

# Package ‘ml’

March 19, 2026

**Title** Supervised Learning with Mandatory Splits and Seeds

**Version** 0.1.2

**Description** Implements the split-fit-evaluate-assess workflow from Hastie, Tibshirani, and Friedman (2009, ISBN:978-0-387-84857-0) ``The Elements of Statistical Learning'', Chapter 7. Provides three-way data splitting with automatic stratification, mandatory seeds for reproducibility, automatic data type handling, and 10 algorithms out of the box. Uses 'Rust' backend for cross-language deterministic splitting. Designed for tabular supervised learning with minimal ceremony. Polyglot parity with the 'Python' 'mlw' package on 'PyPI'.

**License** MIT + file LICENSE

**SystemRequirements** Cargo ('Rust' package manager), rustc (>= 1.56.0, optional)

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**URL** <https://github.com/epagogy/ml>, <https://epagogy.ai>

**BugReports** <https://github.com/epagogy/ml/issues>

**Depends** R (>= 4.1.0)

**Imports** cli, rlang, stats, utils, withr

**Suggests** testthat (>= 3.0.0), xgboost (>= 2.0.0), ranger, rpart, e1071, kkn, glmnet, naivebayes, lightgbm, tm, tibble, knitr, rmarkdown, caret, rsample

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** Simon Roth [aut, cre]

**Maintainer** Simon Roth <[simon@epagogy.ai](mailto:simon@epagogy.ai)>

**Repository** CRAN

**Date/Publication** 2026-03-19 14:30:02 UTC

## Contents

ml	3
ml_algorithms	4
ml_assess	4
ml_best	5
ml_calibrate	5
ml_check	6
ml_check_data	7
ml_compare	8
ml_config	8
ml_cv	9
ml_cv_group	10
ml_cv_temporal	10
ml_dataset	11
ml_drift	12
ml_embed	13
ml_enough	14
ml_evaluate	15
ml_explain	16
ml_fit	16
ml_leak	17
ml_load	18
ml_optimize	19
ml_plot	20
ml_predict	21
ml_predict_proba	21
ml_prepare	22
ml_profile	23
ml_quick	23
ml_report	24
ml_save	25
ml_screen	25
ml_shelf	27
ml_split	28
ml_split_group	29
ml_split_temporal	30
ml_stack	31
ml_tune	32
ml_validate	33
ml_verify	34
predict.ml_model	34
predict.ml_tuning_result	35
print.ml_cv_result	36
print.ml_drift_result	36
print.ml_embedder	37
print.ml_evidence	37
print.ml_explanation	38

<i>ml</i>	3
print.ml_leaderboard . . . . .	38
print.ml_metrics . . . . .	39
print.ml_model . . . . .	39
print.ml_profile_result . . . . .	40
print.ml_shelf_result . . . . .	40
print.ml_split_result . . . . .	41
print.ml_tuning_result . . . . .	41
print.ml_validate_result . . . . .	42
<b>Index</b>	<b>43</b>

---

<code>ml</code>	<i>The ml module — all verbs accessed via ml\$verb()</i>
-----------------	--

---

## Description

Provides the module-style interface `ml$verb()` as an alternative to the standard `ml_verb()` function style. Both styles are equivalent and call the same underlying implementation.

## Usage

`ml`

## Format

A locked environment with verb entries.

## Details

Note: `ml$fit(...)` and `ml_fit(...)` produce identical results.

## Value

A locked environment providing module-style access to all ml verbs.

## Examples

```
s <- ml$split(iris, "Species", seed = 42)
model <- ml$fit(s$train, "Species", seed = 42)
ml$evaluate(model, s$valid)
```

---

ml_algorithms	<i>List available ML algorithms</i>
---------------	-------------------------------------

---

**Description**

Returns a data.frame showing which algorithms support classification and regression, and which require optional packages.

**Usage**

```
ml_algorithms(task = NULL)
```

**Arguments**

task                   Optional filter: "classification" or "regression"

**Value**

A data.frame with columns: algorithm, classification, regression, optional\_dep, installed

**Examples**

```
ml_algorithms()
ml_algorithms(task = "classification")
```

---

ml_assess	<i>Assess model on held-out test data (do once)</i>
-----------	---

---

**Description**

The final exam — separate from `ml_evaluate()` to force a conscious choice. Errors if called more than once on the same model. Use `s$test` (not `s$valid`) for the test data.

**Usage**

```
ml_assess(model, test)
```

**Arguments**

model                   An ml\_model  
test                    Test data.frame (use `s$test`). Specify by name for clarity.

**Value**

An object of class `ml_evidence` (sealed — not substitutable for `ml_metrics`)

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
verdict <- ml_assess(model, test = s$test)
```

---

ml_best	<i>Get the best model from a leaderboard</i>
---------	--

---

**Description**

Returns the top-ranked fitted model from `screen()` or `compare()`. NULL if no models were stored.

**Usage**

```
ml_best(lb)
```

**Arguments**

lb                    An ml\_leaderboard

**Value**

An ml\_model or NULL

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
lb <- ml_screen(s, "Species", seed = 42)
best <- ml_best(lb)
predict(best, s$valid)
```

---

ml_calibrate	<i>Calibrate predicted probabilities</i>
--------------	--

---

**Description**

Applies Platt scaling (logistic regression on raw probabilities) to produce better-calibrated class probability estimates. Use validation data for calibration – never training data.

**Usage**

```
ml_calibrate(model, data = NULL)
```

**Arguments**

model	An ml_model from ml_fit()
data	A data.frame of calibration data (use validation set)

**Details**

Binary classification only.

**Value**

An ml\_calibrated\_model that behaves like an ml\_model but returns calibrated probabilities

**Examples**

```
s <- ml_split(ml_dataset("cancer"), "target", seed = 42)
model <- ml_fit(s$train, "target", algorithm = "xgboost", seed = 42)
cal <- ml_calibrate(model, data = s$valid)
ml_evaluate(cal, s$valid)
```

---

ml\_check

*Verify bitwise reproducibility for a given dataset*


---

**Description**

Fits the same model twice with the same seed and asserts predictions are identical. Returns a list with passed, algorithm, seed, and message.

**Usage**

```
ml_check(data, target, algorithm = "random_forest", seed)
```

**Arguments**

data	A data.frame with features and target
target	Target column name
algorithm	Algorithm to check (default "random_forest")
seed	Random seed

**Value**

A list with passed (logical), algorithm, seed, message. Supports isTRUE(result\$passed) for assertions.

**Examples**

```
result <- ml_check(iris, "Species", seed = 42)
result$passed
```

---

ml_check_data	<i>Pre-flight data quality checks</i>
---------------	---------------------------------------

---

## Description

Runs before fit() to catch common data quality issues that silently degrade model performance.

## Usage

```
ml_check_data(data, target, severity = "warn")
```

## Arguments

data	A data.frame
target	Target column name
severity	"warn" (default) or "error". If "error", raises on any issue.

## Details

Checks performed:

- NaN in target (silently dropped by split)
- Inf in features
- ID columns (100\)
- Zero-variance features (constant columns)
- High-null columns (>50\)
- Severe class imbalance (<5\)
- Duplicate rows (>10\)
- Feature redundancy ( $|r| > 0.95$ )

## Value

A list with warnings, errors, has\_issues, passed. Supports isTRUE(result\$passed) for assertions.

## Examples

```
report <- ml_check_data(iris, "Species")
report$passed
```

---

ml_compare	<i>Compare pre-fitted models on the same data</i>
------------	---

---

**Description**

Evaluates multiple fitted models on the same dataset without re-fitting. All models must share the same target column and task.

**Usage**

```
ml_compare(models, data, sort_by = "auto")
```

**Arguments**

models	A list of ml_model objects (or ml_tuning_result, auto-unwrapped)
data	A data.frame containing the target column
sort_by	"auto" or a metric name string

**Value**

An object of class ml\_leaderboard (data.frame with formatted print)

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
m1 <- ml_fit(s$train, "Species", algorithm = "logistic", seed = 42)
m2 <- ml_fit(s$train, "Species", algorithm = "random_forest", seed = 42)
ml_compare(list(m1, m2), s$valid)
```

---

ml_config	<i>Configure ml package settings</i>
-----------	--------------------------------------

---

**Description**

Set global configuration for the ml package. Currently supports guards to control partition enforcement.

**Usage**

```
ml_config(guards = NULL)
```

**Arguments**

guards	Character: "strict" (default) enforces split provenance — all verbs reject data not produced by ml_split(). "warn" issues warnings instead of errors (useful for migration). "off" disables guards for exploration/education.
--------	---

**Value**

Invisibly returns the previous settings as a list.

**Examples**

```
ml_config(guards = "off") # disable guards
ml_config(guards = "warn") # warn instead of error
ml_config(guards = "strict") # re-enable (default)
```

---

ml\_cv

---

*Create k-fold cross-validation from a split*


---

**Description**

Takes an existing `ml_split_result` and creates k-fold rotations within its dev partition (train + valid). The test partition stays sealed on the original split for `ml_assess()`.

**Usage**

```
ml_cv(s, target, folds = 5L, seed = NULL, stratify = TRUE)
```

**Arguments**

<code>s</code>	An <code>ml_split_result</code> from <code>ml_split()</code>
<code>target</code>	Target column name (string)
<code>folds</code>	Number of folds (default 5)
<code>seed</code>	Random seed for fold assignment
<code>stratify</code>	Logical. Stratify folds by target for classification (default TRUE)

**Details**

Two primitives, strict separation of concerns: `ml_split()` creates the three-way boundary, `ml_cv()` creates rotations within that boundary.

**Value**

An `ml_cv_result` that `ml_fit()` accepts directly. The original split's `$test` remains available via `s$test` for `ml_assess()`.

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
c <- ml_cv(s, "Species", folds = 5, seed = 42)
model <- ml_fit(c, "Species", seed = 42)
model$scores_
```

---

ml_cv_group	<i>Create group-aware cross-validation from a split</i>
-------------	---

---

**Description**

No group appears in both train and validation within any fold. Prevents leakage from repeated measurements (patients, stores, sensors).

**Usage**

```
ml_cv_group(s, target, groups, folds = 5L, seed = NULL)
```

**Arguments**

s	An ml_split_result from ml_split() or ml_split_group()
target	Target column name (string)
groups	Column name identifying groups
folds	Number of folds (default 5)
seed	Random seed for group assignment

**Value**

An ml\_cv\_result with group-aware folds

**Examples**

```
df <- data.frame(pid = rep(1:20, each = 5), x = rnorm(100), y = sample(0:1, 100, TRUE))
s <- ml_split(df, "y", seed = 42)
c <- ml_cv_group(s, "y", groups = "pid", folds = 5, seed = 42)
```

---

ml_cv_temporal	<i>Create temporal cross-validation from a split</i>
----------------	--

---

**Description**

Expanding-window CV for time series. Data must already be sorted chronologically (use ml\_split\_temporal() first).

**Usage**

```
ml_cv_temporal(
  s,
  target,
  folds = 5L,
  embargo = 0L,
  window = "expanding",
  window_size = NULL
)
```

**Arguments**

s	An ml_split_result from ml_split_temporal()
target	Target column name (string)
folds	Number of folds (default 5)
embargo	Integer. Number of rows to skip between train end and valid start (gap to prevent temporal leakage from autocorrelation). Default 0. Must be $\geq 0$ .
window	"expanding" (default, all prior rows as train) or "sliding" (fixed-size training window). When "sliding", window_size must also be supplied.
window_size	Integer. Required when window = "sliding". Number of rows in each training window.

**Value**

An ml\_cv\_result with expanding-window folds

**Examples**

```
df <- data.frame(date = 1:100, x = rnorm(100), y = sample(0:1, 100, TRUE))
s <- ml_split_temporal(df, "y", time = "date")
c <- ml_cv_temporal(s, "y", folds = 5)
# With embargo to prevent autocorrelation leakage:
c2 <- ml_cv_temporal(s, "y", folds = 5, embargo = 5L)
```

---

ml\_dataset

*Load a built-in dataset*


---

**Description**

Returns one of the built-in datasets. Useful for experimenting with the ml API before applying it to your own data.

**Usage**

```
ml_dataset(name, seed = 42L)
```

**Arguments**

name	Dataset name (string)
seed	Random seed for synthetic datasets (default 42)

**Details**

Available datasets: "iris", "wine", "cancer", "diabetes", "houses", "churn", "fraud"

**Value**

A data.frame

**Examples**

```
churn <- ml_dataset("churn")
head(churn)
```

---

ml\_drift

*Detect data drift between reference and new data*


---

**Description**

Compares a reference dataset (typically training data) to new data using per-feature statistical tests or adversarial validation.

**Usage**

```
ml_drift(
  reference,
  new,
  method = "statistical",
  threshold = 0.05,
  exclude = NULL,
  target = NULL,
  seed = NULL,
  algorithm = "random_forest"
)
```

**Arguments**

reference	A data.frame — reference dataset (typically training data)
new	A data.frame — new data to compare against the reference
method	Detection method: "statistical" (default) or "adversarial"
threshold	p-value threshold for statistical method (default 0.05)
exclude	Character vector of column names to skip (e.g., ID columns)
target	Target column name — automatically excluded from drift analysis
seed	Random seed (required for method = "adversarial")
algorithm	Algorithm for adversarial classifier: "random_forest" (default) or "xgboost"

**Details**

**Statistical method** (default): per-feature distribution tests with no labels required.

- Numeric features: Kolmogorov-Smirnov two-sample test
- Categorical features: Chi-squared test on value counts

**Adversarial method**: trains a binary classifier to distinguish reference from new data. AUC near 0.5 means similar distributions; AUC near 1.0 means very different distributions.

- `$train_scores`: per-row probability of "looks like new data" for reference rows. Use `sort(result$train_scores, decreasing = TRUE)[1:n]` to select validation rows that mirror the new distribution.
- `$features`: most discriminative features (temporal leakage candidates)

Pair with `ml_shelf()` for complete monitoring: `drift()` detects input distribution shift (label-free), `shelf()` detects performance degradation (requires labels).

### Value

An object of class `ml_drift_result` with:

- `$shifted`: TRUE if drift detected
- `$features`: named numeric — p-values (statistical) or importances (adversarial)
- `$features_shifted`: character vector of drifted feature names
- `$severity`: "none", "low", "medium", or "high"
- `$auc`: adversarial mode only — classifier AUC
- `$train_scores`: adversarial mode only — per-row reference probabilities

### Examples

```
s <- ml_split(iris, "Species", seed = 42)
# Simulate drift by perturbing test data
new <- s$test
new$Sepal.Length <- new$Sepal.Length + 2
result <- ml_drift(reference = s$train, new = new, target = "Species")
result$shifted
result$features_shifted
```

---

ml\_embed

*Embed texts into numeric features*

---

### Description

Fits a text vectorizer on training texts and returns an embedder object that stores the vocabulary for consistent transform at prediction time.

### Usage

```
ml_embed(texts, method = "tfidf", max_features = 100L)
```

### Arguments

<code>texts</code>	A character vector of texts to embed
<code>method</code>	Embedding method. Currently only "tfidf" is supported.
<code>max_features</code>	Maximum vocabulary size (number of TF-IDF features). Default 100.

**Details**

Currently supports TF-IDF ('tm' package). SBERT and neural methods are planned for future gates.

**Value**

An object of class `ml_embedder` with:

- `$vectors`: data.frame of TF-IDF features (`n_texts` x `max_features`)
- `$method`: the method used
- `$vocab_size`: number of features generated
- `$transform(new_texts)`: apply stored vocabulary to new texts

**Examples**

```
if (requireNamespace("tm", quietly = TRUE)) {
  texts <- c("good product", "bad service", "great value", "poor quality")
  emb <- ml_embed(texts, method = "tfidf", max_features = 20)
  emb$vocab_size
  nrow(emb$vectors)

  # Transform new texts using the fitted vocabulary
  new_texts <- c("excellent quality", "terrible service")
  new_vecs <- emb$transform(new_texts)
}
```

---

ml\_enough

*Learning curve analysis – do you need more data?*


---

**Description**

Trains at increasing data sizes and reports train vs validation performance at each step. Answers: is the model still learning (more data helps), or saturated (more data unlikely to help)?

**Usage**

```
ml_enough(s, target, seed = NULL, algorithm = "auto", steps = 8L, cv = 3L)
```

**Arguments**

<code>s</code>	An <code>ml_split_result</code> from <code>ml_split()</code>
<code>target</code>	Target column name
<code>seed</code>	Random seed (optional in R; auto-generated if NULL)
<code>algorithm</code>	Algorithm to use (default "auto"). Any algorithm supported by <code>ml_fit()</code> .
<code>steps</code>	Integer $\geq 2$ . Number of data-size steps to evaluate, evenly spaced from ~10%% to 100%% of training data. Default 8.
<code>cv</code>	Integer $\geq 2$ . Number of cross-validation folds for validation score at each step. Default 3.

**Value**

An `ml_enough_result` with fields:

- `$saturated` – logical, TRUE if curve plateaus (< 1%% gain in last half)
- `$curve` – data.frame: `n_samples`, `train_score`, `val_score`
- `$metric` – metric name used
- `$n_current` – total training rows in the full dataset
- `$recommendation` – human-readable action

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
result <- ml_enough(s, "Species", seed = 42)
result$recommendation
```

---

ml\_evaluate

*Evaluate model on validation data (iterate freely)*

---

**Description**

The practice exam — call as many times as needed. For the one-time final grade on held-out test data, use [ml\\_assess\(\)](#).

**Usage**

```
ml_evaluate(model, data)
```

**Arguments**

model	An <code>ml_model</code> or <code>ml_tuning_result</code>
data	A data.frame containing the target column

**Value**

An object of class `ml_metrics` (named numeric vector with print method)

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
metrics <- ml_evaluate(model, s$valid)
metrics[["accuracy"]]
```

---

ml_explain	<i>Explain model via feature importance</i>
------------	---

---

**Description**

Returns a data frame of feature importances, normalized to sum to 1.0, sorted descending. Uses tree-based impurity importance for 'xgboost' and 'random\_forest', absolute coefficients for 'logistic', 'linear', and 'elastic\_net'. Not supported for 'svm' or 'knn'.

**Usage**

```
ml_explain(model)
```

**Arguments**

model	An ml_model or ml_tuning_result
-------	---------------------------------

**Value**

An object of class ml\_explanation (a data.frame with columns feature and importance; custom print shows a bar chart)

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", algorithm = "random_forest", seed = 42)
ml_explain(model)
```

---

ml_fit	<i>Fit a machine learning model</i>
--------	-------------------------------------

---

**Description**

Trains a model using cross-validation (if data is an ml\_split\_result with folds) or holdout (if data is a data.frame). Automatically detects task type, handles encoding, and records metadata for reproducibility.

**Usage**

```
ml_fit(
  data,
  target,
  algorithm = "auto",
  seed = NULL,
  task = "auto",
  balance = FALSE,
```

```

    engine = "auto",
    ...
  )

```

### Arguments

data	A data.frame, ml_split_result, or ml_split_result with folds
target	Target column name (string)
algorithm	"auto" (default), "xgboost", "random_forest", "svm", "knn", "logistic", "linear", "naive_bayes", "elastic_net"
seed	Random seed. NULL (default) auto-generates and stores for reproducibility.
task	"auto", "classification", or "regression"
balance	Logical. If TRUE, applies class-weight balancing for imbalanced classification problems. Ignored for regression. Default: FALSE.
engine	Backend engine: "auto" (Rust if available, else CRAN packages), "ml" (Rust required), or "r" (CRAN packages only). Default: "auto".
...	Additional hyperparameters passed to the engine (e.g., max_depth = 6, num. trees = 200)

### Details

Formula interfaces are not supported. Pass the data frame and target column name as a string. Unordered factors use one-hot encoding for linear models and ordinal encoding for tree-based models. Ordered factors always use ordinal encoding.

### Value

An object of class ml\_model

### Examples

```

s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
model$algorithm

```

---

ml\_leak

*Detect potential data leakage*


---

### Description

Analyzes feature-target relationships before modeling. Runs pure data introspection – no model fitting.

### Usage

```
ml_leak(data, target)
```

**Arguments**

data            A data.frame or ml\_split\_result  
target         Target column name

**Details**

Checks performed:

1. Feature-target correlation (Pearson lrl, numeric features)
2. High-cardinality ID columns
3. Target name in feature names
4. Duplicate rows between train and test (SplitResult only)

**Value**

A list with clean (logical), n\_warnings, checks (list of check results), suspects (list of suspect features). Class ml\_leak\_report.

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
report <- ml_leak(s, "Species")
report$clean
```

---

ml\_load                    *Load a model from disk*

---

**Description**

Load a model from disk

**Usage**

```
ml_load(path)
```

**Arguments**

path            Path to a .mlr file saved with [ml\\_save\(\)](#)

**Value**

An ml\_model or ml\_tuning\_result

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
path <- file.path(tempdir(), "iris_model.ml")
ml_save(model, path)
loaded <- ml_load(path)
loaded$algorithm
```

ml\_optimize

*Optimize decision threshold for binary classification***Description**

Sweeps thresholds from `min_threshold` to 0.95 in two phases (coarse 0.05 steps, then fine 0.005 steps around the coarse best) and returns a copy of the model with a tuned threshold. Subsequent `ml_predict()` calls apply this threshold to positive-class probability instead of 0.5.

**Usage**

```
ml_optimize(model, data, metric = "f1", min_threshold = "auto")
```

**Arguments**

<code>model</code>	An <code>ml_model</code> or <code>ml_tuning_result</code> (binary classification only).
<code>data</code>	A <code>data.frame</code> containing the target column used as true labels.
<code>metric</code>	Character. Optimisation objective: "f1", "accuracy", "precision", or "recall". Ranking metrics ("roc_auc", "log_loss") are rejected because they are threshold-independent.
<code>min_threshold</code>	Lower bound of the sweep. "auto" (default) computes $\max(0.001, 1 / n_{\text{positives}})$ – the minimum meaningful threshold for imbalanced data. Pass a numeric value to override.

**Value**

An `ml_optimize_result` (also an `ml_model`). The threshold is baked in – every `ml_predict()` call uses it automatically. Inspect with `result$threshold`. The original model is unchanged.

**Examples**

```
s <- ml_split(iris[iris$Species != "virginica", ], "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
opt <- ml_optimize(model, data = s$valid, metric = "f1")
opt$threshold
```

---

`ml_plot`*Visual diagnostics for a fitted model*

---

**Description**

Produces diagnostic plots using base R graphics. No extra packages required.

**Usage**

```
ml_plot(model, data = NULL, kind = "importance", ...)
```

**Arguments**

<code>model</code>	An <code>ml_model</code> from <code>ml_fit()</code>
<code>data</code>	A <code>data.frame</code> for computing predictions (required for all except "importance")
<code>kind</code>	Plot type. One of "importance", "roc", "confusion", "residual", "calibration". Default: "importance".
<code>...</code>	Passed to the underlying base R plot call

**Details**

Available kinds:

- "importance" — feature importance bar chart
- "roc" — ROC curve (classification)
- "confusion" — confusion matrix heatmap (classification)
- "residual" — residuals vs fitted (regression)
- "calibration" — predicted vs actual probabilities (classification)

**Value**

Invisibly returns `NULL` (called for its side effect)

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", algorithm = "random_forest", seed = 42)
ml_plot(model, kind = "importance")
ml_plot(model, data = s$valid, kind = "confusion")
```

---

ml_predict	<i>Predict from a fitted model (ml_predict style)</i>
------------	---

---

**Description**

Alias for `predict(model, newdata = ...)`. Matches Python `ml.predict()`.

**Usage**

```
ml_predict(model, new_data)
```

**Arguments**

model	An <code>ml_model</code> or <code>ml_tuning_result</code>
new_data	A <code>data.frame</code> with the same features used for training

**Value**

A vector of predicted class labels (classification) or numeric values (regression).

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
preds <- ml_predict(model, s$valid)
head(preds)
```

---

ml_predict_proba	<i>Predict class probabilities</i>
------------------	------------------------------------

---

**Description**

Predict class probabilities

**Usage**

```
ml_predict_proba(model, new_data)
```

**Arguments**

model	An <code>ml_model</code> object (classification only)
new_data	A <code>data.frame</code> with the same features used for training

**Value**

A `data.frame` with one column per class. Values are probabilities summing to 1.0 per row. Column names are the original class labels.

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", algorithm = "random_forest", seed = 42)
probs <- ml_predict_proba(model, s$valid)
head(probs)
```

---

ml\_prepare

*Prepare data for ML: encode, impute, and scale*


---

**Description**

Grammar primitive #2: DataFrame -> PreparedData.

**Usage**

```
ml_prepare(data, target, algorithm = "auto", task = "auto")
```

**Arguments**

data	A data.frame including the target column.
target	Name of the target column (string).
algorithm	Algorithm hint for encoding strategy: "auto", "random_forest", "logistic", etc. Tree-based algorithms use ordinal encoding; linear algorithms use one-hot encoding for low-cardinality categoricals.
task	"classification", "regression", or "auto" (detected from target).

**Details**

In the default workflow, `ml_fit()` calls preparation internally per fold. Use `ml_prepare()` explicitly when you need manual control: inspect the preprocessing state, apply the same encoding to external data, or chain preparation with fitting.

**Value**

An `ml_prepared_data` object with:

- `$data` — transformed data.frame (all-numeric, ready for `ml_fit`)
- `$state` — NormState list; use `.transform(state, X)` on new data
- `$target` — target column name
- `$task` — detected or provided task type

**Examples**

```
df <- data.frame(x1 = rnorm(50), x2 = rnorm(50), y = rnorm(50))
s <- ml_split(df, "y", seed = 42)
p <- ml_prepare(s$train, "y")
p$task      # "classification" or "regression"
p$data      # encoded feature matrix
```

---

ml_profile	<i>Profile data before modeling</i>
------------	-------------------------------------

---

**Description**

Computes per-column statistics and emits warnings for common data quality issues: missing values, constant columns, high cardinality, imbalanced targets, and near-collinear features.

**Usage**

```
ml_profile(data, target = NULL)
```

**Arguments**

data	A data.frame (also accepts tibble or data.table)
target	Optional target column name (enables task detection + distribution stats)

**Value**

An object of class `ml_profile_result` (list with formatted print)

**Examples**

```
ml_profile(iris, "Species")
```

---

ml_quick	<i>One-call workflow: split + screen + fit + evaluate</i>
----------	---

---

**Description**

The fastest path from raw data to a trained, evaluated model. Screens logistic, random\_forest, and xgboost, picks the best, fits on training data, and evaluates on validation.

**Usage**

```
ml_quick(data, target, seed)
```

**Arguments**

data	A data.frame with features and target
target	Target column name
seed	Random seed

**Value**

A list with model (ml\_model), metrics (ml\_metrics), and split (ml\_split\_result).

**Examples**

```
result <- ml_quick(iris, "Species", seed = 42)
result$model
result$metrics
```

---

ml_report	<i>Generate an HTML training report</i>
-----------	---

---

**Description**

Produces a self-contained HTML report with model metadata, evaluation metrics, and feature importances. Open in any browser.

**Usage**

```
ml_report(model, data = NULL, path = "model_report.html")
```

**Arguments**

model	An ml_model from ml_fit()
data	A data.frame for computing metrics (use validation data)
path	Output file path. Default: "model_report.html"

**Value**

The path to the saved report (invisibly)

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", algorithm = "random_forest", seed = 42)
tmp <- tempfile(fileext = ".html")
ml_report(model, data = s$valid, path = tmp)
unlink(tmp)
```

---

ml_save	<i>Save a model to disk</i>
---------	-----------------------------

---

**Description**

Saves an `ml_model` or `ml_tuning_result` to a `.mlr` file using `saveRDS` with a version wrapper.

**Usage**

```
ml_save(model, path)
```

**Arguments**

model	An <code>ml_model</code> or <code>ml_tuning_result</code>
path	File path (recommended extension: <code>.mlr</code> )

**Value**

The normalized path, invisibly.

**Security**

`ml_load()` uses `readRDS()` internally, which can execute arbitrary R code during deserialization. Never load `.mlr` files from untrusted sources.

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
path <- file.path(tempdir(), "iris_model.mlr")
ml_save(model, path)
loaded <- ml_load(path)
```

---

ml_screen	<i>Screen all algorithms on your data</i>
-----------	---

---

**Description**

Fits every available algorithm on the training data and ranks by validation performance. Use this to identify promising candidates before tuning.

**Usage**

```
ml_screen(
  data,
  target,
  algorithms = NULL,
  seed = NULL,
  sort_by = "auto",
  time_budget = NULL,
  keep_models = TRUE,
  ...
)
```

**Arguments**

data	An <code>ml_split_result</code> (NOT a raw <code>data.frame</code> — split first to prevent overfitting)
target	Target column name
algorithms	Character vector of algorithm names, or <code>NULL</code> for all available
seed	Random seed. <code>NULL</code> auto-generates.
sort_by	"auto" ( <code>roc_auc</code> for binary clf, <code>f1_macro</code> for multiclass, <code>rmse</code> for regression), or a metric name string
time_budget	Maximum seconds for entire screen. Stops between algorithms (not mid-fit) when budget exceeded. <code>NULL</code> (default) = no limit.
keep_models	If <code>FALSE</code> , discard fitted models after scoring to save memory. <code>ml_best()</code> will return <code>NULL</code> . Default <code>TRUE</code> .
...	Additional arguments passed to <code>ml_fit()</code>

**Details**

**Multiple comparison bias:** Selecting the best from `N` algorithms on the same validation set produces optimistic estimates. The winning algorithm benefits from selection bias. Use `ml_validate()` on held-out test data for trustworthy comparisons.

For imbalanced data, consider `sort_by = "f1"` — the default `roc_auc` can hide failures on minority classes.

**Value**

An object of class `ml_leaderboard` (`data.frame` with formatted print)

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
lb <- ml_screen(s, "Species", seed = 42)
lb
```

---

ml_shelf	<i>Check if a model is past its shelf life</i>
----------	--

---

### Description

Evaluates the model on new labeled data and compares performance to the model's original training metrics. Requires ground truth labels.

### Usage

```
ml_shelf(model, new, target, tolerance = 0.05)
```

### Arguments

model	An <code>ml_model</code> or <code>ml_tuning_result</code> with <code>\$scores_</code> populated (i.e., trained with <code>ml_fit(cv_result, target)</code> )
new	A <code>data.frame</code> — new labeled dataset including the target column
target	Name of the target column in <code>new</code>
tolerance	Allowed degradation per metric (default 0.05 = 5pp). Any key metric degrading beyond tolerance marks the model as stale.

### Details

Run this when outcome labels become available (e.g., daily/weekly batch scoring, then wait for outcomes). Pair with `ml_drift()` for complete monitoring:

- `ml_drift()`: input distribution shift (label-free, run always)
- `ml_shelf()`: performance degradation (needs labels, run periodically)

**Requires** `model$scores_` from a cross-validated fit. If the model was trained on a holdout split (no CV), `scores_` will be `NULL` and `shelf()` raises a `model_error`.

### Value

An object of class `ml_shelf_result` with:

- `$fresh`: TRUE if model performance is within tolerance
- `$stale`: inverse of fresh
- `$metrics_then`: original training metrics (from `model$scores_`)
- `$metrics_now`: current metrics on new data
- `$degradation`: per-metric delta (negative = worse for higher-is-better)
- `$recommendation`: human-readable guidance

**Examples**

```

cv    <- ml_split(iris, "Species", seed = 42, folds = 3)
model <- ml_fit(cv, "Species", algorithm = "logistic", seed = 42)
# Simulate a new labeled batch
new_batch <- iris[sample(nrow(iris), 30), ]
result <- ml_shelf(model, new = new_batch, target = "Species")
result$fresh
result$degradation

```

ml\_split

*Split data into train/valid/test partitions or cross-validation folds***Description**

Three-way split is the default (60/20/20), following Hastie, Tibshirani, and Friedman (2009, ISBN:978-0-387-84857-0) Chapter 7. Automatically stratifies for classification.

**Usage**

```

ml_split(
  data,
  target = NULL,
  seed = NULL,
  ratio = c(0.6, 0.2, 0.2),
  folds = NULL,
  stratify = TRUE,
  task = "auto",
  time = NULL,
  groups = NULL
)

```

**Arguments**

data	A data.frame (also accepts tibble or data.table)
target	Target column name (enables stratification + task detection)
seed	Random seed. NULL (default) auto-generates and stores for reproducibility. Pass an integer for reproducible splits.
ratio	Numeric vector of length 3: c(train, valid, test). Must sum to 1.0.
folds	Integer for k-fold CV (e.g., folds = 5). Overrides ratio.
stratify	Logical. Auto-stratify for classification targets (default TRUE).
task	"auto", "classification", or "regression". Override task detection.
time	Column name for temporal/chronological split. Data is sorted by this column, and the time column is dropped from output. Deterministic (seed is ignored). Cannot combine with groups.
groups	Column name for group-aware split. No group appears in both train and validation/test. Cannot combine with time.

**Value**

An `ml_split_result`. Access `$train`, `$valid`, `$test`, `$dev` (train + valid). When `fold`s is set, also `$folds` (CV on dev).

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
nrow(s$train)
nrow(s$dev)
```

---

<code>ml_split_group</code>	<i>Split data with group non-overlap — no group leaks across partitions</i>
-----------------------------	---

---

**Description**

Domain specialization of `ml_split()` for clinical trials, repeated measures, and any data where observations are nested within groups (patients, subjects, hospitals). No group appears in more than one partition.

**Usage**

```
ml_split_group(
  data,
  target = NULL,
  groups,
  seed = NULL,
  ratio = c(0.6, 0.2, 0.2),
  folds = NULL,
  stratify = TRUE,
  task = "auto"
)
```

**Arguments**

<code>data</code>	A <code>data.frame</code>
<code>target</code>	Target column name (optional, enables stratification)
<code>groups</code>	Column name identifying groups
<code>seed</code>	Random seed for reproducibility
<code>ratio</code>	Numeric vector <code>c(train, valid, test)</code> . Must sum to 1.0.
<code>folds</code>	Integer for group CV. When set, ignores ratio.
<code>stratify</code>	Logical. Stratify by target within groups (default TRUE).
<code>task</code>	"auto", "classification", or "regression"

**Details**

Also covers Leave-Source-Out CV: when groups represent data sources (hospitals, devices), this produces deployment-realistic evaluation.

**Value**

An `ml_split_result`. When `fold`s is set, includes `$fold`s and `$test`.

**Examples**

```
df <- data.frame(pid = rep(1:10, each = 5), x = rnorm(50), y = sample(0:1, 50, TRUE))
s <- ml_split_group(df, "y", groups = "pid", seed = 42)
nrow(s$train)
```

---

`ml_split_temporal`      *Split data chronologically — no future leakage*

---

**Description**

Domain specialization of `ml_split()` for time series and forecasting. Data is sorted by the time column and partitioned by position. Deterministic: seed is ignored (chronological order is the only order).

**Usage**

```
ml_split_temporal(
  data,
  target = NULL,
  time,
  ratio = c(0.6, 0.2, 0.2),
  folds = NULL,
  task = "auto"
)
```

**Arguments**

<code>data</code>	A <code>data.frame</code>
<code>target</code>	Target column name (optional, enables task detection)
<code>time</code>	Column name containing timestamps or orderable values. Used for sorting, then dropped from output partitions.
<code>ratio</code>	Numeric vector <code>c(train, valid, test)</code> . Must sum to 1.0.
<code>folds</code>	Integer for temporal CV (expanding window). When set, ignores ratio.
<code>task</code>	"auto", "classification", or "regression"

**Value**

An `ml_split_result`. When `fold`s is set, includes `$fold`s and `$test`.

## Examples

```
df <- data.frame(date = 1:100, x = rnorm(100), y = sample(0:1, 100, TRUE))
s <- ml_split_temporal(df, "y", time = "date")
nrow(s$train)
```

---

ml_stack	<i>Ensemble stacking</i>
----------	--------------------------

---

## Description

Trains a stacking ensemble with out-of-fold meta-features. Base models generate out-of-fold predictions, which are used to train a meta-learner.

## Usage

```
ml_stack(data, target, models = NULL, meta = NULL, cv_folds = 5L, seed = NULL)
```

## Arguments

data	A data.frame with features and target
target	Target column name
models	Character vector of base algorithm names, or NULL for defaults
meta	Meta-learner algorithm. Default: "logistic" (classification) or "linear" (regression)
cv_folds	Number of CV folds for generating out-of-fold predictions
seed	Random seed

## Details

**Note:** This function uses global normalization (not per-fold), because the stacking CV is internal to the meta-learner training. This is the one exception to the per-fold normalization rule.

## Value

An ml\_model with \$is\_stacked = TRUE

## Examples

```
s <- ml_split(iris, "Species", seed = 42)
stacked <- ml_stack(s$train, "Species", seed = 42)
predict(stacked, s$valid)
```

---

ml_tune	<i>Tune hyperparameters via random or grid search</i>
---------	---

---

**Description**

Tune hyperparameters via random or grid search

**Usage**

```
ml_tune(
  data,
  target,
  model = NULL,
  algorithm = NULL,
  n_trials = 20L,
  cv_folds = 3L,
  method = "random",
  seed = NULL,
  params = NULL
)
```

**Arguments**

data	A data.frame or ml_split_result
target	Target column name
model	An ml_model object (to clone algorithm from), or NULL
algorithm	Algorithm name (if model is NULL)
n_trials	Number of random search trials (default 20)
cv_folds	Number of CV folds per trial (default 3)
method	"random" (default) or "grid"
seed	Random seed
params	Named list of parameter ranges (overrides defaults). For numeric ranges, provide a 2-element numeric vector c(min, max). For discrete, provide a character/integer vector.

**Value**

An object of class ml\_tuning\_result

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
tuned <- ml_tune(s$train, "Species", algorithm = "xgboost", n_trials = 5, seed = 42)
tuned$best_params_
```

---

ml_validate	<i>Validate model against rules and/or baseline</i>
-------------	---

---

### Description

Three modes: (1) absolute rules, (2) regression prevention vs baseline, (3) combined. Returns a structured result with pass/fail and diagnostics.

### Usage

```
ml_validate(model, test, rules = NULL, baseline = NULL, tolerance = 0)
```

### Arguments

model	An ml_model
test	Test data.frame (use s\$test)
rules	Named list of threshold strings, e.g. <code>list(accuracy = "&gt;0.85", roc_auc = "&gt;=0.90")</code>
baseline	An ml_model — previous model to check for regressions
tolerance	Numeric. Allowed absolute degradation (0.02 = 2pp slack). Default 0.0.

### Details

Tolerance is absolute (not relative): a tolerance of 0.02 means 2 percentage points of allowed degradation, applied uniformly across all metrics.

### Value

An object of class `ml_validate_result`

### Examples

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
gate <- ml_validate(model, test = s$test, rules = list(accuracy = ">0.80"))
gate$passed
```

---

ml_verify	<i>Verify provenance integrity of a model</i>
-----------	---

---

**Description**

Checks provenance chain: split parameters -> training fingerprint -> assess ceremony status. Catches accidental self-deception (load-assess loops, test-set shopping) rather than adversarial tampering.

**Usage**

```
ml_verify(model)
```

**Arguments**

model            An ml\_model, ml\_tuning\_result, or path to .mlr file

**Value**

A list with status ("verified"/"unverified"/"warning"), checks, provenance, and assess\_count.

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
report <- ml_verify(model)
report$status
```

---

predict.ml_model	<i>Predict from a fitted model</i>
------------------	------------------------------------

---

**Description**

Predict from a fitted model

Predict from an ml\_model

**Usage**

```
## S3 method for class 'ml_model'
predict(object, newdata, proba = FALSE, ...)
```

```
## S3 method for class 'ml_model'
predict(object, newdata, proba = FALSE, ...)
```

**Arguments**

object	An ml_model object
newdata	A data.frame
proba	Logical. If TRUE, returns class probabilities (classification only)
...	Ignored

**Value**

A vector of predicted class labels (classification) or numeric values (regression). If proba = TRUE, returns a data.frame with one column per class; values are probabilities summing to 1.0 per row.

Predicted labels (classification) or numeric values (regression). If proba = TRUE, a data.frame of probabilities.

**Examples**

```
s <- ml_split(iris, "Species", seed = 42)
model <- ml_fit(s$train, "Species", seed = 42)
preds <- predict(model, newdata = s$valid)
head(preds)
```

---

predict.ml\_tuning\_result

*Predict from best model in a tuning result*

---

**Description**

Predict from best model in a tuning result

**Usage**

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

**Arguments**

object	An ml_tuning_result
newdata	A data.frame
...	Passed to predict.ml_model

**Value**

Predictions

---

`print.ml_cv_result`     *Print ml\_cv\_result*

---

**Description**

Print `ml_cv_result`

**Usage**

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

**Arguments**

<code>x</code>	An <code>ml_cv_result</code> object
<code>...</code>	Ignored

**Value**

The object `x`, invisibly.

---

`print.ml_drift_result`     *Print ml\_drift\_result*

---

**Description**

Print `ml_drift_result`

**Usage**

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

**Arguments**

<code>x</code>	An <code>ml_drift_result</code> object
<code>...</code>	Ignored

**Value**

The object `x`, invisibly.

---

print.ml\_embedder      *Print ml\_embedder*

---

**Description**

Print ml\_embedder

**Usage**

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

**Arguments**

x	An ml_embedder object
...	Ignored

**Value**

The object x, invisibly.

---

print.ml\_evidence      *Print ml\_evidence*

---

**Description**

Print ml\_evidence

**Usage**

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

**Arguments**

x	An ml_evidence object
...	Ignored

**Value**

The object x, invisibly.

---

print.ml\_explanation *Print ml\_explanation*

---

**Description**

Print ml\_explanation

**Usage**

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

**Arguments**

x	An ml_explanation object
...	Ignored

**Value**

The object x, invisibly.

---

print.ml\_leaderboard *Print ml\_leaderboard*

---

**Description**

Print ml\_leaderboard

**Usage**

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

**Arguments**

x	An ml_leaderboard object
...	Ignored

**Value**

The object x, invisibly.

---

print.ml_metrics	<i>Print ml_metrics</i>
------------------	-------------------------

---

**Description**

Print ml\_metrics

**Usage**

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

**Arguments**

x	An ml_metrics object
...	Ignored

**Value**

The object x, invisibly.

---

print.ml_model	<i>Print an ml_model</i>
----------------	--------------------------

---

**Description**

Print an ml\_model

**Usage**

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

**Arguments**

x	An ml_model object
...	Ignored

**Value**

The object x, invisibly.

```
print.ml_profile_result  
    Print ml_profile_result
```

---

**Description**

Print ml\_profile\_result

**Usage**

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

**Arguments**

x	An ml_profile_result object
...	Ignored

**Value**

The object x, invisibly.

---

```
print.ml_shelf_result Print ml_shelf_result
```

---

**Description**

Print ml\_shelf\_result

**Usage**

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

**Arguments**

x	An ml_shelf_result object
...	Ignored

**Value**

The object x, invisibly.

---

`print.ml_split_result` *Print an ml\_split\_result*

---

### **Description**

Print an `ml_split_result`

### **Usage**

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

### **Arguments**

<code>x</code>	An <code>ml_split_result</code> object
<code>...</code>	Ignored

### **Value**

The object `x`, invisibly.

---

`print.ml_tuning_result`  
*Print an ml\_tuning\_result*

---

### **Description**

Print an `ml_tuning_result`

### **Usage**

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

### **Arguments**

<code>x</code>	An <code>ml_tuning_result</code> object
<code>...</code>	Ignored

### **Value**

The object `x`, invisibly.

---

```
print.ml_validate_result
      Print ml_validate_result
```

---

**Description**

Print ml\_validate\_result

**Usage**

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

**Arguments**

x	An ml_validate_result object
...	Ignored

**Value**

The object x, invisibly.

# Index

## \* datasets

- ml, 3
- ml, 3
- ml\_algorithms, 4
- ml\_assess, 4
- ml\_assess(), 15
- ml\_best, 5
- ml\_calibrate, 5
- ml\_check, 6
- ml\_check\_data, 7
- ml\_compare, 8
- ml\_config, 8
- ml\_cv, 9
- ml\_cv\_group, 10
- ml\_cv\_temporal, 10
- ml\_dataset, 11
- ml\_drift, 12
- ml\_drift(), 27
- ml\_embed, 13
- ml\_enough, 14
- ml\_evaluate, 15
- ml\_evaluate(), 4
- ml\_explain, 16
- ml\_fit, 16
- ml\_fit(), 26
- ml\_leak, 17
- ml\_load, 18
- ml\_optimize, 19
- ml\_plot, 20
- ml\_predict, 21
- ml\_predict\_proba, 21
- ml\_prepare, 22
- ml\_profile, 23
- ml\_quick, 23
- ml\_report, 24
- ml\_save, 25
- ml\_save(), 18
- ml\_screen, 25
- ml\_shelf, 27

- ml\_shelf(), 13, 27
- ml\_split, 28
- ml\_split(), 8
- ml\_split\_group, 29
- ml\_split\_temporal, 30
- ml\_stack, 31
- ml\_tune, 32
- ml\_validate, 33
- ml\_validate(), 26
- ml\_verify, 34
  
- predict.ml\_model, 34
- predict.ml\_tuning\_result, 35
- print.ml\_cv\_result, 36
- print.ml\_drift\_result, 36
- print.ml\_embedder, 37
- print.ml\_evidence, 37
- print.ml\_explanation, 38
- print.ml\_leaderboard, 38
- print.ml\_metrics, 39
- print.ml\_model, 39
- print.ml\_profile\_result, 40
- print.ml\_shelf\_result, 40
- print.ml\_split\_result, 41
- print.ml\_tuning\_result, 41
- print.ml\_validate\_result, 42
  
- saveRDS, 25