Package: fastml (via r-universe)

May 16, 2026

Type Package

Title Guarded Resampling Workflows for Safe and Automated Machine Learning in R

Version 0.7.8

Description Provides a guarded resampling workflow for training and evaluating machine-learning models. When the guarded resampling path is used, preprocessing and model fitting are re-estimated within each resampling split to reduce leakage risk. Supports multiple resampling schemes, integrates with established engines in the 'tidymodels' ecosystem, and aims to improve evaluation reliability by coordinating preprocessing, fitting, and evaluation within supported workflows. Offers a lightweight AutoML-style workflow by automating model training, resampling, and tuning across multiple algorithms, while keeping evaluation design explicit and user-controlled.

Encoding UTF-8

License MIT + file LICENSE

URL https://selcukorkmaz.github.io/fastml-tutorial/, https://github.com/selcukorkmaz/fastml

BugReports https://github.com/selcukorkmaz/fastml/issues

LazyData true

Depends R (>= 4.1.0)

Imports stats, recipes, dplyr, ggplot2, reshape2, rsample, parsnip, tune, workflows, yardstick, tibble, rlang, dials, RColorBrewer, doFuture, foreach, finetune, future, viridisLite, magrittr, pROC, janitor, stringr, broom, tidyr, purrr, survival, xgboost

Suggests testthat (>= 3.0.0), withr, baguette, C50, DALEX, discrim, fairmodels, flexsurv, iBreakDown, iml, lime, modelStudio, pdp, plsmod, probably, ranger, rstpm2, survRM2, aorsf, censored, crayon, kernlab, klaR, kknn, keras, lightgbm, rstanarm, patchwork, GGally, glmnet, themis, DT, UpSetR, VIM, dbscan, gridExtra, htmlwidgets, kableExtra, moments, naniar, plotly, scales, skimr, sparsediscrim, knitr, rmarkdown, pec

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/pak/sysreqs cmake make libicu-dev libuv1-dev

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

Date/Publication 2026-03-17 23:08:40 UTC

RemoteUrl https://github.com/selcukorkmaz/fastml

RemoteRef HEAD

RemoteSha 13564c7a25082105f0f42e5cf0bcff7326c4729a

Contents

`availableMethods` *Get Available Methods*

Description

Returns a character vector of algorithm names available for classification, regression or survival tasks.

Usage

```
availableMethods(type = c("classification", "regression", "survival"), ...)
```

Arguments

`type`	A character string specifying the type of task. Must be one of `"classification"`, `"regression"`, or `"survival"`. Defaults to `c("classification", "regression", "survival")` and uses `match.arg` to select one.
`...`	Additional arguments (currently not used).

Details

Depending on the specified `type`, the function returns a different set of algorithm names:

- For `"classification"`, it returns algorithms such as `"logistic_reg"`, `"multinom_reg"`, `"decision_tree"`, `"C5_rules"`, `"rand_forest"`, `"xgboost"`, `"lightgbm"`, `"svm_linear"`, `"svm_rbf"`, `"nearest_neighbor"`, `"naive_Bayes"`, `"mlp"`, `"discrim_linear"`, `"discrim_quad"`, and `"bag_tree"`.
- For `"regression"`, it returns algorithms such as `"linear_reg"`, `"ridge_reg"`, `"lasso_reg"`, `"elastic_net"`, `"decision_tree"`, `"rand_forest"`, `"xgboost"`, `"lightgbm"`, `"svm_linear"`, `"svm_rbf"`, `"nearest_neighbor"`, `"mlp"`, `"pls"`, and `"bayes_glm"`.
- For `"survival"`, it returns algorithms such as `"rand_forest"`, `"cox_ph"`, `"penalized_cox"`, `"stratified_cox"`, `"time_varying_cox"`, `"survreg"`, `"royston_parmar"`, `"parametric_surv"`, `"piecewise_exp"`, and `"xgboost"`.

Value

A character vector containing the names of the available algorithms for the specified task type.

`counterfactual_explain` *Generate counterfactual explanations for a fastml model*

Description

Uses DALEX ceteris-paribus profiles (‘predict_profile‘) to compute counterfactual-style what-if explanations for a given observation.

Usage

```
counterfactual_explain(
  object,
  observation,
  variables = NULL,
  data = c("train", "test"),
  positive_class = NULL,
  event_class = NULL,
  label_levels = NULL,
  ...
)
```

Arguments

`object`	A ‘fastml‘ object.
`observation`	A single observation (data frame with one row) to compute counterfactuals for.
`variables`	Optional character vector of candidate variables to vary. Only numeric variables are used for counterfactual profiling.
`data`	Character string specifying which data to use for the explainer background: `"train"` (default) or `"test"`.
`positive_class`	Optional string used to filter lines/points in the resulting profiles for classification tasks.
`event_class`	Optional event class indicator propagated from ‘fastml_prepare_explainer_inputs()‘ (kept for compatibility).
`label_levels`	Optional vector of label levels propagated from ‘fastml_prepare_explainer_inputs()‘ (kept for compatibility).
`...`	Additional arguments passed to ‘DALEX::predict_profile‘.

Value

A list (returned invisibly) containing the DALEX profile, filtered lines/points when ‘positive_class‘ is supplied, and the plotted object if rendering succeeds.

defaults_registry *Defaults Registry for Engine and Parameter Transparency*

Description

Functions to track, compare, and warn about differences between fastml defaults and parsnip defaults, providing users with full transparency and control over model configuration.

estimate_tuning_time *Estimate Tuning Time*

Description

Provides a rough estimate of tuning time based on the configuration.

Usage

```
estimate_tuning_time(
  n_params,
  n_folds = 10,
  n_rows = 1000,
  complexity = "balanced",
  tuning_strategy = "grid",
  base_fit_time = 1
)
```

Arguments

n_params	Number of parameters being tuned.
n_folds	Number of cross-validation folds.
n_rows	Number of rows in training data.
complexity	Tuning complexity level.
tuning_strategy	Tuning strategy ("grid" or "bayes").
base_fit_time	Estimated time for a single model fit in seconds.

Value

A list with estimated total time and breakdown.

explain_ale *Compute Accumulated Local Effects (ALE) for a fastml model*

Description

Uses the ‘iml‘ package to calculate ALE for the specified feature.

Usage

```
explain_ale(object, feature, data = c("train", "test"), ...)
```

Arguments

object	A ‘fastml‘ object.
feature	Character string specifying the feature name.
data	Character string specifying which data to use: "train" (default) or "test".
...	Additional arguments passed to ‘iml::FeatureEffect‘.

Value

An ‘iml‘ object containing ALE results.

Examples

```
## Not run:
data(iris)
iris <- iris[iris$Species != "setosa", ]
iris$Species <- factor(iris$Species)
model <- fastml(data = iris, label = "Species")
explain_ale(model, feature = "Sepal.Length")

## End(Not run)
```

explain_dalex *Generate DALEX explanations for a fastml model*

Description

Creates a DALEX explainer and computes permutation based variable importance, partial dependence (model profiles) and Shapley values.

Usage

```
explain_dalex(
  object,
  data = c("train", "test"),
  features = NULL,
  grid_size = 20,
  shap_sample = 5,
  vi_iterations = 10,
  seed = 123,
  loss_function = NULL
)
```

Arguments

object	A fastml object.
data	Character string specifying which data to use for explanations: "train" (default) uses training data, "test" uses held-out test data. Using test data provides explanations that better reflect model generalization, while training data explanations may be influenced by overfitting.
features	Character vector of feature names for partial dependence (model profiles). Default NULL.
grid_size	Number of grid points for partial dependence. Default 20.
shap_sample	Integer number of observations from the selected data source to compute SHAP values for. Default 5.
vi_iterations	Integer. Number of permutations for variable importance (B). Default 10.
seed	Integer. A value specifying the random seed.
loss_function	Function. The loss function for model_parts. • If NULL and task = 'classification', defaults to DALEX::loss_cross_entropy. • If NULL and task = 'regression', defaults to DALEX::loss_root_mean_square.

Value

Invisibly returns a list with variable importance, optional model profiles and SHAP values.

---

explain_lime *Generate LIME explanations for a fastml model*

Description

Creates a 'lime' explainer using processed (encoded, scaled) data and returns feature explanations for new observations. The new observation is automatically preprocessed using the same recipe to ensure alignment with the explainer background.

Usage

```
explain_lime(
  object,
  new_observation,
  data = c("train", "test"),
  n_features = 5,
  n_labels = 1,
  ...
)
```

Arguments

object	A 'fastml' object.
new_observation	A data frame containing the new observation(s) to explain. Must contain the same columns as the original training data (before preprocessing). The function will apply the stored preprocessor to transform it.
data	Character string specifying which data to use for the LIME explainer background: "train" (default) or "test".
n_features	Number of features to show in the explanation. Default 5.
n_labels	Number of labels to explain (classification only). Default 1.
...	Additional arguments passed to 'lime::explain'.

Value

An object produced by 'lime::explain'.

Examples

```
## Not run:
data(iris)
iris <- iris[iris$Species != "setosa", ]
iris$Species <- factor(iris$Species)
model <- fastml(data = iris, label = "Species")
explain_lime(model, new_observation = iris[1, ])

## End(Not run)
```

---

explain_stability *Analyze Feature Importance Stability Across Cross-Validation Folds*

Description

Computes feature importance for each fold model and aggregates results to assess the stability of feature importance rankings across resamples. This helps identify features that are consistently important vs those whose importance varies across different data subsets.

Usage

```
explain_stability(
  object,
  model_name = NULL,
  vi_iterations = 10,
  seed = 123,
  plot = FALSE,
  conf_level = 0.95
)
```

Arguments

`object`	A `fastml` object trained with `store_fold_models = TRUE`.
`model_name`	Character string specifying which model to analyze. If NULL, uses the best model. Should match the format "algorithm (engine)", e.g., "rand_forest (ranger)".
`vi_iterations`	Integer. Number of permutations for variable importance per fold. Default is 10 for faster computation across many folds.
`seed`	Integer. Random seed for reproducibility.
`plot`	Logical. If TRUE, displays a stability plot showing mean importance with confidence intervals. Default is FALSE.
`conf_level`	Numeric. Confidence level for intervals. Default is 0.95.

Details

This function requires that the fastml model was trained with `store_fold_models = TRUE`, which stores the models fitted on each cross-validation fold. Without stored fold models, only the final best model is available, and cross-fold stability analysis is not possible.

The stability analysis computes permutation-based variable importance for each fold's model using DALEX, then aggregates across folds to show:

- Mean importance and standard deviation
- Confidence intervals for importance
- Rank stability (how consistently features rank across folds)

Features with high mean importance but also high variance may be important for some data subsets but not others, suggesting potential instability in the model's reliance on those features.

Value

A list with class `"fastml_stability"` containing:

importance_summary Data frame with aggregated feature importance (mean, sd, se, lower/upper CI) across folds.

fold_importance List of per-fold variable importance results.

rank_stability Data frame showing how feature ranks vary across folds.

n_folds Number of folds analyzed.

model_name Name of the model analyzed.

Examples

```
# Train model with fold models stored
model <- fastml(
  data = iris,
  label = "Species",
  algorithms = "rand_forest",
  store_fold_models = TRUE
)

# Analyze stability
stability <- explain_stability(model)
print(stability)
plot(stability)
```

`fastexplain` *Explain a fastml model using various techniques*

Description

Provides model explainability across several backends. With `method = "dalex"` it:

- Creates a DALEX explainer from the trained model.
- Computes permutation-based variable importance with `vi_iterations` permutations and displays the table and plot.
- Computes partial dependence-like model profiles when `features` are supplied.
- Computes Shapley values (SHAP) for `shap_sample` training rows, displays the SHAP table, and plots a canonical SHAP summary (beeswarm) plot colored by raw feature values and ordered by mean(|SHAP value|) per feature. For classification, separate panels per class are shown.

Usage

```
fastexplain(
  object,
  method = "dalex",
  data = c("train", "test"),
  features = NULL,
  n_features = 5,
  variables = NULL,
  observation = NULL,
  grid_size = 20,
  shap_sample = 5,
  vi_iterations = 10,
  seed = 123,
  loss_function = NULL,
```

```
  protected = NULL,
  ...
)
```

Arguments

object	A fastml object.
method	Character string specifying the explanation method. Supported values are "dalex", "lime", "ice", "ale", "surrogate", "interaction", "studio", "fairness", "breakdown", and "counterfactual". Defaults to "dalex".
data	Character string specifying which data to use for explanations: "train" (default) uses training data, "test" uses held-out test data. Using test data provides explanations that better reflect model generalization, while training data explanations may be influenced by overfitting.
features	Character vector of feature names for partial dependence (model profiles). Default NULL.
n_features	Number of features to show in the explanation (used for lime). Default 5.
variables	Character vector. Variable names to compute explanations for (used for counterfactuals).
observation	A single observation for methods that need a new data point (method = "lime", method = "counterfactual", or method = "breakdown"). Default NULL.
grid_size	Number of grid points for partial dependence. Default 20.
shap_sample	Integer number of observations from the selected data source to compute SHAP values for. Default 5.
vi_iterations	Integer. Number of permutations for variable importance (B). Default 10.
seed	Integer. A value specifying the random seed.
loss_function	Function. The loss function for model_parts. • If NULL and task = 'classification', defaults to DALEX::loss_cross_entropy. • If NULL and task = 'regression', defaults to DALEX::loss_root_mean_square.
protected	Character or factor vector of protected attribute(s) required for method = "fairness". Default NULL.
...	Additional arguments passed to the underlying helper functions for the chosen method.

Details

- **Data source selection:** By default, explanations are computed on training data (data = "train"), which reflects in-sample model behavior and may be influenced by overfitting. Set data = "test" to compute explanations on held-out test data for a more realistic assessment of how the model uses features on unseen data.
- **Method dispatch:** method can route to LIME, ICE, ALE, surrogate tree, interaction strengths, DALEX/modelStudio dashboards, fairness diagnostics, iBreakDown contributions, or counterfactual search.

- **Variable importance controls:** Use `vi_iterations` to tune permutation stability and `loss_function` to override the default DALEX loss (cross-entropy for classification, RMSE for regression).
- **Fairness and breakdown support:** Provide `protected` for `method = "fairness"` and an `observation` for `method = "breakdown"` or `method = "counterfactual"`. Observations are aligned to the explainer data before scoring.

Value

For DALEX-based methods, prints variable importance, model profiles, and SHAP summaries. Other methods return their respective explainer objects (e.g., LIME explanations, ALE plot, surrogate tree, interaction strengths, modelStudio dashboard, fairmodels object, breakdown object, or counterfactual results), usually invisibly after plotting or printing.

Note

By default, explanations use training data. For unbiased feature importance estimates that better reflect model generalization, use `data = "test"` to compute explanations on held-out test data.

`fastexplore` *Lightweight exploratory helper*

Description

'fastexplore()' is an optional, lightweight exploratory data analysis (EDA) helper. It returns summary tables and plot objects; it only writes to disk or renders a report when you explicitly request it via 'save_results' or 'render_report'.

Usage

```
fastexplore(
  data,
  label = NULL,
  visualize = c("histogram", "boxplot", "barplot", "heatmap", "scatterplot"),
  save_results = FALSE,
  render_report = FALSE,
  output_dir = NULL,
  sample_size = NULL,
  interactive = FALSE,
  corr_threshold = 0.9,
  auto_convert_numeric = TRUE,
  visualize_missing = TRUE,
  imputation_suggestions = FALSE,
  report_duplicate_details = TRUE,
  detect_near_duplicates = FALSE,
  auto_convert_dates = FALSE,
  feature_engineering = FALSE,
  outlier_method = c("iqr", "zscore", "dbscan", "lof"),
```

```
  run_distribution_checks = TRUE,
  normality_tests = c("shapiro"),
  pairwise_matrix = TRUE,
  max_scatter_cols = 5,
  grouped_plots = TRUE,
  use_upset_missing = TRUE
)
```

Arguments

data	A 'data.frame' to explore.
label	Optional column name of the target/label. If supplied and categorical, grouped plots and class balance summaries are produced.
visualize	Character vector indicating which plot families to build. Defaults to 'c("histogram", "boxplot", "barplot", "heatmap", "scatterplot")'.
save_results	Logical; if 'TRUE', plots/results are saved under 'output_dir' (defaults to the working directory). Default is 'FALSE'.
render_report	Logical; if 'TRUE', a short HTML report is rendered via 'rmarkdown' (if available). Default is 'FALSE'.
output_dir	Directory to save results/report when 'save_results' or 'render_report' is 'TRUE'.
sample_size	Optional integer; if supplied, visualizations are produced on a random sample of this size.
interactive	Logical; if 'TRUE' and 'plotly' is available, an interactive correlation heatmap is produced. Falls back to static ggplot output otherwise.
corr_threshold	Absolute correlation threshold for flagging high correlations.
auto_convert_numeric	Logical; convert factor/character columns that look numeric into numeric.
visualize_missing	Logical; if 'TRUE', include simple missingness visualizations.
imputation_suggestions	Logical; if 'TRUE', prints lightweight suggestions based on missingness patterns.
report_duplicate_details	Logical; if 'TRUE', returns a small sample of duplicated rows when present.
detect_near_duplicates	Placeholder for future fuzzy duplicate checks.
auto_convert_dates	Logical; convert YYYY-MM-DD strings to 'Date'.
feature_engineering	Logical; if 'TRUE', derive day/month/year from date columns to aid inspection of temporal structure.
outlier_method	One of '"iqr"', '"zscore"', '"dbscan"', '"lof"'.
run_distribution_checks	Logical; if 'TRUE', run normality tests on numeric columns.

`normality_tests`
: Character vector of normality tests to run; currently supports ‘"shapiro"‘ and ‘"ks"‘.

`pairwise_matrix`
: Logical; if ‘TRUE‘ and ‘GGally‘ is available, returns a ggpairs scatterplot matrix for a subset of numeric columns.

`max_scatter_cols`
: Maximum number of numeric columns to include in the pairwise matrix.

`grouped_plots`
: Logical; if ‘TRUE‘ and ‘label‘ is a factor, group histograms/boxplots/density plots by label.

`use_upset_missing`
: Logical; retained for compatibility. When ‘TRUE‘ and ‘UpSetR‘ is installed, an UpSet plot of missingness is returned; otherwise a simpler missingness heatmap is used.

Details

This helper is intentionally decoupled from the core modeling workflow. Most of its heavy dependencies are treated as optional and loaded via ‘requireNamespace()‘ when requested features are used.

Value

A list of summaries (tables/tibbles) and plot objects (ggplot/plotly), plus any saved file paths when ‘save_results‘/‘render_report‘ are enabled.

`fastml` *Fast Machine Learning Function*

Description

Trains and evaluates multiple classification or regression models automatically detecting the task based on the target variable type.

Usage

```
fastml(
  data = NULL,
  train_data = NULL,
  test_data = NULL,
  label,
  algorithms = "all",
  task = "auto",
  test_size = 0.2,
  resampling_method = if (identical(task, "survival")) "none" else "cv",
  folds = ifelse(grepl("cv", resampling_method), 10, 25),
  repeats = NULL,
```

```
  group_cols = NULL,
  block_col = NULL,
  block_size = NULL,
  initial_window = NULL,
  assess_window = NULL,
  skip = 0,
  outer_folds = NULL,
  event_class = "first",
  exclude = NULL,
  recipe = NULL,
  tune_params = NULL,
  engine_params = list(),
  metric = NULL,
  class_threshold = 0.5,
  algorithm_engines = NULL,
  use_parsnip_defaults = FALSE,
  warn_engine_defaults = TRUE,
  n_cores = 1,
  stratify = TRUE,
  impute_method = "error",
  encode_categoricals = TRUE,
  scaling_methods = c("center", "scale"),
  balance_method = "none",
  resamples = NULL,
  summaryFunction = NULL,
  use_default_tuning = FALSE,
  tuning_strategy = "grid",
  tuning_iterations = 10,
  tuning_complexity = "balanced",
  grid_levels = NULL,
  early_stopping = FALSE,
  adaptive = FALSE,
  learning_curve = FALSE,
  seed = 123,
  verbose = FALSE,
  eval_times = NULL,
  survival_metric_convention = "fastml",
  bootstrap_ci = TRUE,
  bootstrap_samples = 500,
  bootstrap_seed = NULL,
  at_risk_threshold = 0.1,
  audit_mode = FALSE,
  multiclass_auc = "macro",
  store_fold_models = FALSE
)
```

Arguments

data	A data frame containing the complete dataset. If both ‘train_data‘ and ‘test_data‘ are ‘NULL‘, ‘fastml()‘ will split this into training and testing sets according to ‘test_size‘ and ‘stratify‘. When ‘group_cols‘ is supplied, the holdout keeps groups intact; when ‘block_col‘ is supplied, the holdout uses the last rows in time order. Defaults to ‘NULL‘.
train_data	A data frame pre-split for model training. If provided, ‘test_data‘ must also be supplied, and no internal splitting will occur. Defaults to ‘NULL‘.
test_data	A data frame pre-split for model evaluation. If provided, ‘train_data‘ must also be supplied, and no internal splitting will occur. Defaults to ‘NULL‘.
label	A string specifying the name of the target variable. For survival analysis, supply a character vector with the names of the time and status columns.
algorithms	A vector of algorithm names to use. Default is `"all"` to run all supported algorithms.
task	Character string specifying model type selection. Use "auto" to let the function detect whether the target is for classification, regression, or survival based on the data. Survival is detected when ‘label‘ is a character vector of length 2 that matches time and status columns in the data. You may also explicitly set to "classification", "regression", or "survival".
test_size	A numeric value between 0 and 1 indicating the proportion of the data to use for testing. For grouped holdout, this is applied to groups; for time-ordered holdout, it selects the final proportion of rows. Default is `0.2`.
resampling_method	A string specifying the resampling method for model evaluation. Default is `"cv"` (cross-validation) for classification/regression. Other options include `"none"`, `"boot"`, `"repeatedcv"`, `"grouped_cv"`, `"blocked_cv"`, `"rolling_origin"`, and `"nested_cv"`. For survival tasks, resampling is supported for parsnip-compatible engines (e.g., censored/ranger, glmnet). Native survival engines (flexsurv/rstpm2/custom xgboost) ignore resampling and will error if custom resamples are supplied. When the task auto-detects survival and `resampling_method` is omitted, it defaults to `"none"` so native engines continue to run; set it explicitly to enable resampling for parsnip survival fits.
folds	An integer specifying the number of folds for cross-validation (default `10` for methods containing "cv", `25` otherwise). When `resampling_method = "boot"`, this controls the number of bootstrap resamples. When `resampling_method = "validation_split"`, the proportion held out for validation is derived as `1 - 1/folds`.
repeats	Number of times to repeat cross-validation (only applicable for methods like "repeatedcv").
group_cols	Character vector naming one or more grouping columns used when `resampling_method = "grouped_cv"` or when grouped nested cross-validation is desired. All rows that share the same combination of values remain together in every fold. Columns must exist in the training data and cannot contain missing values.
block_col	Single column name that defines the ordering variable for `resampling_method = "blocked_cv"` or `"rolling_origin"`. Data must already be sorted in ascending order by this column to avoid leakage from future observations.

`block_size` Positive integer specifying the block size for `"blocked_cv"`.

`initial_window` Positive integer giving the number of observations in the initial training window for `"rolling_origin"` resampling.

`assess_window` Positive integer giving the number of observations in each assessment window for `"rolling_origin"` resampling.

`skip` Non-negative integer specifying how many potential rolling windows to skip between successive resamples when `resampling_method = "rolling_origin"`.

`outer_folds` Positive integer giving the number of outer folds to use when `resampling_method = "nested_cv"` and no custom `resamples` object is supplied.

`event_class` A single string. Either "first" or "second" to specify which level of the binary outcome factor to treat as the positive class (the "event"). For binary classification, "first" treats the first factor level as the positive class, "second" treats the second level as positive. Use `levels(your_data$outcome)` to check level order before training. Default is "first".

`exclude` A character vector specifying the names of the columns to be excluded from the training process.

`recipe` A user-defined `recipe` object for custom preprocessing. If provided, internal recipe steps (imputation, encoding, scaling) are skipped.

`tune_params` A named list of tuning ranges for each algorithm and engine pair. Example: `list(rand_forest = list(ranger = list(mtry = c(1, 3))))` will override the defaults for the ranger engine. Default is `NULL`.

`engine_params` A named list of engine-level arguments to pass directly to the underlying model fitting functions. Use this for fixed settings that should apply whenever an engine is fitted (for example, `list(royston_parmar = list(rstpm2 = list(link = "PO")))`, `list(cox_ph = list(survival = list(ties = "breslow")))`, or `list(rand_forest = list(ranger = list(importance = "impurity")))`). These arguments are distinct from `tune_params`, which define ranges of hyperparameters to explore during tuning. Default is an empty list.

`metric` The performance metric to optimize during training. For classification, options include `"accuracy"`, `"roc_auc"`, `"logloss"`, `"brier_score"`, and `"ece"` (plus other class metrics).

`class_threshold`
For binary classification, controls how class probabilities are converted into hard class predictions during holdout evaluation. The default is `0.5` (standard threshold). Numeric values in (0, 1) set a fixed threshold. Use `"auto"` to tune a threshold on training data to maximize F1, or specify a metric name (e.g., `"sens"`, `"spec"`, `"youden"`) to optimize for that metric. Use `"model"` to keep the model's default predictions.

`algorithm_engines`
A named list specifying the engine to use for each algorithm.

`use_parsnip_defaults`
Logical. If `TRUE`, fastml uses parsnip's default engines instead of fastml's optimized defaults. This provides compatibility with standard tidymodels behavior. If `FALSE` (default), fastml uses its own curated engine defaults which may differ from parsnip. When engines differ, a warning is issued unless `algorithm_engines`

explicitly specifies the engine. Use `print_default_differences()` to see all differences.

`warn_engine_defaults`
: Logical. If `TRUE` (default), warns when fastml's default engine differs from parsnip's default. Set to `FALSE` to suppress these warnings. Warnings are only shown once per algorithm per session.

`n_cores`
: An integer specifying the number of CPU cores to use for parallel processing. Default is `1`.

`stratify`
: Logical indicating whether to use stratified sampling when splitting the data. Only applied to random holdout splitting. Default is `TRUE` for classification and `FALSE` for regression.

`impute_method`
: Method for handling missing values. Options include:

  `"medianImpute"` Impute missing values using median imputation (recipe-based).

  `"knnImpute"` Impute missing values using k-nearest neighbors (recipe-based).

  `"bagImpute"` Impute missing values using bagging (recipe-based).

  `"remove"` Remove rows with missing values from the data (recipe-based).

  `"error"` Do not perform imputation; if missing values are detected, stop execution with an error.

  `NULL` Equivalent to `"error"`. No imputation is performed, and the function will stop if missing values are present.

  All imputation occurs inside the recipe so the same trained preprocessing can be applied at prediction time. Default is `"error"`.

`encode_categoricals`
: Logical indicating whether to encode categorical variables. Default is `TRUE`.

`scaling_methods`
: Vector of scaling methods to apply. Default is `c("center", "scale")`.

`balance_method`
: Method to handle class imbalance. One of `"none"`, `"upsample"`, or `"downsample"`. Applied inside the preprocessing recipe so each resampling split is balanced independently (requires the `themis` package when enabled). Default is `"none"`.

`resamples`
: Optional rsample object providing custom resampling splits. If supplied, `resampling_method`, `folds`, and `repeats` are ignored.

`summaryFunction`
: A custom summary function for model evaluation. Default is `NULL`.

`use_default_tuning`
: Logical. Tuning only runs when resamples are supplied and `tuning_strategy` is not `"none"`. If `TRUE` and `tune_params` is `NULL`, default grids are used; if `tune_params` is provided, those values override/extend defaults. When `FALSE` and no custom parameters are given, models are fitted once with default settings. If no resamples are available or `tuning_strategy = "none"`, tuning requests are ignored with a warning. Default is `FALSE`.

`tuning_strategy`
: A string specifying the tuning strategy. Must be one of `"grid"`, `"bayes"`, or `"none"`. Default is `"grid"`. If custom `tune_params` are provided while `tuning_strategy = "none"`, they will be ignored with a warning.

`tuning_iterations`
: Number of iterations for Bayesian tuning. Ignored when `tuning_strategy` is not `"bayes"`. Validation of this argument only occurs for the Bayesian strategy. Default is `10`.

`tuning_complexity`
: Character string specifying a tuning complexity preset that controls grid density and parameter range width. One of:

  `"quick"` Minimal tuning (2 levels/param, ~32 combinations for 5 params). Best for: prototyping, debugging, time-constrained scenarios.

  `"balanced"` Standard tuning (3 levels/param, ~243 combinations). Best for: most production use cases. This is the default.

  `"thorough"` Comprehensive tuning (5 levels/param, ~3,125 combinations). Best for: final model selection, publications.

  `"exhaustive"` Maximum coverage (7 levels/param, ~16,807 combinations). Best for: research, competitions. Consider Bayesian tuning instead.

  See `print_tuning_presets` for detailed comparison. Ignored if `grid_levels` is explicitly set.

`grid_levels`
: Integer specifying the number of levels per parameter for grid search. Higher values create denser grids but increase computation time exponentially (grid size = levels^n_params). Typical values:

  - 2: Very fast, minimal coverage
  - 3: Balanced (default via `tuning_complexity = "balanced"`)
  - 5: Thorough coverage
  - 7+: Exhaustive (consider Bayesian tuning instead)

  If `NULL` (default), determined by `tuning_complexity`.

`early_stopping`
: Logical indicating whether to use early stopping in Bayesian tuning methods (if supported). Default is `FALSE`.

`adaptive`
: Logical indicating whether to use adaptive/racing methods for tuning. Default is `FALSE`.

`learning_curve`
: Logical. If TRUE, generate learning curves (performance vs. training size).

`seed`
: An integer value specifying the random seed for reproducibility. fastml also configures parallel backends for deterministic RNG streams when possible; some external engines (e.g., h2o, spark, keras) may still be nondeterministic and will emit a warning.

`verbose`
: Logical; if TRUE, prints progress messages during the training and evaluation process.

`eval_times`
: Optional numeric vector of evaluation horizons for survival models. When `NULL`, defaults to the median and 75th percentile of the observed follow-up times (rounded to the dataset's time unit).

`survival_metric_convention`
: Character string specifying which survival metric conventions to follow. ‘"fastml"‘ (default) uses fastml's internal defaults for evaluation horizons and t_max. ‘"tidymodels"‘ uses ‘eval_times‘ as the explicit evaluation grid and applies yardstick-style Brier/IBS normalization; when ‘eval_times‘ is ‘NULL‘, time-dependent Brier metrics are omitted.

`bootstrap_ci` Logical indicating whether bootstrap confidence intervals should be computed for performance metrics. Applies to all task types.

`bootstrap_samples`
Integer giving the number of bootstrap resamples to use when `bootstrap_ci = TRUE`. Defaults to 500.

`bootstrap_seed` Optional seed passed to the bootstrap procedure used to estimate confidence intervals. When omitted, defaults to 'seed' for reproducible intervals; set to 'NULL' to allow random bootstrap draws.

`at_risk_threshold`
Numeric value between 0 and 1 used for survival metrics to determine the last follow-up time (t_{max}). The maximum time is set to the largest observed time where at least this proportion of subjects remain at risk.

`audit_mode` Logical; if `TRUE`, enables runtime auditing of custom preprocessing hooks and records potentially unsafe behaviour (such as global environment access or file I/O) while flagging the run as potentially unsafe.

`multiclass_auc` For multiclass ROC AUC, the averaging method to use: '"macro"' (default, tidymodels) or '"macro_weighted"'. Macro weights each class equally, while macro_weighted weights by class prevalence and can change model rankings on imbalanced data.

`store_fold_models`
Logical. If `TRUE`, stores the models trained on each cross-validation fold (memory intensive). This enables `explain_stability` to compute feature importance across folds and assess explanation stability. Default is `FALSE`.

Details

Fast Machine Learning Function

Trains and evaluates multiple classification or regression models. The function automatically detects the task based on the target variable type and can perform advanced hyperparameter tuning using various tuning strategies.

Model selection is based exclusively on resampling metrics (cross-validation or nested CV). The holdout split is reserved for final performance estimation and is never used to choose the best model, mirroring `tidymodels::last_fit()` semantics.

For multiclass ROC AUC, fastml defaults to macro averaging (tidymodels). Macro treats each class equally, while macro_weighted weights by class prevalence and can change model rankings on imbalanced data. Keep the same setting when comparing runs.

Tuning: Speed vs Robustness Trade-offs

Hyperparameter tuning involves a fundamental trade-off between computational cost and the likelihood of finding optimal hyperparameters. fastml provides presets via `tuning_complexity` to make this trade-off explicit:

Level	Grid Size*	Time	Quality	Use Case
quick	~32	~1x	Low	Prototyping, debugging
balanced	~243	~10x	Medium	Most production use
thorough	~3,125	~100x	High	Final models, papers

exhaustive	~16,807	~1000x	Very High	Research, competitions

*Grid size shown for 5 tunable parameters (levels^5)

Recommendations:

- Start with `tuning_complexity = "quick"` during development
- Use `"balanced"` (default) for most production pipelines
- Switch to `"thorough"` for final model selection
- Consider `tuning_strategy = "bayes"` instead of exhaustive grid search
- Enable `adaptive = TRUE` for early stopping of poor configurations

Use `print_tuning_presets` to see all presets and `estimate_tuning_time` to estimate runtime before starting.

Value

An object of class `fastml` containing the best model, performance metrics, and other information.

Factor Level Warning

For binary classification, the interpretation of metrics like sensitivity, specificity, and ROC AUC depends on which factor level is treated as the "positive" class (the event of interest). The `event_class` parameter controls this:

- `"first"` (default): The first factor level is treated as positive
- `"second"`: The second factor level is treated as positive

Important: Recipe preprocessing steps like `step_other()` or `step_unknown()` can modify factor levels, potentially changing which level is "first" or "second". Always verify factor levels after preprocessing.

To ensure consistent behavior, explicitly set factor levels before calling fastml:

```
# Ensure "positive" is the second level (event_class = "second")
data$outcome <- factor(data$outcome, levels = c("negative", "positive"))

# Or ensure "positive" is the first level (event_class = "first")
data$outcome <- factor(data$outcome, levels = c("positive", "negative"))
```

Examples

```
# Example 1: Using the iris dataset for binary classification (excluding 'setosa')
data(iris)
iris <- iris[iris$Species != "setosa", ]  # Binary classification
iris$Species <- factor(iris$Species)

# Define a custom tuning grid for the ranger engine
tune <- list(
  rand_forest = list(
```

```
    ranger = list(mtry = c(1, 3))
  )
)

# Train models with custom tuning
model <- fastml(
  data = iris,
  label = "Species",
  algorithms = "rand_forest",
  tune_params = tune,
  use_default_tuning = TRUE
)

# View model summary
summary(model)
```

`fastml_compute_holdout_results` *Evaluate Models Function*

Description

Evaluates the trained models on the test data and computes performance metrics.

Usage

```
fastml_compute_holdout_results(
  models,
  train_data,
  test_data,
  label,
  start_col = NULL,
  time_col = NULL,
  status_col = NULL,
  task,
  metric = NULL,
  event_class,
  class_threshold = 0.5,
  eval_times = NULL,
  bootstrap_ci = TRUE,
  bootstrap_samples = 500,
  bootstrap_seed = 1234,
  at_risk_threshold = 0.1,
  survival_metric_convention = "fastml",
  precomputed_predictions = NULL,
```

```
  summaryFunction = NULL,
  multiclass_auc = "macro"
)
```

Arguments

models	A list of trained model objects.
train_data	Preprocessed training data frame.
test_data	Preprocessed test data frame.
label	Name of the target variable. For survival analysis this should be a character vector of length two giving the names of the time and status columns.
start_col	Optional string. The name of the column specifying the start time in counting process (e.g., '(start, stop, event)') survival data. Only used when task = "survival".
time_col	String. The name of the column specifying the event or censoring time (the "stop" time in counting process data). Only used when task = "survival".
status_col	String. The name of the column specifying the event status (e.g., 0 for censored, 1 for event). Only used when task = "survival".
task	Type of task: "classification", "regression", or "survival".
metric	The performance metric to optimize (e.g., "accuracy", "rmse").
event_class	A single string. Either "first" or "second" to specify which level of truth to consider as the "event".
class_threshold	For binary classification, controls how class probabilities are converted into hard class predictions. The default is '0.5' (standard threshold). Numeric values in (0, 1) set a fixed threshold. Use '"auto"' to tune a threshold on training data to maximize F1; use '"model"' to keep the model's default predictions.
eval_times	Optional numeric vector of evaluation horizons for survival metrics. Passed through to process_model.
bootstrap_ci	Logical indicating whether bootstrap confidence intervals should be computed for the evaluation metrics.
bootstrap_samples	Number of bootstrap resamples used when bootstrap_ci = TRUE.
bootstrap_seed	Optional integer seed for the bootstrap procedure used in metric estimation.
at_risk_threshold	Minimum proportion of subjects that must remain at risk to define t_{max} when computing survival metrics such as the integrated Brier score.
survival_metric_convention	Character string specifying which survival metric conventions to follow. '"fastml"' (default) uses fastml's internal defaults for evaluation horizons and t_max. '"tidymodels"' uses 'eval_times' as the explicit evaluation grid and applies yardstick-style Brier/IBS normalization; when 'eval_times' is 'NULL', time-dependent Brier metrics are omitted.

`precomputed_predictions`
: Optional data frame or nested list of previously generated predictions (per algorithm/engine) to reuse instead of recomputing. This is mainly used when combining results across engines.

`summaryFunction`
: Optional custom classification metric function passed through to `process_model` for holdout evaluation.

`multiclass_auc`
: For multiclass ROC AUC, the averaging method to use: ‘"macro"‘ (default, tidymodels) or ‘"macro_weighted"‘. Macro weights each class equally, while macro_weighted weights by class prevalence and can change model rankings on imbalanced data.

Value

A list with two elements:

performance A named list of performance metric tibbles for each model.

predictions A named list of data frames with columns including truth, predictions, and probabilities per model.

`fastml_guard_validate_indices` *Guarded Resampling Utilities*

Description

Internal helpers that enforce the Guarded Resampling Principle by fitting preprocessing pipelines independently within each resampling split. These functions are not exported.

Usage

```
fastml_guard_validate_indices(indices, label)
```

Arguments

`indices`	Numeric vector of row indices for a resample split.
`label`	Character string used to identify the index source in errors.

fastml_normalize_survival_status

Internal helpers for survival-specific preprocessing

Description

These utilities standardize survival status indicators so that downstream metrics always receive the conventional coding (0 = censored, 1 = event). The functions are intentionally unexported and are used across multiple internal modules. Normalize survival status coding to 0/1 representation

Usage

```
fastml_normalize_survival_status(status_vec, reference_length = NULL)
```

Arguments

`status_vec`	A vector containing survival status information. May be numeric, logical, factor, or character.
`reference_length`	Optional integer specifying the desired length of the returned vector. When ‘status_vec‘ is ‘NULL‘, this value controls the length of the output (defaulting to 0 when not supplied).

Details

This helper attempts to coerce a status vector into a numeric format where 0 represents censoring and 1 represents the event indicator. It accepts a variety of common encodings such as 1/2, logical values, factors, or character labels. When the supplied values deviate from the canonical coding, the function records that a recode was performed so callers can communicate this to the user (once).

Value

A list with two elements: ‘status‘, the recoded numeric vector, and ‘recoded‘, a logical flag indicating whether a non-standard encoding was detected.

flatten_and_rename_models

Flatten and Rename Models

Description

Flatten and Rename Models

Usage

```
flatten_and_rename_models(models)
```

Arguments

models	A named list of model objects or nested named lists of model objects. Nested entries are flattened and renamed using the pattern ‘algorithm (engine)‘.

Value

A flat named list of model objects.

---

get_best_model_idx *Get Best Model Indices by Metric and Group*

Description

Get Best Model Indices by Metric and Group

Usage

```
get_best_model_idx(df, metric, group_cols = c("Model", "Engine"))
```

Arguments

df	A data frame containing model performance summaries.
metric	A single character string naming the column in ‘df‘ used to identify the best model.
group_cols	Character vector of column names used to define model groups before selecting the best metric value. Defaults to ‘c("Model", "Engine")‘.

Value

Integer vector of row indices in ‘df‘ corresponding to the best model group or groups.

---

get_best_model_names *Get Best Model Names*

Description

Extracts and returns the best engine names from a named list of model workflows.

Usage

```
get_best_model_names(models)
```

Arguments

models	A named list where each element corresponds to an algorithm and contains a list of model workflows. Each workflow should be compatible with `tune::extract_fit_parsnip`.

Details

For each algorithm, the function extracts the engine names from the model workflows using `tune::extract_fit_parsnip`. It then chooses `"randomForest"` if it is available; otherwise, it selects the first non-`NA` engine. If no engine names can be extracted for an algorithm, `NA_character_` is returned.

Value

A named character vector. The names of the vector correspond to the algorithm names, and the values represent the chosen best engine name for that algorithm.

`get_best_workflows` *Get Best Workflows*

Description

Extracts the best workflows from a nested list of model workflows based on the provided best model names.

Usage

```
get_best_workflows(models, best_model_name)
```

Arguments

`models`	A nested list of model workflows.
`best_model_name`	A named character vector of chosen best engines.

`get_default_differences` *Get All Default Differences Summary*

Description

Returns a summary of all differences between fastml and parsnip defaults for the specified algorithms.

Usage

```
get_default_differences(algorithms, task = "classification")
```

Arguments

`algorithms`	Character vector of algorithm names.
`task`	Task type ("classification", "regression", or "survival").

Value

A data frame summarizing the differences.

`get_default_engine` *Get Default Engine*

Description

Returns the default engine corresponding to the specified algorithm.

Usage

```
get_default_engine(algo, task = NULL)
```

Arguments

`algo`	A character string specifying the name of the algorithm. The value should match one of the supported algorithm names.
`task`	Optional task type (e.g., `"classification"`, `"regression"`, or `"survival"`). Used to determine defaults that depend on the task.

Details

The function uses a `switch` statement to select the default engine based on the given algorithm. For survival random forests, the function defaults to `"aorsf"`. If the provided algorithm does not have a defined default engine, the function terminates with an error.

Value

A character string containing the default engine name associated with the provided algorithm.

`get_default_params` *Get Default Parameters for an Algorithm*

Description

Returns a list of default tuning parameters for the specified algorithm based on the task type, number of predictors, and engine.

Usage

```
get_default_params(algo, task, num_predictors = NULL, engine = NULL)
```

Arguments

algo	A character string specifying the algorithm name. Supported values include: "rand_forest", "C5_rules", "xgboost", "lightgbm", "logistic_reg", "multinom_reg", "decision_tree", "svm_linear", "svm_rbf", "nearest_neighbor", "naive_Bayes", "mlp", "deep_learning", "discrim_linear", "discrim_quad", "bag_tree", "elastic_net", "bayes_glm", "pls", "linear_reg", "ridge_reg", "lasso_reg", and "penalized_cox".
task	A character string specifying the task type, typically "classification" or "regression".
num_predictors	An optional numeric value indicating the number of predictors. This value is used to compute default values for parameters such as mtry. Defaults to NULL.
engine	An optional character string specifying the engine to use. If not provided, a default engine is chosen where applicable.

Details

The function employs a switch statement to select and return a list of default parameters tailored for the given algorithm, task, and engine. The defaults vary by algorithm and, in some cases, by engine. For example:

- For "rand_forest", if engine is not provided, it defaults to "ranger". The parameters such as mtry, trees, and min_n are computed based on the task and the number of predictors.
- For "C5_rules", the defaults include trees, min_n, and sample_size.
- For "xgboost" and "lightgbm", default values are provided for parameters like tree depth, learning rate, and sample size.
- For "logistic_reg" and "multinom_reg", the function returns defaults for regularization parameters (penalty and mixture) that vary with the specified engine.
- For "decision_tree", the parameters (such as tree_depth, min_n, and cost_complexity) are set based on the engine (e.g., "rpart", "C5.0", "partykit", "spark").
- Other algorithms, including "svm_linear", "svm_rbf", "nearest_neighbor", "naive_Bayes", "mlp", "deep_learning", "elastic_net", "bayes_glm", "pls", "linear_reg", "ridge_reg", and "lasso_reg", have their respective default parameter lists.

Value

A list of default parameter settings for the specified algorithm. If the algorithm is not recognized, the function returns NULL.

---

get_default_tune_params

Get Default Tuning Parameters

Description

Returns a list of default tuning parameter ranges for a specified algorithm based on the provided training data, outcome label, and engine.

Usage

```
get_default_tune_params(algo, train_data, label, engine)
```

Arguments

algo	A character string specifying the algorithm name. Supported values include: "rand_forest", "C5_rules", "xgboost", "lightgbm", "logistic_reg", "multinom_reg", "decision_tree", "svm_linear", "svm_rbf", "nearest_neighbor", "naive_Bayes", "mlp", "deep_learning", "discrim_linear", "discrim_quad", "bag_tree", "elastic_net", "bayes_glm", "pls", "linear_reg", "ridge_reg", and "lasso_reg".
train_data	A data frame containing the training data.
label	A character string specifying the name of the outcome variable in train_data. This column is excluded when calculating the number of predictors.
engine	A character string specifying the engine to be used for the algorithm. Different engines may have different tuning parameter ranges.

Details

The function first determines the number of predictors by removing the outcome variable (specified by label) from train_data. It then uses a switch statement to select a list of default tuning parameter ranges tailored for the specified algorithm and engine. The tuning ranges have been adjusted for efficiency and may include parameters such as mtry, trees, min_n, and others depending on the algorithm.

Value

A list of tuning parameter ranges for the specified algorithm. If no tuning parameters are defined for the given algorithm, the function returns NULL.

---

get_engine_names *Get Engine Names from Model Workflows*

Description

Extracts and returns a list of unique engine names from a list of model workflows.

Usage

```
get_engine_names(models)
```

Arguments

models	A list where each element is a list of model workflows. Each workflow is expected to contain a fitted model that can be processed with tune::extract_fit_parsnip.

Details

The function applies `tune::extract_fit_parsnip` to each model workflow to extract the fitted model object. It then retrieves the engine name from the model specification (`spec$engine`). If the extraction fails, `NA_character_` is returned for that workflow. Finally, the function removes any duplicate engine names using `unique`.

Value

A list of character vectors. Each vector contains the unique engine names extracted from the corresponding element of `models`.

`get_model_engine_names`

Get Model Engine Names

Description

Extracts and returns a named vector mapping algorithm names to engine names from a nested list of model workflows.

Usage

```
get_model_engine_names(models)
```

Arguments

`models`	A nested list of model workflows. Each inner list should contain model objects from which a fitted model can be extracted using `tune::extract_fit_parsnip`.

Details

The function iterates over a nested list of model workflows and, for each workflow, attempts to extract the fitted model object using `tune::extract_fit_parsnip`. If successful, it retrieves the algorithm name from the first element of the class attribute of the model specification and the engine name from the specification. The results are combined into a named vector.

Value

A named character vector where the names correspond to algorithm names (e.g., `"rand_forest"`, `"logistic_reg"`) and the values correspond to the associated engine names (e.g., `"ranger"`, `"glm"`).

`get_tuning_complexity` *Tuning Complexity Presets*

Description

Returns the configuration for a given tuning complexity level, including grid levels, parameter ranges, and expected computational characteristics.

Usage

```
get_tuning_complexity(
  complexity = c("balanced", "quick", "thorough", "exhaustive")
)
```

Arguments

`complexity`	Character string specifying the tuning complexity level. One of:
	`"quick"` Minimal tuning for fast iteration. 2-3 levels per parameter, narrow ranges. Best for: initial exploration, prototyping, small datasets, time-constrained scenarios. Typical grid size: 4-27 points.
	`"balanced"` Moderate tuning balancing speed and thoroughness. 3-4 levels per parameter, standard ranges. Best for: most production use cases, medium datasets. Typical grid size: 27-256 points. This is the default.
	`"thorough"` Comprehensive tuning for maximum model quality. 4-5 levels per parameter, wide ranges. Best for: final model selection, publications, competitions, when compute time is not a constraint. Typical grid size: 256-3125 points.
	`"exhaustive"` Maximum coverage tuning. 5-7 levels per parameter, very wide ranges. Best for: research, benchmarking, when you need to be certain you've found the best hyperparameters. Warning: Can be very slow. Typical grid size: 1000-10000+ points. Consider using Bayesian tuning instead.

Details

Speed-Robustness Trade-offs

Hyperparameter tuning involves a fundamental trade-off between computational cost and the likelihood of finding optimal hyperparameters:

Level	Grid Size	Time	Robustness	Use Case
quick	4-27	~1x	Low	Prototyping, debugging
balanced	27-256	~10x	Medium	Most production use
thorough	256-3125	~100x	High	Final models, papers
exhaustive	1000-10000+	~1000x	Very High	Research, competitions

Recommendations:

1. **Start with "quick"** during development to iterate fast 2. **Use "balanced"** for most production pipelines 3. **Switch to "thorough"** for final model selection 4. **Consider Bayesian tuning** (‘tuning_strategy = "bayes"‘) for high-dimensional parameter spaces instead of exhaustive grid search 5. **Use adaptive/racing** (‘adaptive = TRUE‘) to early-stop poor configurations

Computational Scaling:

Grid search scales as O(L^P * F * N) where: - L = number of levels per parameter - P = number of parameters being tuned - F = number of cross-validation folds - N = dataset size

For a model with 5 tunable parameters and 10-fold CV: - quick (L=2): 2^5 * 10 = 320 model fits - balanced (L=3): 3^5 * 10 = 2,430 model fits - thorough (L=5): 5^5 * 10 = 31,250 model fits

Value

A list with components:

grid_levels Integer number of levels per parameter for grid search.

bayes_iterations Integer number of iterations for Bayesian tuning.

description Human-readable description of the complexity level.

speed_estimate Relative speed estimate (1 = baseline).

robustness_estimate Relative robustness estimate (1-5 scale).

Examples

```
# Get configuration for balanced tuning
config <- get_tuning_complexity("balanced")
print(config$grid_levels)  # 3

# See all available presets
print_tuning_presets()
```

`get_tuning_params_for_complexity`
Get Tuning Parameters for Complexity Level

Description

Returns algorithm-specific tuning parameter ranges adjusted for the specified complexity level.

Usage

```
get_tuning_params_for_complexity(
  algo,
  train_data,
  label,
  engine,
  complexity = "balanced"
)
```

Arguments

algo	Character string specifying the algorithm name.
train_data	Data frame containing the training data.
label	Character string specifying the outcome variable name.
engine	Character string specifying the engine.
complexity	Character string specifying tuning complexity level.

Details

Parameter ranges are scaled based on the complexity level: - quick: Narrower ranges (70 - balanced: Standard ranges (100 - thorough: Wider ranges (130 - exhaustive: Very wide ranges (150

Value

A list of tuning parameter ranges.

---

interaction_strength *Compute feature interaction strengths for a fastml model*

Description

Uses the ‘iml‘ package to quantify the strength of feature interactions.

Usage

```
interaction_strength(object, data = c("train", "test"), ...)
```

Arguments

object	A ‘fastml‘ object.
data	Character string specifying which data to use: "train" (default) or "test".
...	Additional arguments passed to ‘iml::Interaction‘.

Value

An ‘iml::Interaction‘ object.

Examples

```
## Not run:
data(iris)
iris <- iris[iris$Species != "setosa", ]
iris$Species <- factor(iris$Species)
model <- fastml(data = iris, label = "Species")
interaction_strength(model)

## End(Not run)
```

load_model *Load Model Function*

Description

Loads a trained model object from a file.

Usage

```
load_model(filepath)
```

Arguments

filepath	A string specifying the file path to load the model from.

Value

An object of class fastml.

plot.fastml *Plot Methods for* fastml *Objects*

Description

plot.fastml produces visual diagnostics for a trained fastml object.

Usage

```
## S3 method for class 'fastml'
plot(
  x,
  algorithm = "best",
  type = c("all", "bar", "roc", "calibration", "residual", "learning_curve"),
  ...
)
```

Arguments

x	A fastml object (output of fastml()).
algorithm	Character vector specifying which algorithm(s) to include when generating certain plots (e.g., ROC curves). Defaults to "best".
type	Character vector indicating which plot(s) to produce. Options are:
	"bar" Bar plot of performance metrics across all models/engines.
	"roc" ROC curve(s) for binary classification models.
	"calibration" Calibration plot for the best model(s).

	`"residual"` Residual diagnostics for the best model.
	`"learning_curve"` Learning-curve plot if recorded during training.
	`"all"` Produce all available plots.
`...`	Additional arguments (currently unused).

Details

When `type = "all"`, `plot.fastml` will produce a bar plot of metrics, ROC curves (classification), calibration plot, and residual diagnostics (regression). If you specify a subset of types, only those will be drawn.

Examples

```
## Create a binary classification dataset from iris
data(iris)
iris <- iris[iris$Species != "setosa",]
iris$Species <- factor(iris$Species)

## Fit fastml model on binary classification task
model <- fastml(data = iris, label = "Species", algorithms = c("rand_forest", "svm_rbf"))

## 1. Plot all available diagnostics
plot(model, type = "all")

## 2. Bar plot of performance metrics
plot(model, type = "bar")

## 3. ROC curves (only for classification models)
plot(model, type = "roc")

## 4. Calibration plot (requires 'probably' package)
plot(model, type = "calibration")

## 5. ROC curves for specific algorithm(s) only
plot(model, type = "roc", algorithm = "rand_forest")

## 6. Residual diagnostics (only available for regression tasks)
model <- fastml(data = mtcars, label = "mpg", algorithms = c("linear_reg", "xgboost"))
plot(model, type = "residual")
```

---

plot.fastml_stability *Plot method for fastml_stability objects*

Description

Plot method for fastml_stability objects

Usage

```
## S3 method for class 'fastml_stability'
plot(x, top_n = 15, ...)
```

Arguments

x	A fastml_stability object from explain_stability().
top_n	Integer. Number of top features to display. Default is 15.
...	Additional arguments (ignored).

Value

A ggplot object.

---

plot_ice *Plot ICE curves for a fastml model*

Description

Generates Individual Conditional Expectation (ICE) plots for selected features using the ‘pdp‘ package (ggplot2 engine), and returns both the underlying data and the plot object.

Usage

```
plot_ice(object, features, data = c("train", "test"), target_class = NULL, ...)
```

Arguments

object	A ‘fastml‘ object.
features	Character vector of feature names to plot.
data	Character string specifying which data to use: "train" (default) or "test".
target_class	For classification, which class probability to plot. If NULL (default), uses the positive class determined by the model settings. For multiclass problems, this shows the probability of the specified class vs all others.
...	Additional arguments passed to ‘pdp::partial‘.

Value

A list with two elements: ‘data‘ (the ICE data frame) and ‘plot‘ (the ggplot object).

Examples

```
## Not run:
data(iris)
iris <- iris[iris$Species != "setosa", ]
iris$Species <- factor(iris$Species)
model <- fastml(data = iris, label = "Species")
plot_ice(model, features = "Sepal.Length")

## End(Not run)
```

`predict.fastml` *Predict method for fastml objects*

Description

Generates predictions from a trained ‘fastml‘ object on new data. Supports both single-model and multi-model workflows, and handles classification and regression tasks with optional post-processing and verbosity.

Usage

```
## S3 method for class 'fastml'
predict(
  object,
  newdata,
  type = "auto",
  model_name = NULL,
  verbose = FALSE,
  postprocess_fn = NULL,
  eval_time = NULL,
  ...
)
```

Arguments

`object`	A fitted ‘fastml‘ object created by the ‘fastml()‘ function.
`newdata`	A data frame or tibble containing new predictor data for which to generate predictions.
`type`	Type of prediction to return. One of ‘"auto"‘ (default), ‘"class"‘, ‘"prob"‘, ‘"numeric"‘, ‘"survival"‘, or ‘"risk"‘. - ‘"auto"‘: chooses ‘"class"‘ for classification, ‘"numeric"‘ for regression, and ‘"survival"‘ for survival. - ‘"prob"‘: returns class probabilities (only for classification). - ‘"class"‘: returns predicted class labels. - ‘"numeric"‘: returns predicted numeric values (for regression). - ‘"survival"‘: returns survival probabilities at the supplied ‘eval_time‘ horizons (for survival tasks). - ‘"risk"‘: returns risk scores on the linear predictor scale (for survival tasks).

model_name	(Optional) Name of a specific model to use when 'object$best_model' contains multiple models.
verbose	Logical; if 'TRUE', prints progress messages showing which models are used during prediction.
postprocess_fn	(Optional) A function to apply to the final predictions (e.g., inverse transforms, thresholding).
eval_time	Optional numeric vector of time points (on the original time scale) at which to return survival probabilities when 'type = "survival"'. Required for survival tasks when requesting survival curves.
...	Additional arguments (currently unused).

Value

A vector of predictions, or a named list of predictions (if multiple models are used). If 'postprocess_fn' is supplied, its output will be returned instead.

---

predict_survival *Predict survival probabilities from a survival model*

Description

Predict survival probabilities from a survival model

Usage

```
predict_survival(fit, newdata, times, ...)

## S3 method for class 'fastml_native_survival'
predict_survival(fit, newdata, times, ...)

## S3 method for class 'workflow'
predict_survival(fit, newdata, times, ...)

## Default S3 method:
predict_survival(fit, newdata, times, ...)
```

Arguments

fit	A fitted survival model.
newdata	A data frame of predictors for which to compute survival curves.
times	Numeric vector of evaluation times.
...	Additional arguments passed to methods.

Value

A numeric matrix with one row per observation and one column per time.

print.fastml_stability

Print method for fastml_stability objects

Description

Print method for fastml_stability objects

Usage

```
## S3 method for class 'fastml_stability'
print(x, top_n = 10, ...)
```

Arguments

x	A fastml_stability object from explain_stability().
top_n	Integer. Number of top features to display. Default is 10.
...	Additional arguments (ignored).

Value

The input object invisibly.

print_default_differences

Print Default Differences Table

Description

Prints a formatted table showing differences between fastml and parsnip defaults for the specified task type.

Usage

```
print_default_differences(task = "classification", algorithms = NULL)
```

Arguments

task	Task type ("classification", "regression", or "survival").
algorithms	Optional character vector of algorithms to check. If NULL, checks all available algorithms for the task.

Value

Invisibly returns the differences data frame.

print_tuning_presets *Print Tuning Presets Summary*

Description

Prints a formatted summary of all available tuning complexity presets with their characteristics and recommended use cases.

Usage

```
print_tuning_presets()
```

Value

Invisibly returns a data frame with preset information.

process_model *Process and Evaluate a Model Workflow*

Description

This function processes a fitted model or a tuning result, finalizes the model if tuning was used, makes predictions on the test set, and computes performance metrics depending on the task type (classification or regression). It supports binary and multiclass classification, and handles probabilistic outputs when supported by the modeling engine.

Usage

```
process_model(
  model_obj,
  model_id,
  task,
  test_data,
  label,
  event_class,
  class_threshold = 0.5,
  start_col = NULL,
  time_col = NULL,
  status_col = NULL,
  engine,
  train_data,
  metric,
  eval_times_user = NULL,
  bootstrap_ci = TRUE,
  bootstrap_samples = 500,
```

```
  bootstrap_seed = 1234,
  at_risk_threshold = 0.1,
  survival_metric_convention = "fastml",
  metrics = NULL,
  summaryFunction = NULL,
  precomputed_predictions = NULL,
  multiclass_auc = "macro"
)
```

Arguments

model_obj	A fitted model or a tuning result (‘tune_results‘ object).
model_id	A character identifier for the model (used in warnings).
task	Type of task, either ‘"classification"‘, ‘"regression"‘, or ‘"survival"‘.
test_data	A data frame containing the test data.
label	The name of the outcome variable (as a character string).
event_class	For binary classification, specifies which class is considered the positive class: ‘"first"‘ or ‘"second"‘.
class_threshold	For binary classification, controls how class probabilities are converted into hard class predictions. The default is ‘0.5‘ (standard threshold). Numeric values in (0, 1) set a fixed threshold. Use ‘"auto"‘ to tune a threshold on training data to maximize F1; use ‘"model"‘ to keep the model’s default predictions.
start_col	Optional string. The name of the column specifying the start time in counting process (e.g., ‘(start, stop, event)‘) survival data. Only used when `task = "survival"`.
time_col	String. The name of the column specifying the event or censoring time (the "stop" time in counting process data). Only used when `task = "survival"`.
status_col	String. The name of the column specifying the event status (e.g., 0 for censored, 1 for event). Only used when `task = "survival"`.
engine	A character string indicating the model engine (e.g., ‘"xgboost"‘, ‘"randomForest"‘). Used to determine if class probabilities are supported. If ‘NULL‘, probabilities are skipped.
train_data	A data frame containing the training data, required to refit finalized workflows.
metric	The name of the metric (e.g., ‘"roc_auc"‘, ‘"accuracy"‘, ‘"rmse"‘) used for selecting the best tuning result.
eval_times_user	Optional numeric vector of time horizons at which to evaluate survival Brier scores. When ‘NULL‘, sensible defaults based on the observed follow-up distribution are used.
bootstrap_ci	Logical; if ‘TRUE‘, bootstrap confidence intervals are estimated for performance metrics.
bootstrap_samples	Integer giving the number of bootstrap resamples used when computing confidence intervals.

`bootstrap_seed` Optional integer seed applied before bootstrap resampling to make interval estimates reproducible.

`at_risk_threshold`
Numeric value between 0 and 1 defining the minimum proportion of subjects required to remain at risk when determining the maximum follow-up time used in survival metrics.

`survival_metric_convention`
Character string specifying which survival metric conventions to follow. ‘"fastml"‘ (default) uses fastml's internal defaults for evaluation horizons and t_max. ‘"tidymodels"‘ uses ‘eval_times_user‘ as the explicit evaluation grid and applies yardstick-style Brier/IBS normalization; when ‘eval_times_user‘ is ‘NULL‘, time-dependent Brier metrics are omitted.

`metrics` Optional yardstick metric set (e.g., ‘yardstick::metric_set(yardstick::rmse)‘) used for computing regression performance.

`summaryFunction`
Optional custom classification metric function passed to ‘yardstick::new_class_metric()‘ and included in holdout evaluation.

`precomputed_predictions`
Optional data frame or nested list of previously generated predictions (per algorithm/engine) to reuse instead of re-predicting; primarily used when combining results across engines.

`multiclass_auc` For multiclass ROC AUC, the averaging method to use: ‘"macro"‘ (default, tidymodels) or ‘"macro_weighted"‘. Macro weights each class equally, while macro_weighted weights by class prevalence and can change model rankings on imbalanced data.

Details

- If the input ‘model_obj‘ is a ‘tune_results‘ object, the function finalizes the model using the best hyperparameters according to the specified ‘metric‘, and refits the model on the full training data.

- For classification tasks, performance metrics include accuracy, kappa, sensitivity, specificity, precision, F1-score, and ROC AUC (if probabilities are available).

- For multiclass ROC AUC, the estimator is controlled by ‘multiclass_auc‘.

- For regression tasks, RMSE, R-squared, and MAE are returned.

- For models with missing prediction lengths, a helpful imputation error is thrown to guide data preprocessing.

Value

A list with two elements:

performance A tibble with computed performance metrics.

predictions A tibble with predicted values and corresponding truth values, and probabilities (if applicable).

recommend_tuning_config

Recommend Tuning Configuration

Description

Provides recommendations for tuning configuration based on dataset characteristics and time constraints.

Usage

```
recommend_tuning_config(
  n_rows,
  n_predictors,
  n_algorithms = 1,
  max_time_minutes = 30,
  tuning_strategy = "grid"
)
```

Arguments

n_rows	Number of rows in training data.
n_predictors	Number of predictor variables.
n_algorithms	Number of algorithms to tune.
max_time_minutes	Maximum acceptable tuning time in minutes.
tuning_strategy	Preferred tuning strategy.

Value

A list with recommended configuration.

reset_default_warnings

Reset Default Override Warnings

Description

Resets the tracking of which default override warnings have been shown. Useful for testing or when starting a new analysis session.

Usage

```
reset_default_warnings()
```

sanitize *Clean Column Names or Character Vectors by Removing Special Characters*

Description

This function can operate on either a data frame or a character vector:

- **Data frame**: Detects columns whose names contain any character that is not a letter, number, or underscore, removes colons, replaces slashes with underscores, and spaces with underscores.
- **Character vector**: Applies the same cleaning rules to every element of the vector.

Usage

```
sanitize(x)
```

Arguments

x	A data frame or character vector to be cleaned.

Value

- If x is a data frame: returns a data frame with cleaned column names.
- If x is a character vector: returns a character vector with cleaned elements.

save.fastml *Save Model Function (Deprecated)*

Description

save.fastml is deprecated in favour of save_fastml. The old name resembled an S3 method for base::save(), which is not a generic, leading to dispatch confusion.

Usage

```
save.fastml(model, filepath)
```

Arguments

model	An object of class fastml.
filepath	A string specifying the file path to save the model.

Value

No return value, called for its side effect of saving the model object to a file.

save_fastml *Save Model Function*

Description

Saves the trained model object to a file.

Usage

```
save_fastml(model, filepath)
```

Arguments

model	An object of class fastml.
filepath	A string specifying the file path to save the model.

Value

No return value, called for its side effect of saving the model object to a file.

summary.fastml *Summary Function for fastml (Using yardstick for ROC Curves)*

Description

Summarizes the results of machine learning models trained using the 'fastml' package. Depending on the task type (classification or regression), it provides customized output such as performance metrics, best hyperparameter settings, and confusion matrices. It is designed to be informative and readable, helping users quickly interpret model results.

Usage

```
## S3 method for class 'fastml'
summary(
  object,
  algorithm = "best",
  type = c("all", "metrics", "params", "conf_mat"),
  sort_metric = NULL,
  show_ci = FALSE,
  brier_times = NULL,
  ...
)
```

Arguments

object	An object of class fastml.
algorithm	A vector of algorithm names to display summary. Default is "best".
type	Character vector indicating which outputs to produce. Options are "all" (all available outputs), "metrics" (performance metrics), "params" (best hyperparameters), and "conf_mat" (confusion matrix). Default is "all".
sort_metric	The metric to sort by. Default uses optimized metric.
show_ci	Logical indicating whether to display 95% confidence intervals for performance metrics in survival models. Defaults to FALSE.
brier_times	Optional numeric or character vector that selects which time-specific Brier scores to display for survival models. When NULL (the default), time-specific Brier scores are omitted from the summary.
...	Additional arguments.

Details

For classification tasks, the summary includes metrics such as Accuracy, F1 Score, Kappa, Precision, ROC AUC, Sensitivity, and Specificity. A confusion matrix is also provided for the best model(s). For regression tasks, the summary reports RMSE, R-squared, and MAE.

Users can control the type of output with the 'type' argument: 'metrics' displays model performance metrics. 'params' shows the best hyperparameter settings. 'conf_mat' prints confusion matrices (only for classification). 'all' includes all of the above.

If multiple algorithms are trained, the summary highlights the best model based on the optimized metric. For survival tasks, Harrell's C-index, Uno's C-index, the integrated Brier score, and (when available) the RMST difference are shown by default. Specific Brier(t) horizons can be requested through the brier_times argument.

Value

Prints summary of fastml models.

---

surrogate_tree *Fit a surrogate decision tree for a fastml model*

Description

Builds an interpretable tree approximating the behaviour of the underlying model using the 'iml' package.

Usage

```
surrogate_tree(object, maxdepth = 3, data = c("train", "test"), ...)
```

Arguments

object	A ‘fastml‘ object.
maxdepth	Maximum depth of the surrogate tree. Default 3.
data	Character string specifying which data to use: "train" (default) or "test".
...	Additional arguments passed to ‘iml::TreeSurrogate‘.

Value

An ‘iml::TreeSurrogate‘ object.

Examples

```
## Not run:
data(iris)
iris <- iris[iris$Species != "setosa", ]
iris$Species <- factor(iris$Species)
model <- fastml(data = iris, label = "Species")
surrogate_tree(model)

## End(Not run)
```

---

train_models *Train Specified Machine Learning Algorithms on the Training Data*

Description

Trains specified machine learning algorithms on the preprocessed training data.

Usage

```
train_models(
  train_data,
  label,
  task,
  algorithms,
  resampling_method,
  folds,
  repeats,
  group_cols = NULL,
  block_col = NULL,
  block_size = NULL,
  initial_window = NULL,
  assess_window = NULL,
  skip = 0,
  outer_folds = NULL,
  resamples = NULL,
  tune_params,
```

```
  engine_params = list(),
  metric,
  summaryFunction = NULL,
  seed = 123,
  recipe,
  use_default_tuning = FALSE,
  tuning_strategy = "grid",
  tuning_iterations = 10,
  tuning_complexity = "balanced",
  grid_levels = 3L,
  early_stopping = FALSE,
  adaptive = FALSE,
  algorithm_engines = NULL,
  use_parsnip_defaults = FALSE,
  warn_engine_defaults = TRUE,
  n_cores = 1,
  verbose = FALSE,
  event_class = "first",
  class_threshold = 0.5,
  start_col = NULL,
  time_col = NULL,
  status_col = NULL,
  eval_times = NULL,
  at_risk_threshold = 0.1,
  survival_metric_convention = "fastml",
  audit_env = NULL,
  multiclass_auc = "macro",
  store_fold_models = FALSE
)
```

Arguments

train_data	Preprocessed training data frame.
label	Name of the target variable.
task	Type of task: "classification", "regression", or "survival".
algorithms	Vector of algorithm names to train.
resampling_method	Resampling method for cross-validation. Supported options include standard "cv", "repeatedcv", and "boot", as well as grouped resampling via "grouped_cv", blocked/rolling schemes via "blocked_cv" or "rolling_origin", nested resampling via "nested_cv", and the passthrough "none" option.
folds	Number of folds for cross-validation.
repeats	Number of times to repeat cross-validation (only applicable for methods like "repeatedcv").
group_cols	Optional character vector of grouping columns used with ‘resampling_method = "grouped_cv"‘. For classification problems the outcome column is used to request grouped stratification where supported; if class imbalance prevents strati-

fication, grouped folds are still created and a warning is emitted to document the limitation.

`block_col` Optional name of the ordering column used with blocked or rolling resampling.

`block_size` Optional integer specifying the block size for ‘resampling_method = "blocked_cv"‘.

`initial_window` Optional integer specifying the initial window size for rolling resampling.

`assess_window` Optional integer specifying the assessment window size for rolling resampling.

`skip` Optional integer number of resamples to skip between rolling resamples.

`outer_folds` Optional integer specifying the number of outer folds for ‘resampling_method = "nested_cv"‘.

`resamples` Optional rsample object. If provided, custom resampling splits will be used instead of those created internally.

`tune_params` A named list of tuning ranges. For each algorithm, supply a list of engine-specific parameter values, e.g. `list(rand_forest = list(ranger = list(mtry = c(1, 3))))`.

`engine_params` A named list of fixed engine-level arguments passed directly to the model fitting call for each algorithm/engine combination. Use this to control options like `ties = "breslow"` for Cox models or `importance = "impurity"` for ranger. Unlike `tune_params`, these values are not tuned over a grid.

`metric` The performance metric to optimize. For classification, options include `"accuracy"`, `"roc_auc"`, `"logloss"`, `"brier_score"`, and `"ece"` (plus other class metrics).

`summaryFunction`
A custom summary function for model evaluation. Default is `NULL`.

`seed` An integer value specifying the random seed for reproducibility.

`recipe` A recipe object for preprocessing.

`use_default_tuning`
Logical; if `TRUE` and `tune_params` is `NULL`, tuning is performed using default grids. Tuning also occurs when custom `tune_params` are supplied. When `FALSE` and no custom parameters are given, the model is fitted once with default settings.

`tuning_strategy`
A string specifying the tuning strategy. Must be one of `"grid"`, `"bayes"`, or `"none"`. Adaptive methods may be used with `"grid"`. If `"none"` is selected, the workflow is fitted directly without tuning. If custom `tune_params` are supplied with `tuning_strategy = "none"`, they will be ignored with a warning.

`tuning_iterations`
Number of iterations for Bayesian tuning. Ignored when `tuning_strategy` is not `"bayes"`; validation occurs only for the Bayesian strategy.

`tuning_complexity`
Character string specifying tuning complexity preset. One of "quick", "balanced", "thorough", or "exhaustive". Controls both grid density and parameter range width.

`grid_levels` Integer specifying number of levels per parameter for grid search. Higher values create denser grids but increase computation exponentially (grid size = levels^n_params).

`early_stopping` Logical for early stopping in Bayesian tuning.

`adaptive` Logical indicating whether to use adaptive/racing methods.

`algorithm_engines`
A named list specifying the engine to use for each algorithm.

`use_parsnip_defaults`
Logical. If `TRUE`, use parsnip's default engines instead of fastml's optimized defaults. Default is `FALSE`.

`warn_engine_defaults`
Logical. If `TRUE` (default), warn when fastml's default engine differs from parsnip's default.

`n_cores` Integer number of cores requested for parallel processing. Used to decide whether tuning/resampling should run in parallel and to configure engine thread settings when supported.

`verbose` Logical. If `TRUE`, print informational messages about engine selection and parameter overrides.

`event_class` Character string identifying the positive class when computing classification metrics ("first" or "second").

`class_threshold`
For binary classification, controls how class probabilities are converted into hard class predictions during evaluation. The default is '0.5' (standard threshold). Numeric values in (0, 1) set a fixed threshold. Use '"auto"' to tune a threshold on training data to maximize F1; use '"model"' to keep the model's default predictions.

`start_col` Optional name of the survival start time column passed through to downstream evaluation helpers.

`time_col` Optional name of the survival stop time column.

`status_col` Optional name of the survival status/event column.

`eval_times` Optional numeric vector of time horizons for survival metrics.

`at_risk_threshold`
Numeric cutoff used to determine the evaluation window for survival metrics within guarded resampling.

`survival_metric_convention`
Character string specifying which survival metric conventions to follow. '"fastml"' (default) uses fastml's internal defaults for evaluation horizons and t_max. '"tidymodels"' uses 'eval_times' as the explicit evaluation grid and applies yardstick-style Brier/IBS normalization; when 'eval_times' is 'NULL', time-dependent Brier metrics are omitted.

`audit_env` Internal environment that tracks security audit findings when custom preprocessing hooks are executed. Typically supplied by `fastml()` and should be left as `NULL` when calling `train_models()` directly.

`multiclass_auc` For multiclass ROC AUC, the averaging method to use: '"macro"' (default, tidymodels) or '"macro_weighted"'. Macro weights each class equally, while macro_weighted weights by class prevalence and can change model rankings on imbalanced data.

`store_fold_models`
Logical. If `TRUE`, store the fitted fold models during resampling for later inspection or stability analysis.

Value

A list of trained model objects.

---

tuning_config *Tuning Configuration and Complexity Presets*

Description

Functions and presets for configuring hyperparameter tuning grids with explicit speed-robustness trade-offs.

---

validate_defaults_registry *Validate Defaults Registry Against Parsnip*

Description

Compares the hardcoded parsnip default engines in fastml's registry against the actual defaults reported by `parsnip::show_engines()`. Returns a list of any mismatches found, which may indicate that parsnip has updated its defaults since fastml's registry was last updated.

Usage

```
validate_defaults_registry()
```

Details

This function queries parsnip for model specifications and compares against the hardcoded `parsnip_defaults` list in `get_parsnip_default_engine()`. Mismatches may occur when:

- Parsnip updates its default engine for a model type
- New engines are added to parsnip that become the new default
- fastml's registry has not been updated after a parsnip release

This validation is intended for package maintenance and testing purposes.

Value

A list of mismatches. Each element is a list with components:

algorithm The algorithm name.

fastml_default The default engine recorded in fastml's registry.

parsnip_default The actual default engine from parsnip.

Returns an empty list if no mismatches are found.

Examples

```
## Not run:
mismatches <- validate_defaults_registry()
if (length(mismatches) > 0) {
  message("Found ", length(mismatches), " mismatch(es) with parsnip defaults")
}

## End(Not run)
```

`warn_default_override` *Warn About Default Overrides*

Description

Issues a warning if fastml defaults differ from parsnip defaults and the warning hasn't been shown yet in this session.

Usage

```
warn_default_override(
  algo,
  task,
  fastml_engine,
  fastml_params = NULL,
  verbose = FALSE,
  warn_once = TRUE
)
```

Arguments

`algo`	Algorithm name.
`task`	Task type.
`fastml_engine`	fastml's default engine for this algorithm.
`fastml_params`	fastml's default parameters (optional).
`verbose`	If TRUE, always show the message (as a message, not warning).
`warn_once`	If TRUE (default), only warn once per algorithm per session.

Value

Invisibly returns the comparison result.

Index