journal of the American Medical Informatics Association*, 29(9), 1525-1534. doi:10.1093/jamia/ocac093

Examples

```
balance_classes(
  data.frame(x = c(1, 2, 3, 4), y = factor(c(0, 0, 0, 1))),
  method = "none", outcome = "y"
)
```

---

calc_ci *Bootstrap confidence intervals for binary metrics*

Description

Resamples paired probabilities and labels with replacement, recalculates the requested binary metrics, and returns percentile confidence intervals.

Usage

```
calc_ci(
  probs,
  labels,
  method = "bootstrap",
  R = 1000L,
  level = 0.95,
  seed = 123L,
  threshold = 0.5,
  threshold_method = "fixed",
  cost = c(fp = 1, fn = 1),
  fixed_threshold = 0.5,
  positive = NULL,
  metrics = "all",
  ...
)
```

Arguments

probs	Numeric predicted probabilities, or a numeric matrix with one column per class for a multiclass prediction. A matrix switches to the multiclass metrics; threshold, threshold_method, cost, and fixed_threshold are then ignored.
labels	Binary truth labels, or the observed classes when probs is a matrix.
method	Confidence-interval method. Currently "bootstrap".
R	Number of bootstrap resamples.
level	Confidence level.
seed	Random seed.
threshold	Classification threshold.
threshold_method	Threshold method used when threshold = NULL.
cost	Named false-positive and false-negative cost vector.
fixed_threshold	Fixed threshold used by the "fixed" method.
positive	Positive class label.
metrics	"all" or a character vector of metric names.
...	Reserved for method-specific arguments.

Value

A data frame with columns metric, mean, se (bootstrap standard error), lower, upper, n_effective, and formatted.

References

Efron B, Tibshirani RJ (1993). *An Introduction to the Bootstrap*. Chapman & Hall. doi:10.1201/9780429246593

Examples

```
calc_ci(
  c(0.1, 0.4, 0.6, 0.9),
  c(0, 0, 1, 1),
  R = 20,
  threshold = 0.5,
  seed = 1
)
```

`calc_metrics.uniml_multiclass` *Calculate binary classification metrics*

Description

Calculates the complete binary metric set used by `uniml`. `x` can be a numeric probability vector, an mlr3 `PredictionClassif`, or a `uniml_binary` prediction object with `prob`, `truth`, and optional `positive` components.

S3 generic that dispatches metric calculation by task type. Method bodies are added in later milestones.

Usage

```
## S3 method for class 'uniml_multiclass'
calc_metrics(x, truth = NULL, metrics = "all", ...)

## S3 method for class 'matrix'
calc_metrics(x, truth = NULL, metrics = "all", ...)

## S3 method for class 'uniml_binary'
calc_metrics(
  x,
  truth = NULL,
  threshold = NULL,
  positive = NULL,
  metrics = "all",
  threshold_method = "fixed",
  cost = c(fp = 1, fn = 1),
  fixed_threshold = 0.5,
  ...
)

## S3 method for class 'data.frame'
calc_metrics(
  x,
  truth = NULL,
  threshold = NULL,
```

```
  positive = NULL,
  metrics = "all",
  probability = c("raw", "calibrated"),
  threshold_method = "fixed",
  cost = c(fp = 1, fn = 1),
  fixed_threshold = 0.5,
  ...
)

## S3 method for class 'numeric'
calc_metrics(
  x,
  truth = NULL,
  threshold = NULL,
  positive = NULL,
  metrics = "all",
  threshold_method = "fixed",
  cost = c(fp = 1, fn = 1),
  fixed_threshold = 0.5,
  ...
)

## S3 method for class 'PredictionClassif'
calc_metrics(
  x,
  threshold = NULL,
  positive = NULL,
  metrics = "all",
  threshold_method = "fixed",
  cost = c(fp = 1, fn = 1),
  fixed_threshold = 0.5,
  ...
)

calc_metrics(x, ...)

## Default S3 method:
calc_metrics(x, ...)
```

Arguments

x	An object whose class identifies the task type, such as "uniml_binary".
truth	Optional binary truth vector.
metrics	"all" or a character vector of metric names.
...	Method-specific arguments.
threshold	Optional fixed classification threshold.
positive	Positive class label. For an mlr3 PredictionClassif the default is the positive

class of the task the prediction was made for, falling back to the first probability column when the prediction does not carry its task.

`threshold_method`
: Threshold method used when `threshold` is `NULL`.

`cost`
: Named false-positive and false-negative cost vector.

`fixed_threshold`
: Threshold used by the `"fixed"` method.

`probability`
: Which probability column of a prediction table to score: `"raw"` or `"calibrated"`.

Value

A named list of metrics and confusion-matrix counts.

The method return value.

References

Steyerberg EW (2019). *Clinical Prediction Models: A Practical Approach to Development, Validation, and Updating*, 2nd edition. Springer. doi:10.1007/978-3-030-16399-0

Robin X, Turck N, Hainard A, et al. (2011). pROC: an open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics*, 12, 77. doi:10.1186/1471-2105-12-77

Examples

```
calc_metrics(
  c(0.2, 0.8, 0.6, 0.3),
  c(0, 1, 1, 0),
  threshold = 0.5
)
```

---

calc_nri_idi *Calculate continuous or categorical NRI and IDI*

Description

Calculate continuous or categorical NRI and IDI

Usage

```
calc_nri_idi(
  pred_new,
  pred_reference,
  truth,
  method = c("continuous", "categorical"),
  threshold_new = 0.5,
  threshold_reference = 0.5,
  positive = NULL
)
```

Arguments

`pred_new`	New-model predicted probabilities.
`pred_reference`	Reference-model predicted probabilities.
`truth`	Binary truth labels.
`method`	`"continuous"` or `"categorical"` NRI.
`threshold_new, threshold_reference`	Thresholds used for categorical NRI.
`positive`	Positive class label.

Value

A list with NRI, IDI, group-specific NRI components, and sample counts. Metrics are `NA` when the required event or non-event group is absent.

References

Pencina MJ, D'Agostino RB Sr, D'Agostino RB Jr, Vasan RS (2008). Evaluating the added predictive ability of a new marker: from area under the ROC curve to reclassification and beyond. *Statistics in Medicine*, 27(2), 157-172. doi:10.1002/sim.2929

Pencina MJ, D'Agostino RB Sr, Steyerberg EW (2011). Extensions of net reclassification improvement calculations to measure usefulness of new biomarkers. *Statistics in Medicine*, 30(1), 11-21. doi:10.1002/sim.4067

Examples

```
calc_nri_idi(
  pred_new = c(0.2, 0.7, 0.8, 0.1),
  pred_reference = c(0.3, 0.6, 0.7, 0.2),
  truth = c(0, 1, 1, 0)
)
```

`calibrate_probabilities.matrix`

Calibrate predicted probabilities

Description

S3 generic that dispatches probability calibration by task type. A binary outcome uses a one-vs-none recalibration (`"platt"`, `"platt_segment"`, `"isotonic"`); a multiclass outcome uses temperature scaling, because a multinomial model needs one scalar applied to every class rather than `K` independent binary calibrators. See `calibrate_probabilities.uniml_multiclass()` for the multiclass conventions.

Usage

```
## S3 method for class 'matrix'
calibrate_probabilities(x, labels = NULL, ...)

## S3 method for class 'numeric'
calibrate_probabilities(
  x,
  labels = NULL,
  method = c("platt", "platt_segment", "isotonic", "none"),
  folds = 5L,
  seed = 123L,
  ...
)

calibrate_probabilities(x, ...)

## Default S3 method:
calibrate_probabilities(x, ...)

## S3 method for class 'uniml_binary'
calibrate_probabilities(x, ...)
```

Arguments

x	An object whose class identifies the task type, such as "uniml_binary" or a probability matrix with three or more class columns.
labels	Binary truth labels. Required by the numeric method.
...	Method-specific arguments.
method	Calibration method: "platt", "platt_segment", "isotonic", or "none".
folds	Number of cross-fitting folds.
seed	Random seed used for cross-fitting.

Value

The method return value.

References

Platt J (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In: *Advances in Large Margin Classifiers*. MIT Press.

Zadrozny B, Elkan C (2002). Transforming classifier scores into accurate multiclass probability estimates. *Proceedings of the 8th ACM SIGKDD*, 694-699. doi:10.1145/775047.775109

Niculescu-Mizil A, Caruana R (2005). Predicting good probabilities with supervised learning. *Proceedings of the 22nd ICML*, 625-632. doi:10.1145/1102351.1102430

Van Calster B, McLernon DJ, van Smeden M, et al. (2019). Calibration: the Achilles heel of predictive analytics. *BMC Medicine*, 17, 230. doi:10.1186/s12916-019-1466-7

Examples

```
probs <- c(0.05, 0.20, 0.35, 0.55, 0.70, 0.90)
labels <- c(0, 0, 0, 1, 1, 1)
calibrated <- calibrate_probabilities(probs, labels = labels, method = "platt")
calibrated$method
calibrated$calibrated
```

`calibrate_probabilities.uniml_multiclass`
Calibrate multiclass probabilities by temperature scaling

Description

Fits the single scalar `T` that makes a multiclass model's reported probabilities match how often it is right. The recalibrated probabilities are `softmax(log(p) / T)`: `T > 1` softens an over-confident model, `T < 1` sharpens an under-confident one, and `T = 1` leaves the predictions alone.

Usage

```
## S3 method for class 'uniml_multiclass'
calibrate_probabilities(
  x,
  labels = NULL,
  new_probs = NULL,
  new_labels = NULL,
  method = c("temperature", "none"),
  folds = 5L,
  seed = 123L,
  bins = 10L,
  ...
)
```

Arguments

`x`	A `uniml_multiclass` prediction, or a probability matrix with `labels` supplied.
`labels`	Observed class labels. Required when `x` is a matrix.
`new_probs`	Optional probability matrix to recalibrate with the scalar fitted on `x` (the validation-to-test handover). Its columns must be the same classes, in the same order.
`new_labels`	Optional labels for `new_probs`, used to report the before/after table on that set.
`method`	`"temperature"` or `"none"`.
`folds`	Number of cross-fitting folds.
`seed`	Random seed for the cross-fitting.
`bins`	Number of reliability bins used in the before/after table.
`...`	Reserved for method-specific arguments.

Details

Because the division is monotone and is applied to every class in the same way, temperature scaling **cannot change which class the model ranks first**. The confusion matrix, the accuracy, and every threshold-based metric come out identical; only the probabilities attached to them move. This is the reason a one-parameter method is worth having: it cannot make the model worse at deciding, only better at reporting.

`T` is fitted by minimising the log loss on the observed labels, and `T = 1` is kept whenever it is at least as good, so the fitted scalar never degrades the probabilities it was fitted on. The returned `calibrated` matrix comes from cross-fitting (each fold's temperature is estimated on the other folds), so the before/after comparison is not measured on the data the scalar was fitted on. That honesty has a cost: on a model that was already calibrated, the cross-fitted log loss can come out a hair worse than the raw probabilities, because each fold's scalar is a noisy estimate of 1. The guarantee is about the scalar, not about the fold-to-fold noise.

Dirichlet calibration (Kull et al. 2019) is the stronger method when there are enough patients, but it estimates `K^2 + K` parameters and can change the predicted class; on the few hundred validation patients this package targets those parameters are not identifiable. Temperature scaling is the part of the idea that a sample of that size can support.

Value

A `uniml_calibration` object with `method`, `calibrated` (a matrix), `raw`, `parameters` (including `fitted$temperature`), `metrics_before`, `metrics_after`, `failed`, `message`, and `n`.

References

Guo C, Pleiss G, Sun Y, Weinberger KQ (2017). On calibration of modern neural networks. *Proceedings of the 34th International Conference on Machine Learning*, 70, 1321-1330.

Kull M, Perello-Nieto M, Kängsepp M, Silva Filho T, Song H, Flach P (2019). Beyond temperature scaling: obtaining well-calibrated multi-class probabilities with Dirichlet calibration. *Advances in Neural Information Processing Systems*, 32.

Van Calster B, McLernon DJ, van Smeden M, Wynants L, Steyerberg EW (2019). Calibration: the Achilles heel of predictive analytics. *BMC Medicine*, 17, 230. doi:10.1186/s12916-019-1466-7

Examples

```
# A multinomial model that is deliberately too confident: the reported
# probabilities are the truth raised to the power 3, so the scalar that undoes
# the damage is about 3.
set.seed(1)
truth <- matrix(stats::runif(180), nrow = 60)
truth <- truth / rowSums(truth)
colnames(truth) <- c("a", "b", "c")
labels <- apply(truth, 1L, function(p) sample(colnames(truth), 1L, prob = p))
reported <- truth^3
reported <- reported / rowSums(reported)
calibrated <- calibrate_probabilities(reported, labels = labels, folds = 2L)
round(calibrated$parameters$fitted$temperature, 2)
calibrated$metrics_before[, c("accuracy", "ece", "logloss")]
calibrated$metrics_after[, c("accuracy", "ece", "logloss")]
```

`calibration_metrics` *Calibration intercept and slope*

Description

The two numbers a clinical reader looks for after the calibration plot. Both come from one logistic recalibration of the predictions, `logit(observed) = intercept + slope * logit(predicted)`, which is the convention of Steyerberg and of van Calster's calibration guidance: an ideal model has slope 1, and the intercept is 0. Calibration-in-the-large is reported separately, as the intercept of the same model with the slope fixed at 1, so a systematic over- or under-prediction is not hidden inside a slope away from 1.

Usage

```
calibration_metrics(
  probs,
  labels,
  positive = NULL,
  level = 0.95,
  R = 500L,
  seed = 2026L
)
```

Arguments

`probs`	Predicted probabilities of the positive class.
`labels`	Observed outcomes.
`positive`	Value of `labels` that means "event".
`level`	Confidence level for the interval, or `NULL` for no interval.
`R`	Bootstrap resamples used for the interval.
`seed`	Random seed for the bootstrap.

Value

A one-row data frame with `calib_intercept`, `calib_slope`, `calib_large`, their `lower`/`upper` bounds when `level` is not `NULL`, and `n`.

References

Steyerberg EW (2019). *Clinical Prediction Models: A Practical Approach to Development, Validation, and Updating*, 2nd edition. Springer. doi:10.1007/978-3-030-16399-0

Van Calster B, McLernon DJ, van Smeden M, Wynants L, Steyerberg EW (2019). Calibration: the Achilles heel of predictive analytics. *BMC Medicine*, 17, 230. doi:10.1186/s12916-019-1466-7

Cox DR (1958). Two further applications of a model for binary regression. *Biometrika*, 45(3-4), 562-565.

Examples

```
set.seed(1)
probs <- stats::runif(200)
labels <- stats::rbinom(200, 1, probs)
calibration_metrics(probs, labels, level = NULL)
```

---

combine_selections *Combine feature-selection results*

Description

Combines a list of selected feature vectors or selection result objects.

Usage

```
combine_selections(selections, how = c("intersect", "union"))
```

Arguments

selections	A non-empty list of character vectors or selection results.
how	Combination rule, "intersect" or "union".

Value

A sorted character vector of selected feature names.

Examples

```
combine_selections(list(c("x1", "x2"), c("x2", "x3")), how = "intersect")
```

---

dca *Decision-curve analysis*

Description

S3 generic that dispatches decision-curve analysis by task type. Method bodies are added in later milestones.

Usage

```
dca(x, ...)

## Default S3 method:
dca(x, ...)

## S3 method for class 'uniml_binary'
dca(x, ...)
```

Arguments

x	An object whose class identifies the task type, such as "uniml_binary".
...	Method-specific arguments.

Value

The method return value.

References

Vickers AJ, Elkin EB (2006). Decision curve analysis: a novel method for evaluating prediction models. *Medical Decision Making*, 26(6), 565-574. doi:10.1177/0272989X06295361

Vickers AJ, van Calster B, Steyerberg EW (2016). Net benefit approaches to the evaluation of prediction models, molecular markers, and diagnostic tests. *BMJ*, 352, i6. doi:10.1136/bmj.i6

Examples

```
probs <- c(0.05, 0.20, 0.35, 0.55, 0.70, 0.90)
labels <- c(0, 0, 0, 1, 1, 1)
curve <- dca(probs, labels = labels)
head(curve)
```

---

dca.numeric *Decision-curve analysis*

Description

Decision-curve analysis

Usage

```
## S3 method for class 'numeric'
dca(
  x,
  labels = NULL,
  thresholds = NULL,
  positive = NULL,
  model = "model",
  calibrated = FALSE,
  ...
)
```

Arguments

x	Predicted probabilities.
labels	Binary truth labels.
thresholds	Decision thresholds in [0, 1).
positive	Positive class label.
model	Model name used in the returned data frame.
calibrated	Whether the probabilities are calibrated.
...	Reserved for method-specific arguments.

Value

A data frame of class uniml_dca containing model and baseline net benefits.

Examples

```
dca(
  c(0.1, 0.4, 0.6, 0.9),
  c(0, 0, 1, 1),
  thresholds = c(0.2, 0.5, 0.8)
)
```

---

delong_test *Compare binary model AUCs with DeLong tests*

Description

Compare binary model AUCs with DeLong tests

Usage

```
delong_test(pred_list, labels, positive = NULL)
```

Arguments

pred_list	Named list of predicted-probability vectors.
labels	Binary truth labels.
positive	Positive class label.

Value

A list containing AUC values, the best model, a pairwise p-value matrix, a formatted matrix, and a summary data frame. Fewer than two models returns a skipped result without an error.

References

DeLong ER, DeLong DM, Clarke-Pearson DL (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, 44(3), 837-845. doi:10.2307/2531595

Examples

```
delong_test(
  list(a = c(0.1, 0.7, 0.8, 0.2), b = c(0.2, 0.6, 0.7, 0.3)),
  c(0, 1, 1, 0)
)
```

`detect_var_types` *Detect feature variable types*

Description

Detect feature variable types

Usage

```
detect_var_types(data, outcome = NULL, ordered_as = c("ordinal", "numeric"))
```

Arguments

`data`	A data frame.
`outcome`	Outcome column name, excluded from the result.
`ordered_as`	How ordered factors are handled, `"ordinal"` or `"numeric"`.

Value

A data frame with `column`, `type`, and `n_levels`.

Examples

```
detect_var_types(iris)
```

diagnose_missing *Diagnose missing values*

Description

Diagnose missing values

Usage

```
diagnose_missing(data)
```

Arguments

data	A data frame.

Value

A list with per_column (missing count and share) and overall.

Examples

```
diagnose_missing(data.frame(a = c(1, NA), b = c("x", "y")))
```

encode_categorical *Encode categorical variables*

Description

Convenience wrapper returning the encoded data.

Usage

```
encode_categorical(
  data,
  method = c("onehot", "onehot_drop", "none"),
  outcome = NULL,
  ...
)
```

Arguments

data	A data frame or uniml_data object.
method	Encoding method.
outcome	Outcome column name.
...	Arguments passed to fit_preprocessor().

Value

The encoded data, matching the class of `data`.

Examples

```
encode_categorical(
  data.frame(x = factor(c("a", "b", "a")), y = c(0, 1, 0)),
  outcome = "y"
)
```

`explain_shap.Learner` *Compute SHAP explanations*

Description

S3 generic that dispatches SHAP computation by task type. Method bodies are added in later milestones.

Usage

```
## S3 method for class 'Learner'
explain_shap(
  x,
  data = NULL,
  raw_data = NULL,
  feature_names = NULL,
  n_explain = 100L,
  bg_n = 100L,
  seed = 123L,
  positive = NULL,
  parallel = FALSE,
  task_type = "binary",
  ...
)

## S3 method for class 'list'
explain_shap(
  x,
  data = NULL,
  raw_data = NULL,
  feature_names = NULL,
  n_explain = 100L,
  bg_n = 100L,
  seed = 123L,
  positive = NULL,
  parallel = FALSE,
  task_type = "binary",
```

```
  ...
)

## S3 method for class 'uniml_result'
explain_shap(
  x,
  data = NULL,
  raw_data = NULL,
  feature_names = NULL,
  model = "all",
  n_explain = 100L,
  bg_n = 100L,
  seed = 123L,
  positive = NULL,
  parallel = FALSE,
  task_type = "binary",
  ...
)

explain_shap(x, ...)

## Default S3 method:
explain_shap(x, ...)

## S3 method for class 'uniml_binary'
explain_shap(x, ...)
```

Arguments

x	An object whose class identifies the task type, such as "uniml_binary".
data	Feature data used for SHAP computation.
raw_data	Optional raw feature data aligned with data.
feature_names	Optional feature subset.
n_explain	Number of rows to explain. Larger values are subsampled with seed, because the cost of Kernel SHAP grows linearly with the number of explained rows.
bg_n	Number of background rows used to integrate out inactive features.
seed	Random seed.
positive	Positive class label.
parallel	Whether kernelshap may loop over rows in parallel with a registered future backend.
task_type	Task type. Only "binary" is implemented.
...	Method-specific arguments.
model	For a uniml_result, which models to explain: "best", "all", or one or more model names.

Value

The method return value.

References

Lundberg SM, Lee SI (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30.

Lundberg SM, Erion G, Chen H, et al. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1), 56-67. doi:10.1038/s42256-019-0138-9

Covert I, Lee SI (2021). Improving KernelSHAP: practical Shapley value estimation using linear regression. *Proceedings of AISTATS*, PMLR 130, 3457-3465.

Molnar C (2022). *Interpretable Machine Learning*, 2nd edition. https://christophm.github.io/interpretable-ml-book/

Examples

```
## Not run:
result <- run_workflow(spec, train, validation)
explained <- explain_shap(result, model = "best", n_explain = 100)
plot_shap_importance(explained)

## End(Not run)
```

---

export_results *Export a uniml result*

Description

Writes the complete result and all available stage tables using the directory layout defined by the package specification. Missing stages are skipped rather than treated as errors.

Usage

```
export_results(
  result,
  dir = "output",
  include_plots = TRUE,
  plot_args = list()
)
```

Arguments

`result`	A `uniml_result` object.
`dir`	Output directory.
`include_plots`	Whether to save available ggplot objects as image files.
`plot_args`	Optional list of `ggplot2::ggsave()` arguments (`width`, `height`, `dpi`, `units`, `device`, ...).

Value

Invisibly returns a list with the output directory, written files, and skipped stages.

Examples

```
## Not run:
result <- run_workflow(spec, train, validation)
exported <- export_results(result, dir = tempfile("uniml-"))
basename(exported$files)

## End(Not run)
```

`extract_thresholds` *Extract SHAP-zero clinical thresholds*

Description

Fits a LOESS curve to the SHAP value of `target` and locates zero crossings, optionally within strata of an interaction variable.

Usage

```
extract_thresholds(
  shap_result,
  target,
  interaction = NULL,
  min_n = 30L,
  min_unique = 10L,
  span = 0.75,
  main_grid = 1000L,
  stratum_grid = 500L
)
```

Arguments

`shap_result`	A `uniml_shap` object or a list with SHAP and feature value matrices.
`target`	Continuous target feature.
`interaction`	Optional continuous or categorical interaction feature.
`min_n`	Minimum observations required in a stratum.
`min_unique`	Minimum number of unique target values.
`span`	LOESS smoothing span.
`main_grid`	Number of grid points for the main curve.
`stratum_grid`	Number of grid points for interaction strata.

Value

A list with `zero_crossings` and `paper_ready`.

Examples

```
shap <- list(
  values_matrix_raw = matrix(seq(-1, 1, length.out = 40),
    ncol = 1L, dimnames = list(NULL, "x")
  ),
  feature_values_raw = data.frame(x = seq(0, 1, length.out = 40))
)
extract_thresholds(shap, target = "x")
```

`find_threshold` *Find a binary classification threshold*

Description

Find a binary classification threshold

Usage

```
find_threshold(
  probs,
  labels,
  method = c("youden", "cost", "fixed"),
  cost = c(fp = 1, fn = 1),
  fixed_threshold = 0.5,
  positive = NULL
)
```

Arguments

`probs`	Numeric predicted probabilities.
`labels`	Binary truth labels.
`method`	One of `"youden"`, `"cost"`, or `"fixed"`.
`cost`	Named false-positive and false-negative costs.
`fixed_threshold`	Fixed threshold used by the `"fixed"` method.
`positive`	Positive class label.

Value

A single numeric threshold, or `NA_real_` when it cannot be estimated from the supplied classes.

References

Youden WJ (1950). Index for rating diagnostic tests. *Cancer*, 3(1), 32-35.

Examples

```
find_threshold(c(0.1, 0.4, 0.6, 0.9), c(0, 0, 1, 1), method = "youden")
```

---

fit_balance *Fit a class-balancing step*

Description

Balancing is applied to the training data only; validation and test data are never resampled. Supported methods are `"none"`, `"rose"`, `"smote"`, and `"both"`. A failed or inapplicable resampling falls back to the input data with a warning.

Usage

```
fit_balance(
  data,
  method = c("none", "rose", "smote", "both"),
  outcome = NULL,
  p = 0.5,
  n_neighbors = 5L,
  dup_size = 0,
  seed = 123,
  verbose = 1
)
```

Arguments

`data`	A data frame.
`method`	Balancing method.
`outcome`	Outcome column name.
`p`	ROSE target proportion of the minority class.
`n_neighbors`	SMOTE number of neighbours.
`dup_size`	SMOTE duplication size; `0` balances the classes exactly.
`seed`	Random seed.
`verbose`	Verbosity level.

Value

A fitted `uniml_step_balance` step.

Examples

```
fit_balance(data.frame(x = c(1, 2, 3, 4), y = factor(c(0, 0, 0, 1))),
  outcome = "y", method = "none")
```

`fit_encode` *Fit a categorical encoding step*

Description

Builds dummy-variable mappings from training data. Binary categorical columns become a single 0/1 indicator; multi-level columns expand to one column per level (or per level minus the reference level with `"onehot_drop"`).

Usage

```
fit_encode(
  data,
  method = c("onehot", "onehot_drop", "none"),
  outcome = NULL,
  ordered_as = c("ordinal", "numeric"),
  max_levels = 50L,
  verbose = 1
)
```

Arguments

`data`	A data frame.
`method`	Encoding method, `"onehot"`, `"onehot_drop"`, or `"none"`.
`outcome`	Outcome column name, excluded from encoding.
`ordered_as`	Ordered-factor handling.
`max_levels`	Maximum levels allowed before a high-cardinality warning.
`verbose`	Verbosity level.

Value

A fitted `uniml_step_encode` step.

Examples

```
fit_encode(data.frame(x = factor(c("a", "b", "a")), y = c(0, 1, 0)),
  outcome = "y")
```

`fit_imputer` *Fit an imputer*

Description

Fits a missing-value imputer on training data. Supported methods are `"median_mode"`, `"mice"`, `"missForest"`, `"missRanger"`, and `"none"`. All-NA columns are dropped with a warning.

Usage

```
fit_imputer(
  data,
  method = c("median_mode", "mice", "missForest", "missRanger", "none"),
  outcome = NULL,
  ordered_as = c("ordinal", "numeric"),
  m = 5L,
  max_iter = 5L,
  seed = 123,
  verbose = 1
)
```

Arguments

`data`	A data frame.
`method`	Imputation method.
`outcome`	Outcome column name, excluded from imputation.
`ordered_as`	Ordered-factor handling.
`m`	Number of imputations for `mice`.
`max_iter`	Iterations passed to `mice`.
`seed`	Random seed.
`verbose`	Verbosity level.

Value

A fitted `uniml_step_impute` step.

Examples

```
fit_imputer(data.frame(x = c(1, NA, 3), y = c(0, 1, 0)), outcome = "y")
```

`fit_preprocessor` *Fit a preprocessing pipeline*

Description

Fits imputation, scaling, encoding, and class balancing on training data in the fixed order `impute -> scale -> encode -> balance`. Fitted parameters are reused by `apply_preprocessor()` on validation or test data.

Usage

```
fit_preprocessor(
  data,
  spec = NULL,
  impute = NULL,
  scale = NULL,
  encode = NULL,
  balance = NULL,
  m = NULL,
  p = 0.5,
  n_neighbors = 5L,
  dup_size = 0,
  vars = NULL,
  outcome = NULL,
  ordered_as = NULL,
  max_iter = 5,
  seed = 123,
  verbose = 1
)
```

Arguments

`data`	A data frame or `uniml_data` object.
`spec`	Optional `uniml_spec`; preprocessing options are read from it.
`impute, scale, encode, balance`	Optional method overrides.
`m`	Number of imputations for `mice`.
`p`	ROSE target proportion of the minority class.
`n_neighbors`	SMOTE number of neighbours.
`dup_size`	SMOTE duplication size.
`vars`	Optional variables for scaling; defaults to type detection.
`outcome`	Outcome column name.
`ordered_as`	Ordered-factor handling.
`max_iter`	Iterations passed to `mice`.
`seed`	Random seed.
`verbose`	Verbosity level.

Value

A fitted `uniml_preprocessor` object.

Examples

```
fit_preprocessor(data.frame(x = c(1, NA, 3), y = c(0, 1, 0)), outcome = "y")
```

`fit_scale` *Fit a numeric scaling step*

Description

Computes Z-score or Min-Max scaling parameters on training data. Only continuous numeric columns are scaled; binary 0/1 numeric columns are excluded. Columns with zero variance are skipped with a warning.

Usage

```
fit_scale(
  data,
  method = c("zscore", "minmax", "none"),
  vars = NULL,
  outcome = NULL,
  ordered_as = c("ordinal", "numeric"),
  verbose = 1
)
```

Arguments

`data`	A data frame.
`method`	Scaling method, `"zscore"`, `"minmax"`, or `"none"`.
`vars`	Optional character vector restricting which columns are scaled.
`outcome`	Outcome column name, excluded from scaling.
`ordered_as`	Ordered-factor handling.
`verbose`	Verbosity level.

Value

A fitted `uniml_step_scale` step.

Examples

```
fit_scale(data.frame(x = c(1, 2, 3), y = c(0, 1, 0)), outcome = "y")
```

get_task_type *Get a task-type descriptor*

Description

Get a task-type descriptor

Usage

```
get_task_type(id)
```

Arguments

id	Task-type id.

Value

The matching unimL_task_type object.

Examples

```
get_task_type("binary")
```

hl_test *Hosmer-Lemeshow goodness-of-fit test*

Description

Uses ResourceSelection::hoslem.test() when available and falls back to a quantile-grouped implementation.

Usage

```
hl_test(probs, labels, g = 10L, positive = NULL)
```

Arguments

probs	Predicted probabilities.
labels	Binary truth labels.
g	Requested number of groups.
positive	Positive class label.

Value

A list with the test statistic, degrees of freedom, p value, group count, method, group details, and a status message.

References

Hosmer DW, Lemeshow S (1980). A goodness-of-fit test for the multiple logistic regression model. *Communications in Statistics - Theory and Methods*, 10, 1043-1069. doi:10.1080/03610928008827941

Examples

```
set.seed(1)
p <- stats::runif(80)
y <- stats::rbinom(80, 1, p)
hl_test(p, y, g = 10)
```

`impute_data` *Impute missing values*

Description

Convenience wrapper that fits an imputer and applies it to the same data.

Usage

```
impute_data(
  data,
  method = c("median_mode", "mice", "missForest", "missRanger", "none"),
  outcome = NULL,
  ...
)
```

Arguments

`data`	A data frame or `uniml_data` object.
`method`	Imputation method.
`outcome`	Outcome column name.
`...`	Arguments passed to `fit_imputer()`.

Value

The imputed data, matching the class of `data`.

References

van Buuren S, Groothuis-Oudshoorn K (2011). mice: multivariate imputation by chained equations in R. *Journal of Statistical Software*, 45(3), 1-67. doi:10.18637/jss.v045.i03

Stekhoven DJ, Buhlmann P (2012). MissForest: non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1), 112-118. doi:10.1093/bioinformatics/btr597

Examples

```
impute_data(data.frame(x = c(1, NA, 3), y = c(0, 1, 0)),
  method = "median_mode", outcome = "y")
```

`list_task_types` *List registered task types*

Description

List registered task types

Usage

```
list_task_types(implemented_only = FALSE)
```

Arguments

`implemented_only`	Whether to return only implemented task types.

Value

A character vector of task-type ids.

Examples

```
list_task_types()
list_task_types(implemented_only = TRUE)
```

`make_split` *Split uniml data into train, validation, and optional test sets*

Description

Split uniml data into train, validation, and optional test sets

Usage

```
make_split(data, prop, stratify = FALSE, seed = 123)
```

Arguments

`data`	A `uniml_data` object created by `as_uniml_data()`.
`prop`	Numeric vector of length 2 or 3 with proportions summing to 1.
`stratify`	Whether to stratify the split by outcome class.
`seed`	Random seed.

Value

A `uniml_split` object.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
data <- data.frame(
  y = factor(rep(c("no", "yes"), 20), levels = c("no", "yes")),
  x1 = stats::rnorm(40)
)
split <- make_split(as_uniml_data(data, spec), prop = c(0.7, 0.3),
  stratify = TRUE, seed = 1)
names(split)
```

---

make_task *Build an mlr3 classification task*

Description

Build an mlr3 classification task

Usage

```
make_task(data, spec)
```

Arguments

`data`	A data frame or `uniml_data` object.
`spec`	A `uniml_spec` object.

Value

An `mlr3::TaskClassif`.

Examples

```
data <- data.frame(
  y = factor(c("no", "yes", "no", "yes")),
  x = c(0, 1, 0.5, 0.8)
)
spec <- uniml_spec(task_type = "binary", outcome = "y", positive = "yes")
make_task(data, spec)
```

`multiclass_calibration` *Calibration of a multiclass model*

Description

Measures how well a multiclass probability matrix is calibrated, under two stated conventions, and returns the reliability tables that the plots use.

Usage

```
multiclass_calibration(probs, labels, bins = 10L)
```

Arguments

`probs`	A numeric probability matrix with one column per class and one row per patient. The columns must be named and the rows must sum to 1.
`labels`	Observed class labels, one per row of `probs`.
`bins`	Number of equal-width probability bins.

Details

Top-label calibration keeps the largest probability of each row as the confidence and asks whether patients with that confidence really belong to the predicted class. **Class-wise** calibration repeats a one-vs-rest comparison for every class, so the classes are assessed separately instead of only where the model is most confident. The class-wise ECE is the unweighted mean of the per-class ECEs; the top-label ECE weights by how many patients fall in each confidence bin.

ECE and MCE are not proper scoring rules and must not be used to choose between models. Use the Brier score or the log loss from `calc_metrics()` for that, and read ECE/MCE as "how wrong is the stated probability when the model is wrong".

Note that a two-class matrix is *not* the same problem as a binary outcome with `calc_metrics()`: there the binning is on the probability of the event, here it is on the confidence in the class the model picked. The two agree only through `per_class$ece` for the positive class, which is exactly the binary number.

Value

A list with

- `summary`: one row with `n`, `n_classes`, `accuracy`, `mean_confidence`, `level_bias`, `ece`, `mce`, `ece_classwise`, `bins`;
- `per_class`: one row per class with `n`, `prevalence`, `mean_predicted`, `ece`, and `mce`;
- `reliability`: a long table with `scope` (`"top_label"` or `"class"`), `class`, `bin`, `n`, `predicted`, `observed`, and `difference`, which is what `plot_calibration()` draws.

References

Guo C, Pleiss G, Sun Y, Weinberger KQ (2017). On calibration of modern neural networks. *Proceedings of the 34th International Conference on Machine Learning*, 70, 1321-1330.

Kull M, Perello-Nieto M, Kängsepp M, Silva Filho T, Song H, Flach P (2019). Beyond temperature scaling: obtaining well-calibrated multi-class probabilities with Dirichlet calibration. *Advances in Neural Information Processing Systems*, 32.

Van Calster B, McLernon DJ, van Smeden M, Wynants L, Steyerberg EW (2019). Calibration: the Achilles heel of predictive analytics. *BMC Medicine*, 17, 230. doi:10.1186/s12916-019-1466-7

Examples

```
prob <- rbind(
  c(a = 0.7, b = 0.2, c = 0.1),
  c(a = 0.2, b = 0.6, c = 0.2),
  c(a = 0.1, b = 0.3, c = 0.6),
  c(a = 0.5, b = 0.3, c = 0.2)
)
calibration <- multiclass_calibration(prob, labels = c("a", "b", "c", "a"),
                                      bins = 2L)
calibration$summary
calibration$per_class
```

---

new_uniml_result *Construct a uniml result container*

Description

Construct a uniml result container

Usage

```
new_uniml_result(...)
```

Arguments

...	Named result components.

Value

A `uniml_result` object.

Examples

```
result <- new_uniml_result()
class(result)
```

plot.uniml_result *Plot a uniml result*

Description

Dispatches to the performance plots implemented by the plotting layer.

Usage

```
## S3 method for class 'uniml_result'
plot(x, type = "roc", ...)
```

Arguments

x	A uniml_result object.
type	Plot type: "roc", "calibration", "dca", "confusion", "shap_importance", "shap_beeswarm", "shap_dependence", "shap_waterfall", or "shap_force".
...	Arguments passed to the selected plot function.

Value

A ggplot object.

Examples

```
## Not run:
plot(result, type = "roc")

## End(Not run)
```

plot_calibration.uniml_multiclass *Plot probability-calibration curves*

Description

When the predictions contain both raw and calibrated probabilities, a faceted before/after comparison is produced.

Usage

```
## S3 method for class 'uniml_multiclass'
plot_calibration(
  x,
  scope = c("top_label", "class", "both"),
  bins = 10L,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  annotate = TRUE,
  ...
)

## S3 method for class 'uniml_result'
plot_calibration(
  x,
  models = NULL,
  bins = 10L,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  set = NULL,
  annotate = TRUE,
  ...
)

## S3 method for class 'uniml_binary'
plot_calibration(x, models = NULL, bins = 10L, palette = "bmj", ...)

## S3 method for class 'PredictionClassif'
plot_calibration(x, positive = NULL, ...)

## S3 method for class 'data.frame'
plot_calibration(x, models = NULL, bins = 10L, palette = "bmj", ...)

plot_calibration(x, ...)
```

Arguments

x	A `uniml_result` or prediction collection.
scope	Which calibration to draw: `"top_label"` (confidence against accuracy), `"class"` (one one-vs-rest curve per class), or `"both"`. See `multiclass_calibration()` for how the two are defined.
bins	Number of quantile bins.

palette	Discrete palette: a ggsci name, a colour, or a colour vector.
theme	A ggplot2 theme applied to the figure.
title, subtitle, caption	Optional figure labels.
annotate	Whether to add the calibration slope, intercept, and calibration-in-the-large to the caption. They are computed on the predictions themselves, not on the binned points, so they do not depend on bins.
...	Reserved for method-specific arguments.
models	Optional subset of model names. "best" keeps the top-ranked model and "all" keeps every model.
set	Data set to plot; defaults to the comparison set used for ranking.
positive	Positive class label, used when x is an mlr3 prediction.

Value

A ggplot object.

Examples

```
predictions <- data.frame(
  model = "lr",
  prob = c(0.1, 0.4, 0.6, 0.9),
  truth = factor(c("no", "no", "yes", "yes"), levels = c("no", "yes")),
  positive = "yes"
)
plot_calibration(predictions)
```

---

plot_confusion.uniml_multiclass

Plot confusion-matrix heatmaps

Description

Draws counts and row percentages, faceted by model when more than one model is supplied.

Usage

```
## S3 method for class 'uniml_multiclass'
plot_confusion(
  x,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  ...
```

```
)

## S3 method for class 'uniml_result'
plot_confusion(
  x,
  models = NULL,
  threshold = NULL,
  calibrated = FALSE,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  set = NULL,
  ...
)

## S3 method for class 'uniml_binary'
plot_confusion(
  x,
  models = NULL,
  threshold = NULL,
  calibrated = FALSE,
  palette = "bmj",
  ...
)

## S3 method for class 'PredictionClassif'
plot_confusion(x, positive = NULL, ...)

## S3 method for class 'data.frame'
plot_confusion(
  x,
  models = NULL,
  threshold = NULL,
  calibrated = FALSE,
  palette = "bmj",
  ...
)

plot_confusion(x, ...)
```

Arguments

x	A uniml_result or prediction collection.
palette	Continuous fill palette name or colour.
theme	A ggplot2 theme applied to the figure.

title, subtitle, caption	Optional figure labels.
...	Reserved for method-specific arguments.
models	Optional subset of model names. "best" keeps the top-ranked model and "all" keeps every model.
threshold	Classification threshold. When NULL, the configured threshold method is used.
calibrated	Whether to draw the calibrated probabilities.
set	Data set to plot; defaults to the comparison set used for ranking.
positive	Positive class label, used when x is an mlr3 prediction.

Value

A ggplot object.

Examples

```
predictions <- data.frame(
  model = "lr",
  prob = c(0.1, 0.4, 0.6, 0.9),
  truth = factor(c("no", "no", "yes", "yes"), levels = c("no", "yes")),
  positive = "yes"
)
plot_confusion(predictions, threshold = 0.5)
```

---

plot_dca.uniml_multiclass *Plot decision curves*

Description

Plot decision curves

Usage

```
## S3 method for class 'uniml_multiclass'
plot_dca(x, ...)

## S3 method for class 'uniml_result'
plot_dca(
  x,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  ...
```

```
)

## S3 method for class 'uniml_binary'
plot_dca(x, palette = "bmj", ...)

## S3 method for class 'uniml_dca'
plot_dca(x, palette = "bmj", ...)

plot_dca(x, ...)
```

Arguments

x	A uniml_result or uniml_dca table.
...	Reserved for method-specific arguments.
palette	Discrete palette: a ggsci name, a colour, or a colour vector.
theme	A ggplot2 theme applied to the figure.
title, subtitle, caption	Optional figure labels.

Value

A ggplot object.

Examples

```
plot_dca(dca(c(0.1, 0.4, 0.6, 0.9), c(0, 0, 1, 1)))
```

`plot_pr_curve.uniml_result` *Plot precision-recall curves*

Description

The companion figure to `plot_roc()` for an imbalanced outcome. A rare event makes the ROC curve look reassuring, because the false-positive rate divides by a large number of negatives; the precision-recall curve divides by the predicted positives instead, so it stays honest when the event rate is low.

Usage

```
## S3 method for class 'uniml_result'
plot_pr_curve(
  x,
  models = NULL,
  palette = "bmj",
  theme = theme_uniml(),
```

```
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  set = NULL,
  ...
)

## S3 method for class 'uniml_binary'
plot_pr_curve(x, models = NULL, palette = "bmj", ...)

## S3 method for class 'PredictionClassif'
plot_pr_curve(x, positive = NULL, ...)

## S3 method for class 'data.frame'
plot_pr_curve(x, models = NULL, palette = "bmj", ...)

plot_pr_curve(x, ...)
```

Arguments

x	A uniml_result or prediction collection.
models	Optional subset of model names. "best" keeps the top-ranked model and "all" keeps every model.
palette	Discrete palette: a ggsci name, a colour, or a colour vector.
theme	A ggplot2 theme applied to the figure.
title, subtitle, caption	Optional figure labels.
set	Data set to plot; defaults to the comparison set used for ranking.
...	Reserved for method-specific arguments.
positive	Positive class label, used when x is an mlr3 prediction.

Details

The dotted line is the prevalence: a model with no information sits on it, and a useful model sits above it.

Value

A ggplot object.

References

Saito T, Rehmsmeier M (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3), e0118432. doi:10.1371/journal.pone.0118432

Davis J, Goadrich M (2006). The relationship between precision-recall and ROC curves. *Proceedings of the 23rd International Conference on Machine Learning*, 233-240. doi:10.1145/1143844.1143874

Examples

```
predictions <- data.frame(
  model = "lr",
  prob = c(0.1, 0.4, 0.6, 0.9),
  truth = factor(c("no", "no", "yes", "yes"), levels = c("no", "yes")),
  positive = "yes"
)
plot_pr_curve(predictions)
```

---

plot_roc.uniml_multiclass

Plot ROC curves

Description

Accepts a `uniml_result`, a `uniml_binary` prediction, a named list of prediction objects, or a data frame with `model`, `prob`, and `truth` columns. ROC curves are invariant under monotone calibration, so a single curve per model is drawn.

Usage

```
## S3 method for class 'uniml_multiclass'
plot_roc(
  x,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  micro = TRUE,
  ...
)

## S3 method for class 'uniml_result'
plot_roc(
  x,
  models = NULL,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  set = NULL,
  ...
)

## S3 method for class 'uniml_binary'
```

```
plot_roc(x, models = NULL, palette = "bmj", ...)

## S3 method for class 'PredictionClassif'
plot_roc(x, positive = NULL, ...)

## S3 method for class 'data.frame'
plot_roc(x, models = NULL, palette = "bmj", ...)

plot_roc(x, ...)
```

Arguments

x	A uniml_result or prediction collection.
palette	Discrete palette: a ggsci name, a colour, or a colour vector.
theme	A ggplot2 theme applied to the figure.
title, subtitle, caption	Optional figure labels.
micro	For a multiclass prediction, whether to add the micro-average curve beside the one-vs-rest curves. The subtitle reports all three AUCs (Hand-Till, macro one-vs-rest, micro).
...	Reserved for method-specific arguments.
models	Optional subset of model names. "best" keeps the top-ranked model and "all" keeps every model.
set	Data set to plot ("validation", "test", "external", ...); defaults to the comparison set used for ranking.
positive	Positive class label, used when x is an mlr3 prediction.

Value

A ggplot object.

Examples

```
predictions <- data.frame(
  model = "lr",
  prob = c(0.1, 0.4, 0.6, 0.9),
  truth = factor(c("no", "no", "yes", "yes"), levels = c("no", "yes")),
  positive = "yes"
)
plot_roc(predictions)
```

`plot_shap_beeswarm.uniml_result` *Plot a SHAP beeswarm*

Description

Draws a SHAP beeswarm through `shapviz::sv_importance(kind = "beeswarm")`.

Usage

```
## S3 method for class 'uniml_result'
plot_shap_beeswarm(
  x,
  model = "best",
  max_display = 15L,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  ...
)

## S3 method for class 'uniml_binary'
plot_shap_beeswarm(x, model = "best", max_display = 15L, palette = "bmj", ...)

## S3 method for class 'shapviz'
plot_shap_beeswarm(x, model = "best", max_display = 15L, palette = "bmj", ...)

## S3 method for class 'mshapviz'
plot_shap_beeswarm(x, model = "best", max_display = 15L, palette = "bmj", ...)

plot_shap_beeswarm(x, ...)
```

Arguments

`x`	A `uniml_result`, `shapviz`, or `mshapviz` object.
`model`	For `mshapviz` results, `"best"`, `"all"`, or a model name.
`max_display`	Maximum number of features to display.
`palette`	Discrete palette name used by the shared plot theme.
`theme`	A `ggplot2` theme applied to the figure.
`title`, `subtitle`, `caption`	Optional figure labels.
`...`	Additional arguments passed to `shapviz::sv_importance()`.

Value

A ggplot or patchwork object.

Examples

```
## Not run:
plot_shap_beeswarm(result)

## End(Not run)
```

`plot_shap_dependence.uniml_result`

Plot a SHAP dependence curve

Description

When `feature` is `NULL`, the feature with the largest mean absolute SHAP value is used.

Usage

```
## S3 method for class 'uniml_result'
plot_shap_dependence(
  x,
  feature = NULL,
  model = "best",
  color_by = NULL,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  ...
)

## S3 method for class 'uniml_binary'
plot_shap_dependence(
  x,
  feature = NULL,
  model = "best",
  color_by = NULL,
  palette = "bmj",
  ...
)

## S3 method for class 'shapviz'
plot_shap_dependence(
  x,
```

```
  feature = NULL,
  model = "best",
  color_by = NULL,
  palette = "bmj",
  ...
)

## S3 method for class 'mshapviz'
plot_shap_dependence(
  x,
  feature = NULL,
  model = "best",
  color_by = NULL,
  palette = "bmj",
  ...
)

plot_shap_dependence(x, ...)
```

Arguments

x	A uniml_result, shapviz, or mshapviz object.
feature	Feature to draw on the x-axis; NULL picks the most important feature.
model	For mshapviz results, "best", "all", or a model name.
color_by	Optional feature used to colour the points.
palette	Discrete palette name used by the shared plot theme.
theme	A ggplot2 theme applied to the figure.
title, subtitle, caption	Optional figure labels.
...	Additional arguments passed to shapviz::sv_dependence().

Value

A ggplot or patchwork object.

Examples

```
## Not run:
plot_shap_dependence(result)

## End(Not run)
```

`plot_shap_force.uniml_result` *Plot a single-observation SHAP force plot*

Description

Draws a force plot for one row through `shapviz::sv_force()`. A stored `mshapviz` object is drawn one panel per model unless a single `model` is requested.

Usage

```
## S3 method for class 'uniml_result'
plot_shap_force(
  x,
  model = "best",
  row_id = 1L,
  max_display = 6L,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  ...
)

## S3 method for class 'uniml_binary'
plot_shap_force(
  x,
  model = "best",
  row_id = 1L,
  max_display = 6L,
  palette = "bmj",
  ...
)

## S3 method for class 'shapviz'
plot_shap_force(
  x,
  model = "best",
  row_id = 1L,
  max_display = 6L,
  palette = "bmj",
  ...
)

## S3 method for class 'mshapviz'
plot_shap_force(
```

```
  x,
  model = "best",
  row_id = 1L,
  max_display = 6L,
  palette = "bmj",
  ...
)

plot_shap_force(x, ...)
```

Arguments

x	A uniml_result, shapviz, or mshapviz object.
model	For mshapviz results, "best", "all", or a model name.
row_id	Observation (row) to display.
max_display	Maximum number of features to display.
palette	Discrete palette name used by the shared plot theme.
theme	A ggplot2 theme applied to the figure.
title, subtitle, caption	Optional figure labels.
...	Additional arguments passed to shapviz::sv_force().

Value

A ggplot or patchwork object.

Examples

```
## Not run:
plot_shap_force(result)

## End(Not run)
```

---

plot_shap_importance.uniml_result

Plot SHAP importance

Description

Draws a SHAP importance plot through shapviz::sv_importance(). A stored mshapviz object is drawn one panel per model unless a single model is requested.

Usage

```
## S3 method for class 'uniml_result'
plot_shap_importance(
  x,
  model = "best",
  kind = "bar",
  max_display = 15L,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  ...
)

## S3 method for class 'uniml_binary'
plot_shap_importance(
  x,
  model = "best",
  kind = "bar",
  max_display = 15L,
  palette = "bmj",
  ...
)

## S3 method for class 'shapviz'
plot_shap_importance(
  x,
  model = "best",
  kind = "bar",
  max_display = 15L,
  palette = "bmj",
  ...
)

## S3 method for class 'mshapviz'
plot_shap_importance(
  x,
  model = "best",
  kind = "bar",
  max_display = 15L,
  palette = "bmj",
  ...
)

plot_shap_importance(x, ...)
```

Arguments

x	A uniml_result, shapviz, or mshapviz object.
model	For mshapviz results, "best", "all", or a model name.
kind	Importance style passed to shapviz::sv_importance().
max_display	Maximum number of features to display.
palette	Discrete palette name used by the shared plot theme.
theme	A ggplot2 theme applied to the figure.
title, subtitle, caption	Optional figure labels.
...	Additional arguments passed to shapviz::sv_importance().

Value

A ggplot or patchwork object.

Examples

```
## Not run:
plot_shap_importance(result)

## End(Not run)
```

---

plot_shap_waterfall.uniml_result

Plot a single-observation SHAP waterfall

Description

Draws the standard SHAP waterfall for one observation through shapviz::sv_waterfall().

Usage

```
## S3 method for class 'uniml_result'
plot_shap_waterfall(
  x,
  model = "best",
  row_id = 1L,
  max_display = 10L,
  palette = "bmj",
  theme = theme_uniml(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  ...
)
```

```
## S3 method for class 'uniml_binary'
plot_shap_waterfall(
  x,
  model = "best",
  row_id = 1L,
  max_display = 10L,
  palette = "bmj",
  ...
)

## S3 method for class 'shapviz'
plot_shap_waterfall(
  x,
  model = "best",
  row_id = 1L,
  max_display = 10L,
  palette = "bmj",
  ...
)

## S3 method for class 'mshapviz'
plot_shap_waterfall(
  x,
  model = "best",
  row_id = 1L,
  max_display = 10L,
  palette = "bmj",
  ...
)

plot_shap_waterfall(x, ...)
```

Arguments

x	A uniml_result, shapviz, or mshapviz object.
model	For mshapviz results, "best", "all", or a model name.
row_id	Observation to explain.
max_display	Maximum number of features to display.
palette	Discrete palette name used by the shared plot theme.
theme	A ggplot2 theme applied to the figure.
title, subtitle, caption	Optional figure labels.
...	Additional arguments passed to shapviz::sv_waterfall().

Value

A ggplot or patchwork object.

Examples

```
## Not run:
plot_shap_waterfall(result, row_id = 1)

## End(Not run)
```

`pool_mi` *Pool estimates across multiple imputations (strict Rubin's rules)*

Description

Combines `m` per-imputation point estimates and their variance estimates into a single pooled estimate. The total variance follows Rubin (1987), `T = U + (1 + 1/m) B`, and the confidence interval uses the Barnard-Rubin (1999) degrees of freedom with a t quantile, so small `m` does not produce overconfident intervals.

Usage

```
pool_mi(estimates, variances, level = 0.95)
```

Arguments

`estimates`	Numeric vector with one estimate per imputation.
`variances`	Numeric vector with one variance estimate (standard error squared) per imputation, aligned with `estimates`.
`level`	Confidence level.

Value

A list with `mean` (pooled estimate), `se` (square root of the total variance), `lower`, `upper`, `df` (Barnard-Rubin degrees of freedom; `Inf` when no between-imputation variance is present), `m`, `within` (`U`), `between` (`B`), and `total` (`T`).

References

Rubin DB (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley. doi:10.1002/9780470316696

Barnard J, Rubin DB (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86(4), 948-955. doi:10.1093/biomet/86.4.948

Examples

```
pool_mi(c(0.80, 0.82, 0.79), c(0.0004, 0.0004, 0.0005))
```

predict_uniml *Predict with a fitted uniml workflow*

Description

Reuses the stored preprocessing, feature selection, calibration, and final model. The new data are never used to refit any component.

Usage

```
predict_uniml(object, newdata, model = "best", ...)

## S3 method for class 'uniml_result'
predict(object, newdata, model = "best", ...)
```

Arguments

object	A uniml_result returned by run_workflow().
newdata	New data frame with the same feature columns as the training data.
model	"best" or one of names(object$models).
...	Reserved for future use.

Value

A data frame with the model key, raw and calibrated probabilities, the classification from the stored threshold, the stored threshold itself, and the truth labels when an outcome column is available.

Examples

```
## Not run:
res <- run_workflow(spec, train = train, validation = validation)
predict_uniml(res, train)

## End(Not run)
```

print.uniml_calibration *Print a fitted calibration*

Description

Print a fitted calibration

Usage

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

Arguments

x	A uniml_calibration object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

---

print.uniml_data *Print a uniml data object*

Description

Print a uniml data object

Usage

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

Arguments

x	A uniml_data object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

print.uniml_preprocessor *Print a uniml preprocessor*

Description

Print a uniml preprocessor

Usage

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

Arguments

x	A uniml_preprocessor object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

print.uniml_result *Print a uniml result*

Description

Print a uniml result

Usage

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

Arguments

x	A uniml_result object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

print.uniml_shap *Print a SHAP result*

Description

Print a SHAP result

Usage

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

Arguments

x	A uniml_shap object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

print.uniml_spec *Print a uniml specification*

Description

Print a uniml specification

Usage

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

Arguments

x	A uniml_spec object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

print.uniml_split *Print a uniml split*

Description

Print a uniml split

Usage

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

Arguments

x	A uniml_split object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

print.uniml_task_type *Print a task-type descriptor*

Description

Print a task-type descriptor

Usage

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

Arguments

x	A uniml_task_type object.
...	Additional arguments passed to methods.

Value

Invisibly returns x.

rank_models *Rank models on a comparison set*

Description

Ranks one row per model using either a single metric or the default composite score. The composite score is the mean of AUC, sensitivity, specificity, F2, balanced accuracy, and Youden's index when those metrics are available.

Usage

```
rank_models(
  x,
  select_on = "auto",
  best_model_metric = "composite",
  task_type = "auto"
)
```

Arguments

x	An `uniml_result` object or a metric data frame.
select_on	Comparison set: `"auto"`, `"validation"`, or `"test"`. `"auto"` uses the validation set when present and otherwise the test set, so the test set stays reporting-only whenever a validation set exists.
best_model_metric	Metric used for ranking, or `"composite"`.
task_type	Task type whose composite metric set is used, or `"auto"`. `"auto"` reads the task type from a `uniml_result` and otherwise assumes `"binary"`, which keeps the historical behaviour for plain metric tables.

Value

A data frame with model scores, ranks, metric, and comparison set.

Examples

```
metrics <- data.frame(
  model = c("lr", "rf"), set = "validation",
  auc = c(0.80, 0.84), sensitivity = c(0.72, 0.76),
  specificity = c(0.70, 0.73), f2 = c(0.70, 0.75),
  balanced_accuracy = c(0.71, 0.75), youden = c(0.42, 0.49)
)
rank_models(metrics)
```

register_learner *Register a learner for use by* resolve_learners()

Description

Registers an mlr3 learner and an optional parameter search space. Registering an existing key replaces the previous entry.

Usage

```
register_learner(key, learner, search_space = NULL)
```

Arguments

key	Single character learner key.
learner	An mlr3 learner object.
search_space	Optional paradox::ParamSet used for tuning.

Value

Invisibly returns the registered learner specification.

Examples

```
## Not run:
register_learner(
  "rpart_custom",
  mlr3::lrn("classif.rpart", predict_type = "prob"),
  paradox::ps(cp = paradox::p_dbl(0.001, 0.1))
)

## End(Not run)
```

register_task_type *Register a task type*

Description

Register a task type

Usage

```
register_task_type(spec, overwrite = FALSE)
```

Arguments

spec	A uniml_task_type object created by task_type_spec().
overwrite	Whether to replace an existing registration.

Value

Invisibly returns spec.

Examples

```
register_task_type(task_type_spec(id = "toy", mlr3_task = "classif"))
list_task_types()
```

---

register_workflow_adapter

Register a workflow adapter for a task type

Description

The workflow engine (workflow_engine()) is shared across task types. What a task type contributes is an adapter: how its data frame is prepared, how its mlr3 task is built, how a learner's prediction is extracted, how predictions from several imputations are pooled, and how a prediction becomes metrics.

Usage

```
register_workflow_adapter(task_type, adapter, overwrite = FALSE)
```

Arguments

task_type	Task-type id, such as "binary".
adapter	A list of adapter functions. See Details.
overwrite	Whether to replace an existing registration.

Details

An adapter is a list with these elements:

- prepare(data, spec, name, levels, require_two): returns list(data, outcome, levels), like the internal binary preparation does.
- make_task(data, spec): returns an mlr3 task.
- predict(learner, task, spec): returns list(prediction, value), where value is the object the metrics are computed from (a numeric vector for binary outcomes, a matrix of survival probabilities for survival ones).
- pool(values): combines a list of value objects, one per imputation.
- metric_names(spec): the metric names the workflow reports.

- `metrics(value, truth, spec, metrics, context)`: returns a named list. `context` carries stage-specific information such as the threshold.

Two hooks are optional and default to `train_model()` and `tune_model()`:

- `train(task, learner, params, spec)`: returns a fitted learner.
- `tune(task, learner_spec, spec)`: returns a list with at least `learner`, `best_params`, `best_metric`, and `status`, like `tune_model()` does. A task type whose learners are not `TaskClassif` learners supplies both.

Value

Invisibly returns `adapter`.

Examples

```
hooks <- names(workflow_adapter("binary"))
hooks
```

`remove_collinear` *Remove collinear features*

Description

Iteratively removes one feature from the strongest absolute correlation pair and/or the feature with the largest variance inflation factor (VIF). Correlations and VIF values are recomputed after each removal.

Usage

```
remove_collinear(data, cor_threshold = NULL, vif_threshold = NULL)
```

Arguments

`data`	A numeric data frame or matrix with named columns.
`cor_threshold`	Optional absolute-correlation threshold in `[0, 1]`.
`vif_threshold`	Optional VIF threshold, at least 1.

Value

A list containing `selected_features`, `removed_features`, removal `details`, `history`, the final `correlation` matrix, and final `vif` values when VIF removal was requested.

Examples

```
data <- data.frame(
  x1 = c(1, 2, 3, 4, 5),
  x2 = c(1.1, 2.0, 3.1, 4.0, 5.1),
  x3 = c(5, 3, 4, 1, 2)
)
remove_collinear(data, cor_threshold = 0.99)
```

---

resolve_learners *Resolve learners from a uniml specification*

Description

Converts learner keys and learner objects in `spec` into learner specifications containing an mlr3 learner, parameter values, search space, and tuning flag.

Usage

```
resolve_learners(spec, task = NULL, n_train = NULL)
```

Arguments

`spec`	A `uniml_spec` object.
`task`	Optional mlr3 classification task used to resolve task-dependent search spaces such as `mtry`.
`n_train`	Optional number of rows available to a learner during a training fold. Used to keep sample-size-dependent search spaces safe.

Value

A list of `uniml_learner_spec` objects.

Examples

```
## Not run:
spec <- uniml_spec(task_type = "binary", outcome = "y", positive = "yes")
spec <- set_learners(spec, c("lr", "rf"))
resolve_learners(spec)

## End(Not run)
```

`run_workflow` *Run a complete uniml workflow*

Description

Chains preprocessing, feature selection, tuning, cross validation, calibration, evaluation, and ranking into a single `uniml_result`. The validation and optional test sets never take part in fitting; they are only transformed with parameters estimated on the training data.

Usage

```
run_workflow(
  spec,
  train,
  validation,
  test = NULL,
  external = NULL,
  ci_R = 1000
)
```

Arguments

`spec`	A `uniml_spec` object.
`train`	Training data frame.
`validation`	Validation data frame.
`test`	Optional test data frame; reporting-only.
`external`	Optional external validation data frame; reporting-only.
`ci_R`	Number of bootstrap resamples for confidence intervals.

Details

The fitting half of the pipeline lives in `workflow_engine()`, which is task-type neutral: it prepares the data, tunes, trains, and predicts through the adapter registered by `register_workflow_adapter()`. What `run_workflow()` adds is the binary report, because thresholds, calibration, DeLong and Hosmer-Lemeshow tests, NRI/IDI, and decision curves are defined for a single probability per patient. Other task types call `workflow_engine()` and score the predictions themselves instead of running this function in a reduced form.

Value

A `uniml_result` with fitted models, predictions, metrics, and confidence intervals.

References

Collins GS, Moons KGM, Dhiman P, et al. (2024). TRIPOD+AI statement: updated guidance for reporting clinical prediction models that use regression or machine learning methods. *BMJ*, 385, e078378. doi:10.1136/bmj-2023-078378

Steyerberg EW, Harrell FE, Borsboom GJ, et al. (2001). Internal validation of predictive models: efficiency of some procedures for logistic regression analysis. *Journal of Clinical Epidemiology*, 54(8), 774-781. doi:10.1016/S0895-4356(01)00341-9

Steyerberg EW (2019). *Clinical Prediction Models: A Practical Approach to Development, Validation, and Updating*, 2nd edition. Springer. doi:10.1007/978-3-030-16399-0

Examples

```
## Not run:
data <- data.frame(
  y = factor(rep(c("no", "yes"), 20)),
  x1 = stats::rnorm(40),
  x2 = stats::rnorm(40)
)
spec <- uniml_spec(task_type = "binary", outcome = "y", positive = "yes")
spec <- set_preprocess(spec, impute = "none", scale = "none", encode = "none")
spec <- set_feature_selection(spec, methods = "univariate")
spec <- set_learners(spec, "cart", tune = FALSE)
spec <- set_tuning(spec, folds = 2, evals = 1)
spec <- set_evaluation(spec, ci = "none", calibration = "none")
run_workflow(spec, train = data[1:30, ], validation = data[31:40, ])

## End(Not run)
```

---

save_result *Save and read a fitted result*

Description

`save_result()` writes the whole `uniml_result` — the fitted learners, the fitted preprocessor, the selected features, and the specification — to a single file. That file is what someone else needs to score new patients and to compute their own analyses on it, including the subgroup metrics this package deliberately does not report.

Usage

```
save_result(result, path)

read_result(path)
```

Arguments

`result`	A `uniml_result` object.
`path`	File to write or read. The `.rds` extension is added when missing.

Details

The saved object is a plain R object: `readRDS()` reads it, and `predict_uniml()` scores new data with it. `save_result()` exists so the file has a version stamp and is validated on the way out and back in, rather than so it can do something `saveRDS()` cannot.

Value

`save_result()` returns the normalised path; `read_result()` returns the `uniml_result`.

References

R Core Team (2024). *R: A Language and Environment for Statistical Computing*. Section "Serialization".

Examples

```
## Not run:
result <- run_workflow(spec, train, validation)
path <- save_result(result, tempfile(fileext = ".rds"))
restored <- read_result(path)

# Subgroup analysis is yours to define: score the subgroup, then use the same
# metric function the workflow uses.
subgroup <- validation[validation$age_group == "older", ]
scored <- predict_uniml(restored, subgroup)
calc_metrics(scored, subgroup$y, positive = "yes")

## End(Not run)
```

`scale_numeric` *Scale numeric variables*

Description

Convenience wrapper that applies the fixed preprocessing order and returns the scaled data.

Usage

```
scale_numeric(
  data,
  method = c("zscore", "minmax", "none"),
  vars = NULL,
  outcome = NULL,
  ...
)
```

Arguments

data	A data frame or uniml_data object.
method	Scaling method.
vars	Optional character vector restricting which columns are scaled.
outcome	Outcome column name.
...	Arguments passed to fit_preprocessor().

Value

The scaled data, matching the class of data.

Examples

```
scale_numeric(data.frame(x = c(1, 2, 3), y = c(0, 1, 0)), outcome = "y")
```

---

select_boruta *Select features with Boruta*

Description

Runs the Boruta all-relevant algorithm on the encoded feature matrix and retains Confirmed features by default. The importance history is stored in scores$ImpHistory together with the final decision, so that it is kept for later inspection or plotting.

Usage

```
select_boruta(
  data,
  outcome,
  maxRuns = 50L,
  num_threads = 1L,
  seed = 123,
  tentative = FALSE,
  ...
)
```

Arguments

data	A data frame, uniml_data, or fitted uniml_preprocessor.
outcome	Outcome column name.
maxRuns	Maximum number of Boruta runs.
num_threads	Number of threads used by Boruta.
seed	Random seed.
tentative	Whether tentative features are also retained.
...	Additional arguments passed to Boruta::Boruta().

Value

A `uniml_selection_result` with `selected_features`, `details`, `params`, and `scores`.

References

Kursa MB, Rudnicki WR (2010). Feature selection with the Boruta package. *Journal of Statistical Software*, 36(11), 1-13. doi:10.18637/jss.v036.i11

Examples

```
## Not run:
data <- data.frame(
  y = factor(rep(c("no", "yes"), 10)),
  x1 = stats::rnorm(20),
  x2 = stats::rnorm(20)
)
select_boruta(data, "y", maxRuns = 11, num_threads = 1)

## End(Not run)
```

---

select_features *Run feature selection from a uniml specification*

Description

Runs one or more feature-selection methods on the training/encoded data, combines their results, and maps selected encoded columns back to original variables when encoding metadata is available.

Usage

```
select_features(data, spec)
```

Arguments

`data`	A data frame, `uniml_data`, or fitted `uniml_preprocessor`.
`spec`	A `uniml_spec` object.

Value

A list containing selected encoded and original features, method details, and optional collinearity results.

Examples

```
## Not run:
spec <- uniml_spec(task_type = "binary", outcome = "y", positive = "yes")
spec <- set_feature_selection(spec, methods = "lasso")
data <- data.frame(
  y = factor(rep(c("no", "yes"), 10)),
  x1 = stats::rnorm(20),
  x2 = stats::rnorm(20)
)
select_features(data, spec)

## End(Not run)
```

`select_lasso` *Select features with LASSO*

Description

Fits a cross-validated binomial LASSO using `glmnet` and retains features with non-zero coefficients at the selected lambda rule.

Usage

```
select_lasso(
  data,
  outcome,
  lambda = c("1se", "min"),
  nfolds = 10L,
  seed = 123,
  alpha = 1,
  type_measure = "auc",
  standardize = TRUE
)
```

Arguments

`data`	A data frame, `uniml_data`, or fitted `uniml_preprocessor`.
`outcome`	Outcome column name.
`lambda`	Lambda selection rule, `"1se"` or `"min"`.
`nfolds`	Number of cross-validation folds.
`seed`	Random seed.
`alpha`	Elastic-net mixing parameter (1 = LASSO).
`type_measure`	`cv.glmnet()` loss, `"auc"`, `"deviance"`, or `"class"`.
`standardize`	Whether predictors are standardised inside `glmnet`.

Value

A list with `selected_features`, `details`, and `params`.

References

Tibshirani R (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1), 267-288. doi:10.1111/j.2517-6161.1996.tb02080.x

Friedman J, Hastie T, Tibshirani R (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 1-22. doi:10.18637/jss.v033.i01

Examples

```
## Not run:
data <- data.frame(
  y = factor(rep(c("no", "yes"), 10)),
  x1 = stats::rnorm(20),
  x2 = stats::rnorm(20)
)
select_lasso(data, "y", nfolds = 3)

## End(Not run)
```

`select_multivariate` *Select features with multivariable logistic regression*

Description

Fits a multivariable logistic model and retains features using either stepwise selection or repeated p-value elimination.

Usage

```
select_multivariate(
  data,
  outcome,
  method = c("stepwise", "pvalue"),
  p_threshold = 0.2
)
```

Arguments

`data`	A data frame, `uniml_data`, or fitted `uniml_preprocessor`.
`outcome`	Outcome column name.
`method`	Multivariable method, `"stepwise"` or `"pvalue"`.
`p_threshold`	P-value threshold for `"pvalue"`.

Value

A list with `selected_features`, `details`, and `params`.

Examples

```
data <- data.frame(
  y = factor(c("no", "yes", "no", "yes", "no", "yes")),
  x1 = c(0, 1, 0, 1, 0, 1),
  x2 = c(1, 0, 1, 0, 1, 0)
)
select_multivariate(data, "y", method = "pvalue", p_threshold = 1)
```

`select_rf_importance` *Select features with random-forest importance*

Description

Fits a ranger random forest and ranks encoded features by importance. By default the top `ceiling(sqrt(number_of_featur` features are retained.

Usage

```
select_rf_importance(
  data,
  outcome,
  top_k = NULL,
  importance_threshold = NULL,
  num_trees = 100L,
  importance = c("impurity", "permutation"),
  seed = 123
)
```

Arguments

`data`	A data frame, `uniml_data`, or fitted `uniml_preprocessor`.
`outcome`	Outcome column name.
`top_k`	Optional number of top features to retain.
`importance_threshold`	Optional minimum importance score.
`num_trees`	Number of trees in the random forest.
`importance`	Importance type supported by ranger, `"impurity"` or `"permutation"`.
`seed`	Random seed.

Value

A list with `selected_features`, `details`, and `params`.

Examples

```
## Not run:
data <- data.frame(
  y = factor(rep(c("no", "yes"), 10)),
  x1 = stats::rnorm(20),
  x2 = stats::rnorm(20)
)
select_rf_importance(data, "y", top_k = 1, num_trees = 20)

## End(Not run)
```

`select_rfe` *Select features with recursive feature elimination*

Description

Uses `mlr3fselect` with a recursive feature elimination selector. By default the target size is `ceiling(sqrt(number_of_features))`.

Usage

```
select_rfe(
  data,
  outcome,
  n_features = NULL,
  learner = NULL,
  resampling = NULL,
  measure = NULL,
  seed = 123
)
```

Arguments

`data`	A data frame, `uniml_data`, or fitted `uniml_preprocessor`.
`outcome`	Outcome column name.
`n_features`	Optional target number of features.
`learner`	Optional mlr3 learner. Defaults to `classif.rpart`.
`resampling`	Optional mlr3 resampling object.
`measure`	Optional mlr3 measure.
`seed`	Random seed.

Value

A list with `selected_features`, `details`, and `params`.

References

Guyon I, Weston J, Barnhill S, Vapnik V (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1), 389-422. doi:10.1023/A:1012487302797

Examples

```
## Not run:
data <- data.frame(
  y = factor(rep(c("no", "yes"), 10)),
  x1 = stats::rnorm(20),
  x2 = stats::rnorm(20),
  x3 = stats::rnorm(20)
)
select_rfe(data, "y", n_features = 2)

## End(Not run)
```

---

select_univariate *Select features with univariable logistic regression*

Description

Fits one logistic regression per encoded feature and retains features with a p-value below `p_threshold`.

Usage

```
select_univariate(data, outcome, p_threshold = 0.2)
```

Arguments

`data`	A data frame, `uniml_data`, or fitted `uniml_preprocessor`.
`outcome`	Outcome column name.
`p_threshold`	P-value threshold used to retain a feature.

Value

A list with `selected_features`, `details`, `params`, and `scores`.

Examples

```
data <- data.frame(
  y = factor(c("no", "yes", "no", "yes")),
  x = c(0, 1, 0.2, 0.8)
)
select_univariate(data, "y", p_threshold = 1)
```

`set_evaluation` *Set evaluation options*

Description

Set evaluation options

Usage

```
set_evaluation(
  spec,
  metrics = "all",
  ci = "bootstrap",
  threshold_method = "youden",
  cost = c(fp = 1, fn = 1),
  fixed_threshold = NULL,
  reference_model = NULL,
  best_model = "composite",
  calibration = "none",
  calibration_args = NULL,
  tests = c("delong", "hl"),
  dca = TRUE,
  select_on = "validation",
  ...
)
```

Arguments

`spec`	A `uniml_spec` object.
`metrics`	Metrics to calculate.
`ci`	Confidence-interval method.
`threshold_method`	Classification-threshold method.
`cost`	Cost vector for cost-based thresholds.
`fixed_threshold`	Fixed threshold when requested.
`reference_model`	Reference model for NRI or IDI.
`best_model`	Best-model selection metric.
`calibration`	Calibration method.
`calibration_args`	Named list with `folds`, `bins`, and `fit` (`"out_of_fold"` or `"in_sample"`).
`tests`	Post-model statistical tests.
`dca`	Whether to calculate decision-curve analysis.
`select_on`	Data set used for model comparison.
`...`	Additional evaluation fields.

Value

Updated `uniml_spec`.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_evaluation(
  spec,
  ci = "bootstrap",
  calibration = "platt",
  select_on = "validation"
)
spec
```

`set_explain` *Set explanation options*

Description

Set explanation options

Usage

```
set_explain(
  spec,
  model = "best",
  plots = c("importance", "beeswarm", "dependence", "waterfall"),
  n_explain = NULL,
  bg_n = NULL,
  thresholds = NULL,
  ...
)
```

Arguments

`spec`	A `uniml_spec` object.
`model`	Model to explain. `"none"` skips SHAP entirely, which is the way to keep a workflow fast when no explanation is needed.
`plots`	SHAP plots to produce: any of `"importance"`, `"beeswarm"`, `"dependence"`, `"waterfall"`, and `"force"`.
`n_explain`	Number of rows to explain. Kernel SHAP costs roughly one prediction pass per explained row, so this is the first knob to turn when a workflow feels slow.
`bg_n`	Size of the background sample Kernel SHAP averages over.
`thresholds`	Optional list configuring SHAP zero-crossing thresholds: `targets` (character vector), `interaction`, `min_n`, `min_unique`, `span`.
`...`	Additional explanation fields.

Value

Updated `uniml_spec`.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_explain(spec, model = "best", plots = c("importance", "beeswarm"))
spec
```

`set_feature_selection` *Set feature-selection options*

Description

Set feature-selection options

Usage

```
set_feature_selection(
  spec,
  methods = NULL,
  lambda = NULL,
  p_threshold = NULL,
  maxRuns = NULL,
  multivariate = NULL,
  combine = NULL,
  representation = NULL,
  k = NULL,
  cor_threshold = NULL,
  vif_threshold = NULL,
  args = NULL,
  ...
)
```

Arguments

`spec`	A `uniml_spec` object.
`methods`	Feature-selection method names, or `"none"` to keep every candidate. `"none"` exists for task types whose outcome is not a class label: the other methods screen the outcome column, which would be the wrong column.
`lambda`	LASSO lambda rule.
`p_threshold`	Univariate p-value threshold.
`maxRuns`	Boruta maximum runs.
`multivariate`	Multivariable selection method.
`combine`	How selected feature sets are combined.

representation	Feature representation used for modelling.
k	Optional target number of features for RFE or importance selection.
cor_threshold	Optional absolute-correlation threshold for collinearity removal.
vif_threshold	Optional VIF threshold for collinearity removal.
args	Named list of method-specific arguments, e.g. `list(lasso = list(nfolds = 5), boruta = list(tentative = TRUE))`.
...	Additional selection fields.

Value

Updated `uniml_spec`.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_feature_selection(
  spec,
  methods = c("univariate", "lasso"),
  combine = "intersect"
)
spec$selection$methods
```

---

set_labels *Set display labels*

Description

Set display labels

Usage

```
set_labels(spec, mapping = NULL)
```

Arguments

spec	A `uniml_spec` object.
mapping	A named character vector mapping column names to labels.

Value

Updated `uniml_spec`.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_labels(spec, c(age = "Age (years)"))
spec
```

set_leakage_policy *Set the data-leakage policy*

Description

Set the data-leakage policy

Usage

```
set_leakage_policy(
  spec,
  mode = "custom",
  in_cv = c("selection", "balance", "tuning")
)
```

Arguments

spec	A uniml_spec object.
mode	Leakage policy mode.
in_cv	Steps fitted inside cross-validation.

Value

Updated uniml_spec.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_leakage_policy(
  spec,
  mode = "strict_in",
  in_cv = c("selection", "balance")
)
spec$leakage_policy
spec$in_cv
```

set_learners *Set learners*

Description

Set learners

Usage

```
set_learners(spec, learners, tune = TRUE, params = NULL, search_space = NULL)
```

Arguments

spec	A uniml_spec object.
learners	Learner keys or learner objects.
tune	Whether to tune the configured learners.
params	Optional learner parameters.
search_space	Optional parameter search space.

Value

Updated uniml_spec.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_learners(spec, c("lr", "cart"), tune = FALSE)
spec
```

---

set_output *Set output options*

Description

Set output options

Usage

```
set_output(
  spec,
  verbose = 1,
  n_workers = 1,
  future_strategy = "multisession",
  dir = "output",
  ...
)
```

Arguments

spec	A uniml_spec object.
verbose	Verbosity level from 0 to 2.
n_workers	Number of workers.
future_strategy	Future backend strategy.
dir	Output directory.
...	Additional output fields.

Value

Updated `uniml_spec`.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_output(spec, dir = tempfile("uniml-out-"))
spec
```

`set_preprocess` *Set preprocessing options*

Description

Set preprocessing options

Usage

```
set_preprocess(
  spec,
  impute = NULL,
  m = NULL,
  scale = NULL,
  encode = NULL,
  ordered_as = NULL,
  outcome_na_action = NULL,
  balance = NULL,
  vars = NULL,
  impute_args = NULL,
  scale_args = NULL,
  encode_args = NULL,
  balance_args = NULL,
  ...
)
```

Arguments

`spec`	A `uniml_spec` object.
`impute`	Imputation method.
`m`	Number of imputations when applicable.
`scale`	Scaling method.
`encode`	Encoding method.
`ordered_as`	Ordered-factor handling.
`outcome_na_action`	How to handle missing outcome values.

balance	Class-balance method.
vars	Optional character vector naming the columns the preprocessor may transform. Columns outside it are passed through untouched, which is how a task type keeps its own outcome columns (a survival time and status, for example) out of scaling and encoding.
impute_args	Named list forwarded to fit_imputer().
scale_args	Named list forwarded to fit_scale().
encode_args	Named list forwarded to fit_encode().
balance_args	Named list forwarded to fit_balance().
...	Additional preprocessing fields.

Value

Updated uniml_spec.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_preprocess(
  spec,
  impute = "median_mode",
  scale = "zscore",
  encode = "onehot"
)
spec$preprocess$impute
```

---

set_tuning *Set tuning options*

Description

Set tuning options

Usage

```
set_tuning(
  spec,
  tuner = "random_search",
  folds = 5,
  repeats = 1,
  evals = 30,
  metric = "auc",
  nested = FALSE,
  seed = 123,
  terminator = "evals",
  seconds = NULL,
  ...
)
```

Arguments

spec	A uniml_spec object.
tuner	Tuning strategy.
folds	Number of cross-validation folds.
repeats	Number of cross-validation repeats.
evals	Number of tuning evaluations.
metric	Tuning metric.
nested	Whether tuning is nested.
seed	Random seed.
terminator	Tuning terminator: "evals" or "seconds".
seconds	Wall-clock budget used when terminator = "seconds".
...	Additional tuning fields.

Value

Updated uniml_spec.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
spec <- set_tuning(
  spec,
  tuner = "random_search",
  folds = 5,
  evals = 20,
  metric = "auc"
)
spec
```

---

summary.uniml_result *Summarise a uniml result*

Description

Summarise a uniml result

Usage

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

Arguments

object	A uniml_result object.
...	Additional arguments passed to methods.

Value

A summary list.

`task_type_spec` *Task-type descriptor*

Description

A descriptor records which mlr3 backend, outcome shape, metrics, learners, and optional capabilities (SHAP / DCA / calibration) a task type provides.

Usage

```
task_type_spec(
  id,
  mlr3_task = NULL,
  outcome_class = NULL,
  positive = TRUE,
  metrics = character(),
  composite_metrics = character(),
  default_learners = character(),
  supports_shap = FALSE,
  supports_dca = FALSE,
  supports_calibration = FALSE,
  implemented = TRUE
)
```

Arguments

`id`	Task-type id, such as `"binary"`.
`mlr3_task`	mlr3 task backend, such as `"classif"`.
`outcome_class`	Human-readable outcome shape.
`positive`	Whether the positive outcome class must be specified.
`metrics`	Supported metric names.
`composite_metrics`	Metric names averaged into the composite ranking score used by `rank_models()`. Registered task types that leave this empty are ranked on an explicit metric only.
`default_learners`	Default learner keys.
`supports_shap`	Whether SHAP explanations are supported.
`supports_dca`	Whether decision-curve analysis is supported.
`supports_calibration`	Whether probability calibration is supported.
`implemented`	Whether the task type is implemented.

Value

A `uniml_task_type` object.

Examples

```
task_type_spec(id = "binary", mlr3_task = "classif", metrics = "auc")
```

`theme_uniml` *Unified plotting theme*

Description

Creates the restrained plotting theme used by all `uniml` figures. When `ggsci` is installed, the default `"bmj"` palette is used for discrete colour and fill scales.

Usage

```
theme_uniml(
  base_size = 11,
  base_family = "",
  legend_position = "right",
  grid = TRUE
)
```

Arguments

`base_size`	Base font size.
`base_family`	Base font family.
`legend_position`	Legend position.
`grid`	Whether to draw major grid lines.

Value

A `ggplot2` theme object.

Examples

```
theme_uniml()
```

`train_model` *Train a final classification model*

Description

Clones the supplied learner, applies optional named parameters, and trains it on the provided training task. The task is not modified.

Usage

```
train_model(task, learner, params = NULL)
```

Arguments

`task`	An mlr3 classification task.
`learner`	An mlr3 learner or a `uniml_learner_spec` object.
`params`	Optional named list of learner parameters. Values supplied here override parameters stored in `learner`.

Value

A trained mlr3 `Learner`.

References

Lang M, Binder M, Richter J, et al. (2019). mlr3: a modern object-oriented machine learning framework in R. *Journal of Open Source Software*, 4(44), 1903. doi:10.21105/joss.01903

Examples

```
## Not run:
data <- data.frame(
  y = factor(rep(c("no", "yes"), 20)),
  x1 = stats::rnorm(40),
  x2 = stats::rnorm(40)
)
spec <- uniml_spec(task_type = "binary", outcome = "y", positive = "yes")
task <- make_task(data, spec)
train_model(task, mlr3::lrn("classif.rpart", predict_type = "prob"))

## End(Not run)
```

`tripod_ai_check` *Audit a result against the TRIPOD+AI reporting domains*

Description

Reads a `run_workflow()` result and reports, domain by domain, whether the information required by the TRIPOD+AI statement (Collins et al., 2024) is present in the object, partially present, absent, or out of scope for the package. The checklist is an audit aid: it never claims compliance, and the items marked `"not_supported"` must be written by the author.

Usage

```
tripod_ai_check(result, file = NULL)
```

Arguments

`result`	A `uniml_result` from `run_workflow()`.
`file`	Optional path. When supplied, the checklist is also written to CSV.

Value

A data frame with columns `section`, `domain`, `requirement`, `status` (`"reported"`, `"partial"`, `"not_reported"`, or `"not_supported"`), `evidence`, and `note`.

References

Collins GS, Moons KGM, Dhiman P, et al. (2024). TRIPOD+AI statement: updated guidance for reporting clinical prediction models that use regression or machine learning methods. *BMJ*, 385, e078378. doi:10.1136/bmj-2023-078378

Collins GS, Reitsma JB, Altman DG, Moons KGM (2015). Transparent reporting of a multivariable prediction model for individual prognosis or diagnosis (TRIPOD): the TRIPOD statement. *Annals of Internal Medicine*, 162(1), 55-63. doi:10.7326/M14-0697

Examples

```
## Not run:
result <- run_workflow(spec, train, validation, test)
checklist <- tripod_ai_check(result)
table(checklist$status)

## End(Not run)
```

tune_model *Tune an mlr3 learner*

Description

Tunes one resolved learner specification on a training task. Random search, grid search, and Bayesian optimization are supported. When `nested = FALSE`, the tuning instance is optimized on the supplied resampling and the selected parameters are evaluated again with the same resampling. When `nested = TRUE`, an `mlr3tuning::auto_tuner()` is evaluated on an outer resampling before the best configuration is extracted on the full task. The tuning metric is taken from `learner_spec$metric`, which is normally populated from `set_tuning()`.

Usage

```
tune_model(
  task,
  learner_spec,
  resampling = NULL,
  tuner = "random_search",
  terminator = NULL,
  nested = FALSE,
  seed = 123
)
```

Arguments

`task`	An mlr3 classification task.
`learner_spec`	A `uniml_learner_spec` object from `resolve_learners()`.
`resampling`	An mlr3 resampling object used for tuning and evaluation.
`tuner`	Tuning strategy: `"random_search"`, `"grid_search"`, `"bayesian"`, or an mlr3 tuner object. Bayesian tuning requires the `mlr3mbo`, `DiceKriging`, and `rgenoud` packages; the latter two are used for the Kriging surrogate and the acquisition-function optimizer.
`terminator`	An mlr3tuning terminator. The default uses 20 evals.
`nested`	Whether to use nested tuning.
`seed`	Random seed.

Value

A structured `uniml_tuning_result` with the configured learner, best parameters, best tuning value, reproducibility metadata, and the resulting cross-validation result.

References

Jones DR, Schonlau M, Welch WJ (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13, 455-492.

Snoek J, Larochelle H, Adams RP (2012). Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 25.

Bischl B, Binder M, Lang M, et al. (2023). Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, 13(2), e1484.

Examples

```
## Not run:
data <- data.frame(
  y = factor(rep(c("no", "yes"), 20)),
  x1 = stats::rnorm(40),
  x2 = stats::rnorm(40)
)
spec <- uniml_spec(task_type = "binary", outcome = "y", positive = "yes")
spec <- set_learners(spec, "cart", tune = TRUE)
task <- make_task(data, spec)
learners <- resolve_learners(spec, task = task)
tune_model(
  task, learners[[1L]],
  resampling = mlr3::rsmp("cv", folds = 3),
  tuner = "random_search",
  terminator = mlr3tuning::trm("evals", n_evals = 2)
)

## End(Not run)
```

`uniml_spec` *Construct a uniml specification*

Description

The specification is the top-level configuration object used by uniml. It stores variable roles and the default settings for each later stage.

Usage

```
uniml_spec(
  task_type,
  outcome,
  positive = NULL,
  id = NULL,
  exclude = NULL,
  labels = NULL,
  ...
)
```

Arguments

task_type	Task type. M2 supports "binary".
outcome	Outcome column name.
positive	Positive outcome class. Required for a binary task and must be NULL for a task type without a positive class, such as "multiclass".
id	Optional ID column name.
exclude	Optional character vector of columns excluded from modelling.
labels	Optional named character vector mapping column names to display labels.
...	Named specification fields, such as leakage_policy or seed.

Value

A uniml_spec object.

Examples

```
spec <- uniml_spec(
  task_type = "binary",
  outcome = "died",
  positive = "yes",
  id = "id"
)
spec
```

---

validate_uniml_data *Validate a uniml data object*

Description

Validate a uniml data object

Usage

```
validate_uniml_data(x)
```

Arguments

x	A uniml_data object.

Value

Invisibly returns TRUE when valid.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
data <- data.frame(
  y = factor(c("no", "yes"), levels = c("no", "yes")),
  x1 = c(0.1, 0.2)
)
validate_uniml_data(as_uniml_data(data, spec))
```

`validate_uniml_preprocessor` *Validate a uniml preprocessor*

Description

Validate a uniml preprocessor

Usage

```
validate_uniml_preprocessor(x)
```

Arguments

`x`	A `uniml_preprocessor` object.

Value

Invisibly returns `TRUE` when valid.

Examples

```
train <- data.frame(x = c(1, NA, 3), y = c(0, 1, 0))
validate_uniml_preprocessor(fit_preprocessor(train, outcome = "y"))
```

`validate_uniml_result` *Validate a uniml result*

Description

Validate a uniml result

Usage

```
validate_uniml_result(x)
```

Arguments

`x`	A `uniml_result` object.

Value

Invisibly returns TRUE when valid.

Examples

```
validate_uniml_result(assemble_result())
```

`validate_uniml_spec` *Validate a uniml specification*

Description

Validate a uniml specification

Usage

```
validate_uniml_spec(x)
```

Arguments

x	A uniml_spec object.

Value

Invisibly returns TRUE when valid.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
validate_uniml_spec(spec)
```

`validate_uniml_split` *Validate a uniml split*

Description

Validate a uniml split

Usage

```
validate_uniml_split(x)
```

Arguments

x	A uniml_split object.

Value

Invisibly returns TRUE when valid.

Examples

```
spec <- uniml_spec("binary", "y", "yes")
data <- data.frame(
  y = factor(rep(c("no", "yes"), 10), levels = c("no", "yes")),
  x1 = stats::rnorm(20)
)
split <- make_split(as_uniml_data(data, spec), prop = c(0.5, 0.5), seed = 1)
validate_uniml_split(split)
```

---

validate_uniml_task_type

Validate a task-type descriptor

Description

Validate a task-type descriptor

Usage

```
validate_uniml_task_type(x)
```

Arguments

x	A uniml_task_type object.

Value

Invisibly returns TRUE when valid.

Examples

```
validate_uniml_task_type(get_task_type("binary"))
```

workflow_adapter *Get the workflow adapter of a task type*

Description

Get the workflow adapter of a task type

Usage

```
workflow_adapter(task_type = "binary")
```

Arguments

task_type	Task-type id, such as "binary".

Value

The registered adapter list.

Examples

```
workflow_adapter("binary")$task_type
```

workflow_engine *Fit a task type's models and predict the evaluation sets*

Description

The shared half of a workflow. Given a specification and the data sets, it prepares the data, fits imputation, preprocessing, and feature selection, tunes and trains one model per learner, and predicts the training, validation, test, and external sets.

Usage

```
workflow_engine(
  spec,
  train,
  validation,
  test = NULL,
  external = NULL,
  adapter = workflow_adapter(spec$task_type),
  learners = NULL
)
```

Arguments

spec	A uniml_spec object.
train	Training data frame.
validation	Validation data frame.
test	Optional test data frame; reporting-only.
external	Optional external data frame; reporting-only.
adapter	Workflow adapter; defaults to the one registered for spec$task_type.
learners	Optional resolved learner list, used when the task type's learners are not resolvable by resolve_learners().

Details

It stops there. Turning predictions into numbers is the caller's job, because that is where task types differ; see register_workflow_adapter() and run_workflow(), which is the binary report on top of this engine.

Value

A list with the prepared data (prepared, train_data), the fitted preprocessor, the selection, the mlr3 task, the learners, one fitted state per learner in states, the tuning results, and the failures log. Each state holds tuned, learner, task, preprocessor, selection, imputation, and values, where values[[set]] is list(value, truth, prediction).

Examples

```
## Not run:
spec <- uniml_spec(task_type = "binary", outcome = "y", positive = "yes")
engine <- workflow_engine(spec, train, validation)
names(engine$states)

## End(Not run)
```

Index

apply_balance, 4
apply_encode, 5
apply_imputer, 6
apply_preprocessor, 6
apply_preprocessor(), *6*, *36*
apply_scale, 7
apply_step, 8
as_uniml_binary, 9
as_uniml_data, 9
as_uniml_data(), *40*
as_uniml_multiclass, 10
assemble_result, 11

balance_classes, 12

calc_ci, 13
calc_metrics
(calc_metrics.uniml_multiclass), 15
calc_metrics.uniml_multiclass, 15
calc_nri_idi, 17
calibrate_probabilities
(calibrate_probabilities.matrix), 18
calibrate_probabilities.matrix, 18
calibrate_probabilities.uniml_multiclass, 20
calibrate_probabilities.uniml_multiclass(), *18*
calibration_metrics, 22
combine_selections, 23

dca, 23
dca.numeric, 24
delong_test, 25
detect_var_types, 26
diagnose_missing, 27

encode_categorical, 27
explain_shap (explain_shap.Learner), 28
explain_shap.Learner, 28
export_results, 30
extract_thresholds, 31

find_threshold, 32
fit_balance, 33
fit_balance(), *89*
fit_encode, 34
fit_encode(), *89*
fit_imputer, 35
fit_imputer(), *39*, *89*
fit_preprocessor, 36
fit_preprocessor(), *13*, *27*, *75*
fit_scale, 37
fit_scale(), *89*

get_task_type, 38

hl_test, 38

impute_data, 39

list_task_types, 40

make_split, 40
make_task, 41
mlr3::PredictionClassif, *9*, *10*
mlr3tuning::auto_tuner(), *95*
multiclass_calibration, 42
multiclass_calibration(), *45*

new_uniml_result, 43

plot.uniml_result, 44
plot_calibration
(plot_calibration.uniml_multiclass), 44
plot_calibration.uniml_multiclass, 44
plot_confusion
(plot_confusion.uniml_multiclass), 46

plot_confusion.uniml_multiclass, 46
plot_dca *(plot_dca.uniml_multiclass)*, 48
plot_dca.uniml_multiclass, 48
plot_pr_curve
(plot_pr_curve.uniml_result), 49
plot_pr_curve.uniml_result, 49
plot_roc *(plot_roc.uniml_multiclass)*, 51
plot_roc(), *49*
plot_roc.uniml_multiclass, 51
plot_shap_beeswarm
(plot_shap_beeswarm.uniml_result), 53
plot_shap_beeswarm.uniml_result, 53
plot_shap_dependence
(plot_shap_dependence.uniml_result), 54
plot_shap_dependence.uniml_result, 54
plot_shap_force
(plot_shap_force.uniml_result), 56
plot_shap_force.uniml_result, 56
plot_shap_importance
(plot_shap_importance.uniml_result), 57
plot_shap_importance.uniml_result, 57
plot_shap_waterfall
(plot_shap_waterfall.uniml_result), 59
plot_shap_waterfall.uniml_result, 59
pool_mi, 61
predict.uniml_result *(predict_uniml)*, 62
predict_uniml, 62
predict_uniml(), *74*
print.uniml_calibration, 62
print.uniml_data, 63
print.uniml_preprocessor, 64
print.uniml_result, 64
print.uniml_shap, 65
print.uniml_spec, 65
print.uniml_split, 66
print.uniml_task_type, 66

rank_models, 67
rank_models(), *91*
read_result *(save_result)*, 73
register_learner, 68
register_task_type, 68
register_workflow_adapter, 69
register_workflow_adapter(), *72*, *102*
remove_collinear, 70
resolve_learners, 71
resolve_learners(), *95*, *102*
run_workflow, 72
run_workflow(), *94*, *102*

save_result, 73
scale_numeric, 74
select_boruta, 75
select_features, 76
select_lasso, 77
select_multivariate, 78
select_rf_importance, 79
select_rfe, 80
select_univariate, 81
set_evaluation, 82
set_explain, 83
set_feature_selection, 84
set_labels, 85
set_leakage_policy, 86
set_learners, 86
set_output, 87
set_preprocess, 88
set_tuning, 89
summary.uniml_result, 90

task_type_spec, 91
task_type_spec(), *69*
theme_uniml, 92
train_model, 93
train_model(), *70*
tripod_ai_check, 94
tune_model, 95
tune_model(), *70*

uniml_spec, 96

validate_uniml_data, 97
validate_uniml_preprocessor, 98
validate_uniml_result, 98
validate_uniml_spec, 99
validate_uniml_split, 99
validate_uniml_task_type, 100

workflow_adapter, 101
workflow_engine, 101
workflow_engine(), *69*, *72*