Package: E2E (via r-universe)

May 18, 2026

Title Ensemble Learning Framework for Diagnostic and Prognostic Modeling

Version 0.1.3

Description Provides a framework to build and evaluate diagnosis or prognosis models using stacking, voting, and bagging ensemble techniques with various base learners. The package also includes tools for visualization and interpretation of models. The development version of the package is available on 'GitHub' at <https://github.com/xiaojie0519/E2E>. The methods are based on the foundational work of Breiman (1996) <doi:10.1007/BF00058655> on bagging and Wolpert (1992) <doi:10.1016/S0893-6080(05)80023-1> on stacking.

License MIT + file LICENSE

Encoding UTF-8

URL https://xiaojie0519.github.io/E2E/

BugReports https://github.com/xiaojie0519/E2E/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports caret, dplyr, tidyr, gbm, ggplot2, glmnet, magrittr, MASS, patchwork, pROC, PRROC, randomForestSRC, readr, RSNNS, shapviz, survival, survivalROC, survminer, xgboost, plsRcox, cowplot,

Suggests ada, doParallel, e1071, kernlab, klaR, knitr, nnet, randomForest, RColorBrewer, rmarkdown, rpart, survcomp,

Depends R (>= 3.5)

LazyData true

VignetteBuilder knitr

Language en

Config/pak/sysreqs cmake libfreetype6-dev libglpk-dev libglu1-mesa-dev make texlive libicu-dev libjpeg-dev libpng-dev libuv1-dev libxml2-dev libgl1-mesa-dev libssl-dev libx11-dev zlib1g-dev

Repository https://xiaojie0519.r-universe.dev

Date/Publication 2026-03-19 15:25:30 UTC

RemoteUrl https://github.com/xiaojie0519/e2e

RemoteRef HEAD

RemoteSha 608bcfd714bba0c623f7f58267a2aa89fb74ea6c

Contents

`apply_dia` *Apply a Trained Model to New Data*

Description

Applies a trained diagnostic model (single or ensemble) to a new dataset to generate predictions. It can handle various model objects created by the package, including single caret models, Bagging, Stacking, Voting, and EasyEnsemble objects.

Usage

```
apply_dia(
  trained_model_object,
  new_data,
  label_col_name = NULL,
  pos_class = "Positive",
  neg_class = "Negative"
)
```

Arguments

trained_model_object	A trained model object from models_dia, bagging_dia, stacking_dia, voting_dia, or imbalance_dia.
new_data	A data frame containing the new samples for prediction. The first column must be the sample ID.
label_col_name	An optional character string specifying the name of the column in new_data that contains the true labels. **If NULL (the default), the function will assume the second column is the label column.** To explicitly prevent label extraction (e.g., for data without labels), provide NA.
pos_class	A character string for the positive class label used in the model's probability predictions. **Defaults to "Positive".**
neg_class	A character string for the negative class label. This parameter is mainly for consistency, as prediction focuses on pos_class probability. **Defaults to "Negative".**

Value

A data frame with three columns: sample (the sample IDs), label (the true labels from new_data, or NA if not available/specified), and score (the predicted probability for the positive class).

Examples

```
# Assuming `bagging_results` and `test_dia` are available from previous steps
# bagging_model <- bagging_results$model_object

# Example 1: Default behavior - use the second column of test_dia as label
# predictions <- apply_dia(
#   trained_model_object = bagging_model,
#   new_data = test_dia
# )

# Example 2: Explicitly specify the label column by name
# predictions_explicit <- apply_dia(
#   trained_model_object = bagging_model,
#   new_data = test_dia,
#   label_col_name = "outcome"
# )

# Example 3: Predict on data without labels
# test_data_no_labels <- test_dia[, -2] # Remove outcome column
# predictions_no_label <- apply_dia(
#   trained_model_object = bagging_model,
#   new_data = test_data_no_labels,
#   label_col_name = NA # Explicitly disable label extraction
# )
```

apply_pro *Apply Prognostic Model to New Data*

Description

Generates risk scores for new patients using a trained model.

Usage

```
apply_pro(trained_model_object, new_data, time_unit = "day")
```

Arguments

trained_model_object	A trained object (class pro_model).
new_data	Data frame of new patients.
time_unit	Time unit for data preparation.

Value

Data frame with IDs, outcomes (if available), and risk scores.

bagging_dia *Train a Bagging Diagnostic Model*

Description

Implements a Bagging (Bootstrap Aggregating) ensemble for diagnostic models. It trains multiple base models on bootstrapped samples of the training data and aggregates their predictions by averaging probabilities.

Usage

```
bagging_dia(
  data,
  base_model_name,
  n_estimators = 50,
  subset_fraction = 0.632,
  tune_base_model = FALSE,
  threshold_choices = "default",
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Positive",
  new_negative_label = "Negative",
  seed = 456
)
```

Arguments

`data`	A data frame where the first column is the sample ID, the second is the outcome label, and subsequent columns are features.
`base_model_name`	A character string, the name of the base diagnostic model to use (e.g., "rf", "lasso"). This model must be registered.
`n_estimators`	An integer, the number of base models to train.
`subset_fraction`	A numeric value between 0 and 1, the fraction of samples to bootstrap for each base model.
`tune_base_model`	Logical, whether to enable tuning for each base model.
`threshold_choices`	A character string (e.g., "f1", "youden", "default") or a numeric value (0-1) for determining the evaluation threshold for the ensemble.
`positive_label_value`	A numeric or character value in the raw data representing the positive class.
`negative_label_value`	A numeric or character value in the raw data representing the negative class.
`new_positive_label`	A character string, the desired factor level name for the positive class (e.g., "Positive").
`new_negative_label`	A character string, the desired factor level name for the negative class (e.g., "Negative").
`seed`	An integer, for reproducibility.

Value

A list containing the `model_object`, `sample_score`, and `evaluation_metrics`.

See Also

`initialize_modeling_system_dia`, `evaluate_model_dia`

Examples

```
# This example assumes your package includes a dataset named 'train_dia'.
# If not, create a toy data frame first.
if (exists("train_dia")) {
  initialize_modeling_system_dia()

  bagging_rf_results <- bagging_dia(
    data = train_dia,
    base_model_name = "rf",
    n_estimators = 5, # Reduced for a quick example
    threshold_choices = "youden",
```

```
    positive_label_value = 1,
    negative_label_value = 0,
    new_positive_label = "Case",
    new_negative_label = "Control"
  )
  print_model_summary_dia("Bagging (RF)", bagging_rf_results)
}
```

---

bagging_pro *Train Bagging Ensemble for Prognosis*

Description

Implements Bootstrap Aggregating (Bagging) for survival models. It trains multiple base models on bootstrapped subsets and averages the risk scores. This method reduces variance and improves stability.

Usage

```
bagging_pro(
  data,
  base_model_name,
  n_estimators = 10,
  subset_fraction = 0.632,
  tune_base_model = FALSE,
  time_unit = "day",
  years_to_evaluate = c(1, 3, 5),
  seed = 456
)
```

Arguments

data	Input data frame (ID, Status, Time, Features).
base_model_name	Character string name of the base model (e.g., "rsf_pro").
n_estimators	Integer. Number of bootstrap iterations.
subset_fraction	Numeric (0-1). Fraction of data to sample in each iteration.
tune_base_model	Logical. Whether to tune each base model (computationally expensive).
time_unit	Time unit of the input data.
years_to_evaluate	Numeric vector of years for time-dependent AUC evaluation.
seed	Integer seed for reproducibility.

Value

A list containing the ensemble object, sample scores, and evaluation metrics.

`calculate_metrics_at_threshold_dia` *Calculate Classification Metrics at a Specific Threshold*

Description

Calculates various classification performance metrics (Accuracy, Precision, Recall, F1-score, Specificity, True Positives, etc.) for binary classification at a given probability threshold.

Usage

```
calculate_metrics_at_threshold_dia(
  prob_positive,
  y_true,
  threshold,
  pos_class,
  neg_class
)
```

Arguments

`prob_positive`	A numeric vector of predicted probabilities for the positive class.
`y_true`	A factor vector of true class labels.
`threshold`	A numeric value between 0 and 1, the probability threshold above which a prediction is considered positive.
`pos_class`	A character string, the label for the positive class.
`neg_class`	A character string, the label for the negative class.

Value

A list containing:

- `Threshold`: The threshold used.
- `Accuracy`: Overall prediction accuracy.
- `Precision`: Precision for the positive class.
- `Recall`: Recall (Sensitivity) for the positive class.
- `F1`: F1-score for the positive class.
- `Specificity`: Specificity for the negative class.
- `TP`, `TN`, `FP`, `FN`, `N`: Counts of True Positives, True Negatives, False Positives, False Negatives, and total samples.

Examples

```
y_true_ex <- factor(c("Negative", "Positive", "Positive", "Negative", "Positive"),
                    levels = c("Negative", "Positive"))
prob_ex <- c(0.1, 0.8, 0.6, 0.3, 0.9)
metrics <- calculate_metrics_at_threshold_dia(
  prob_positive = prob_ex,
  y_true = y_true_ex,
  threshold = 0.5,
  pos_class = "Positive",
  neg_class = "Negative"
)
print(metrics)
```

---

dt_dia *Train a Decision Tree Model for Classification*

Description

Trains a single Decision Tree model using `caret::train` (via `rpart` method) for binary classification.

Usage

```
dt_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

`X`	A data frame of features.
`y`	A factor vector of class labels.
`tune`	Logical, whether to perform hyperparameter tuning for `cp` (complexity parameter) (if `TRUE`) or use a fixed value (if `FALSE`).
`cv_folds`	An integer, the number of cross-validation folds for `caret`.

Value

A `caret::train` object representing the trained Decision Tree model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))
```

```
# Train the model
dt_model <- dt_dia(X_toy, y_toy)
print(dt_model)
```

---

en_dia *Train an Elastic Net (L1 and L2 Regularized Logistic Regression) Model for Classification*

Description

Trains an Elastic Net-regularized logistic regression model using `caret::train` (via `glmnet` method) for binary classification.

Usage

```
en_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

`X`	A data frame of features.
`y`	A factor vector of class labels.
`tune`	Logical, whether to perform hyperparameter tuning for `lambda` (if `TRUE`) or use a fixed value (if `FALSE`). `alpha` is fixed at 0.5 for Elastic Net.
`cv_folds`	An integer, the number of cross-validation folds for `caret`.

Value

A `caret::train` object representing the trained Elastic Net model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
en_model <- en_dia(X_toy, y_toy)
print(en_model)
```

en_pro *Train Elastic Net Cox Model*

Description

Fits a Cox model with Elastic Net regularization (mixture of L1 and L2 penalties). Alpha is fixed at 0.5.

Usage

```
en_pro(X, y_surv, tune = FALSE)
```

Arguments

X	A data frame of predictors.
y_surv	A Surv object containing time and status.
tune	Logical. If TRUE, performs internal tuning (currently handled by cv.glmnet automatically).

Value

An object of class survival_glmnet and pro_model.

evaluate_model_dia *Evaluate Diagnostic Model Performance*

Description

Evaluates the performance of a trained diagnostic model using various metrics relevant to binary classification, including AUROC, AUPRC, and metrics at an optimal or specified probability threshold.

Usage

```
evaluate_model_dia(
  model_obj = NULL,
  X_data = NULL,
  y_data,
  sample_ids,
  threshold_choices = "default",
  pos_class,
  neg_class,
  precomputed_prob = NULL,
  y_original_numeric = NULL
)
```

Arguments

model_obj	A trained model object (typically a caret::train object or a list from an ensemble like Bagging). Can be NULL if precomputed_prob is provided.
X_data	A data frame of features corresponding to the data used for evaluation. Required if model_obj is provided and precomputed_prob is NULL.
y_data	A factor vector of true class labels for the evaluation data.
sample_ids	A vector of sample IDs for the evaluation data.
threshold_choices	A character string specifying the thresholding strategy ("default", "f1", "youden") or a numeric probability threshold value (0-1).
pos_class	A character string, the label for the positive class.
neg_class	A character string, the label for the negative class.
precomputed_prob	Optional. A numeric vector of precomputed probabilities for the positive class. If provided, model_obj and X_data are not used for score derivation.
y_original_numeric	Optional. The original numeric/character vector of labels. If not provided, it's inferred from y_data using global pos_label_value and neg_label_value.

Value

A list containing:

- sample_score: A data frame with sample (ID), label (original numeric), and score (predicted probability for positive class).
- evaluation_metrics: A list of performance metrics:
  - Threshold_Strategy: The strategy used for threshold selection.
  - _Threshold: The chosen probability threshold.
  - Accuracy, Precision, Recall, F1, Specificity: Metrics calculated at _Threshold.
  - AUROC: Area Under the Receiver Operating Characteristic curve.
  - AUROC_95CI_Lower, AUROC_95CI_Upper: 95% confidence interval for AUROC.
  - AUPRC: Area Under the Precision-Recall curve.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))
ids_toy <- paste0("Sample", 1:n_obs)

# 2. Train a model
```

```
rf_model <- rf_dia(X_toy, y_toy)

# 3. Evaluate the model using F1-score optimal threshold
eval_results <- evaluate_model_dia(
  model_obj = rf_model,
  X_data = X_toy,
  y_data = y_toy,
  sample_ids = ids_toy,
  threshold_choices = "f1",
  pos_class = "Case",
  neg_class = "Control"
)
str(eval_results)
```

`evaluate_model_pro` *Evaluate Prognostic Model Performance*

Description

Comprehensive evaluation of survival models using:

1. Harrell's Concordance Index (C-index).
2. Time-dependent Area Under the ROC Curve (AUROC) at specified years.
3. Kaplan-Meier analysis comparing high vs. low risk groups (based on median split).

Usage

```
evaluate_model_pro(
  trained_model_obj = NULL,
  X_data = NULL,
  Y_surv_obj,
  sample_ids,
  years_to_evaluate = c(1, 3, 5),
  precomputed_score = NULL,
  meta_normalize_params = NULL
)
```

Arguments

`trained_model_obj`	A trained model object (optional if precomputed_score provided).
`X_data`	Features for prediction (optional if precomputed_score provided).
`Y_surv_obj`	True survival object.
`sample_ids`	Vector of IDs.
`years_to_evaluate`	Numeric vector of years for time-dependent AUC.

`precomputed_score`
: Numeric vector of pre-calculated risk scores.

`meta_normalize_params`
: Internal use.

Value

A list containing a dataframe of scores and a list of evaluation metrics.

`evaluate_predictions_dia` *Evaluate Predictions from a Data Frame*

Description

Evaluates model performance from a data frame of predictions, calculating metrics like AUROC, AUPRC, F1 score, etc. This function is designed for use with prediction results, such as the output from `apply_dia`.

Usage

```
evaluate_predictions_dia(
  prediction_df,
  threshold_choices = "default",
  pos_class = "Positive",
  neg_class = "Negative"
)
```

Arguments

`prediction_df`	A data frame containing predictions. Must contain the columns `sample`, `label` (true labels), and `score` (predicted probabilities).
`threshold_choices`	A character string specifying the thresholding strategy ("default", "f1", "youden") or a numeric probability threshold value (0-1).
`pos_class`	A character string for the positive class label used in reporting. **Defaults to `"Positive"`.**
`neg_class`	A character string for the negative class label used in reporting. **Defaults to `"Negative"`.**

Details

This function strictly requires the `label` column in `prediction_df` to adhere to the following format:

- `1`: Represents the positive class.
- `0`: Represents the negative class.

- NA: Will be ignored during calculation.

The function will stop with an error if any other values are found in the label column.

Value

A named list containing all calculated performance metrics.

Examples

```
# # Create a sample prediction data frame
# predictions_df <- data.frame(
#   sample = 1:10,
#   label = c(1, 0, 1, 1, 0, 0, 1, 0, 1, 0),
#   score = c(0.9, 0.2, 0.8, 0.6, 0.3, 0.4, 0.95, 0.1, 0.7, 0.5)
# )
#
# # Evaluate the predictions using the 'f1' threshold strategy
# evaluation_results <- evaluate_predictions_dia(
#   prediction_df = predictions_df,
#   threshold_choices = "f1"
# )
#
# print(evaluation_results)
```

---

evaluate_predictions_pro

Evaluate External Predictions

Description

Calculates performance metrics for external prediction sets.

Usage

```
evaluate_predictions_pro(prediction_df, years_to_evaluate = c(1, 3, 5))
```

Arguments

prediction_df	Data frame with columns time, outcome, score, ID.
years_to_evaluate	Years for AUC.

Value

List of evaluation metrics.

figure_dia *Plot Diagnostic Model Evaluation Figures*

Description

Generates and returns a ggplot object for Receiver Operating Characteristic (ROC) curves, Precision-Recall (PRC) curves, or confusion matrices.

Usage

```
figure_dia(type, data, file = NULL)
```

Arguments

type	String, specifies the type of plot to generate. Options are "roc", "prc", or "matrix".
data	A list object containing model evaluation results. It must include: • sample_score: A data frame with "label" (0/1) and "score" columns. • evaluation_metrics: A list with a "Final_Threshold" or "Final_Threshold" value.
file	Optional. A string specifying the path to save the plot (e.g., "plot.png"). If NULL (the default), the plot object is returned instead of being saved.

Value

A ggplot object. If the file argument is provided, the plot is also saved to the specified path.

Examples

```
# Create example data for a diagnostic model
external_eval_example_dia <- list(
  sample_score = data.frame(
    ID = paste0("S", 1:100),
    label = sample(c(0, 1), 100, replace = TRUE),
    score = runif(100, 0, 1)
  ),
  evaluation_metrics = list(
    Final_Threshold = 0.53
  )
)

# Generate an ROC curve plot object
roc_plot <- figure_dia(type = "roc", data = external_eval_example_dia)
# To display the plot, simply run:
# print(roc_plot)

# Generate a PRC curve and save it to a temporary file
# tempfile() creates a safe, temporary path as required by CRAN
```

```
temp_prc_path <- tempfile(fileext = ".png")
figure_dia(type = "prc", data = external_eval_example_dia, file = temp_prc_path)

# Generate a Confusion Matrix plot
matrix_plot <- figure_dia(type = "matrix", data = external_eval_example_dia)
```

`figure_pro` *Plot Prognostic Model Evaluation Figures*

Description

Generates and returns a ggplot object for Kaplan-Meier (KM) survival curves or time-dependent ROC curves.

Usage

```
figure_pro(type, data, file = NULL, time_unit = "days")
```

Arguments

`type`	"km" or "tdroc"
`data`	list with: • sample_score: data.frame(time, outcome, score) • evaluation_metrics: for "km" needs KM_Cutoff; for "tdroc" needs AUROC_Years (numeric years like c(1,3,5), OR a named vector/list like c('1'=0.74,'3'=0.82,'5'=0.85))
`file`	optional path to save
`time_unit`	"days" (default), "months", or "years" for df$time

Value

ggplot object

`figure_shap` *Generate and Plot SHAP Explanation Figures*

Description

Creates SHAP (SHapley Additive exPlanations) plots to explain feature contributions by training a surrogate model on the original model's scores.

Usage

```
figure_shap(data, raw_data, target_type, file = NULL, model_type = "xgboost")
```

Arguments

data	A list containing sample_score, a data frame with sample IDs and score.
raw_data	A data frame with original features. The first column must be the sample ID.
target_type	String, the analysis type: "diagnosis" or "prognosis". This determines which columns in raw_data are treated as features.
file	Optional. A string specifying the path to save the plot. If NULL (default), the plot object is returned.
model_type	String, the surrogate model for SHAP calculation. "xgboost" (default) or "lasso".

Value

A patchwork object combining SHAP summary and importance plots. If file is provided, the plot is also saved.

Examples

```
# --- Example for a Diagnosis Model ---
set.seed(123)
train_dia_data <- data.frame(
  SampleID = paste0("S", 1:100),
  Label = sample(c(0, 1), 100, replace = TRUE),
  FeatureA = rnorm(100, 10, 2),
  FeatureB = runif(100, 0, 5)
)
model_results <- list(
  sample_score = data.frame(ID = paste0("S", 1:100), score = runif(100, 0, 1))
)

# Generate SHAP plot object
shap_plot <- figure_shap(
  data = model_results,
  raw_data = train_dia_data,
  target_type = "diagnosis",
  model_type = "xgboost"
)
# To display the plot:
# print(shap_plot)
```

---

find_optimal_threshold_dia

Find Optimal Probability Threshold

Description

Determines an optimal probability threshold for binary classification based on maximizing F1-score or Youden's J statistic.

Usage

```
find_optimal_threshold_dia(
  prob_positive,
  y_true,
  type = c("f1", "youden"),
  pos_class,
  neg_class
)
```

Arguments

prob_positive	A numeric vector of predicted probabilities for the positive class.
y_true	A factor vector of true class labels.
type	A character string, specifying the optimization criterion: "f1" for F1-score or "youden" for Youden's J statistic (Sensitivity + Specificity - 1).
pos_class	A character string, the label for the positive class.
neg_class	A character string, the label for the negative class.

Value

A numeric value, the optimal probability threshold.

Examples

```
y_true_ex <- factor(c("Negative", "Positive", "Positive", "Negative", "Positive"),
                    levels = c("Negative", "Positive"))
prob_ex <- c(0.1, 0.8, 0.6, 0.3, 0.9)

# Find threshold maximizing F1-score
opt_f1_threshold <- find_optimal_threshold_dia(
  prob_positive = prob_ex,
  y_true = y_true_ex,
  type = "f1",
  pos_class = "Positive",
  neg_class = "Negative"
)
print(opt_f1_threshold)

# Find threshold maximizing Youden's J
opt_youden_threshold <- find_optimal_threshold_dia(
  prob_positive = prob_ex,
  y_true = y_true_ex,
  type = "youden",
  pos_class = "Positive",
  neg_class = "Negative"
)
print(opt_youden_threshold)
```

gbm_dia *Train a Gradient Boosting Machine (GBM) Model for Classification*

Description

Trains a Gradient Boosting Machine (GBM) model using `caret::train` for binary classification.

Usage

```
gbm_dia(X, y, tune = FALSE, cv_folds = 5, tune_length = 10)
```

Arguments

X	A data frame of features.
y	A factor vector of class labels.
tune	Logical, whether to perform hyperparameter tuning for `interaction.depth`, `n.trees`, and `shrinkage` (if `TRUE`) or use fixed values (if `FALSE`).
cv_folds	An integer, the number of cross-validation folds for `caret`.
tune_length	An integer, the number of random parameter combinations to try when tune=TRUE. Only used when search="random". Default is 20.

Value

A `caret::train` object representing the trained GBM model.

Examples

```
set.seed(42)
n_obs <- 200
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model with default parameters
gbm_model <- gbm_dia(X_toy, y_toy)
print(gbm_model)

# Train with extensive tuning (random search)
gbm_model_tuned <- gbm_dia(X_toy, y_toy, tune = TRUE, tune_length = 30)
print(gbm_model_tuned)
```

gbm_pro *Train Gradient Boosting Machine (GBM) for Survival*

Description

Fits a stochastic gradient boosting model using the Cox Partial Likelihood distribution. Supports random search for hyperparameter optimization.

Usage

```
gbm_pro(X, y_surv, tune = FALSE, cv.folds = 5, max_tune_iter = 10)
```

Arguments

X	A data frame of predictors.
y_surv	A Surv object.
tune	Logical. If TRUE, performs random search.
cv.folds	Integer. Number of cross-validation folds.
max_tune_iter	Integer. Maximum iterations for random search.

Value

An object of class survival_gbm and pro_model.

---

get_registered_models_dia *Get Registered Diagnostic Models*

Description

Retrieves a list of all diagnostic model functions currently registered in the internal environment.

Usage

```
get_registered_models_dia()
```

Value

A named list where names are the registered model names and values are the corresponding model functions.

See Also

register_model_dia, initialize_modeling_system_dia

Examples

```
# Ensure system is initialized to see the default models
initialize_modeling_system_dia()
models <- get_registered_models_dia()
# See available model names
print(names(models))
```

`get_registered_models_pro` *Get Registered Prognostic Models*

Description

Retrieves the list of available models.

Usage

```
get_registered_models_pro()
```

Value

Named list of functions.

`imbalance_dia` *Train an EasyEnsemble Model for Imbalanced Classification*

Description

Implements the EasyEnsemble algorithm. It trains multiple base models on balanced subsets of the data (by undersampling the majority class) and aggregates their predictions.

Usage

```
imbalance_dia(
  data,
  base_model_name = "rf",
  n_estimators = 10,
  tune_base_model = FALSE,
  threshold_choices = "default",
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Positive",
  new_negative_label = "Negative",
  seed = 456
)
```

Arguments

`data`	A data frame where the first column is the sample ID, the second is the outcome label, and subsequent columns are features.
`base_model_name`	A character string, the name of the base diagnostic model to use (e.g., "xb", "rf"). This model must be registered.
`n_estimators`	An integer, the number of base models to train (number of subsets).
`tune_base_model`	Logical, whether to enable tuning for each base model.
`threshold_choices`	A character string (e.g., "f1", "youden", "default") or a numeric value (0-1) for determining the evaluation threshold for the ensemble.
`positive_label_value`	A numeric or character value in the raw data representing the positive class.
`negative_label_value`	A numeric or character value in the raw data representing the negative class.
`new_positive_label`	A character string, the desired factor level name for the positive class (e.g., "Positive").
`new_negative_label`	A character string, the desired factor level name for the negative class (e.g., "Negative").
`seed`	An integer, for reproducibility.

Value

A list containing the `model_object`, `sample_score`, and `evaluation_metrics`.

See Also

`initialize_modeling_system_dia`, `evaluate_model_dia`

Examples

```
# 1. Initialize the modeling system
initialize_modeling_system_dia()

# 2. Create an imbalanced toy dataset
set.seed(42)
n_obs <- 100
n_minority <- 10
data_imbalanced_toy <- data.frame(
  ID = paste0("Sample", 1:n_obs),
  Status = c(rep(1, n_minority), rep(0, n_obs - n_minority)),
  Feat1 = rnorm(n_obs),
  Feat2 = runif(n_obs)
)
```

```
# 3. Run the EasyEnsemble algorithm
# n_estimators is reduced for a quick example
easyensemble_results <- imbalance_dia(
  data = data_imbalanced_toy,
  base_model_name = "rf",
  n_estimators = 3,
  threshold_choices = "f1"
)
print_model_summary_dia("EasyEnsemble (RF)", easyensemble_results)
```

`initialize_modeling_system_dia` *Initialize Diagnostic Modeling System*

Description

Initializes the diagnostic modeling system by loading required packages and registering default diagnostic models (Random Forest, XGBoost, SVM, MLP, Lasso, Elastic Net, Ridge, LDA, QDA, Naive Bayes, Decision Tree, GBM). This function should be called once before using `models_dia()` or ensemble methods.

Usage

```
initialize_modeling_system_dia()
```

Value

Invisible NULL. Initializes the internal model registry.

Examples

```
# Initialize the system (typically run once at the start of a session or script)
initialize_modeling_system_dia()

# Check if a default model like Random Forest is now registered
"rf" %in% names(get_registered_models_dia())
```

initialize_modeling_system_pro *Initialize Prognosis Modeling System*

Description

Initializes the environment and registers default survival models (Lasso, Elastic Net, Ridge, RSF, StepCox, GBM, XGBoost, PLS).

Usage

```
initialize_modeling_system_pro()
```

int_dia *Comprehensive Diagnostic Modeling Pipeline*

Description

Executes a complete diagnostic modeling workflow including single models, bagging, stacking, and voting ensembles across training and multiple test datasets. Returns structured results with AUROC values for visualization.

Usage

```
int_dia(
  ...,
  model_names = NULL,
  tune = TRUE,
  n_estimators = 10,
  seed = 123,
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Positive",
  new_negative_label = "Negative"
)
```

Arguments

...	Data frames for analysis. The first is the training dataset; all subsequent arguments are test datasets.
model_names	Character vector specifying which models to use. If NULL (default), uses all registered models.
tune	Logical, enable hyperparameter tuning. Default TRUE.
n_estimators	Integer, number of bootstrap samples for bagging. Default 10.

seed Integer for reproducibility. Default 123.

positive_label_value
Value representing positive class. Default 1.

negative_label_value
Value representing negative class. Default 0.

new_positive_label
Factor level name for positive class. Default "Positive".

new_negative_label
Factor level name for negative class. Default "Negative".

Value

A list containing all_results, auroc_matrix, model_categories, dataset_names.

---

int_imbalance *Imbalanced Data Diagnostic Modeling Pipeline*

Description

Extends int_dia by adding imbalance-specific models (EasyEnsemble). Produces a comprehensive set of models optimized for imbalanced datasets.

Usage

```
int_imbalance(
  ...,
  model_names = NULL,
  tune = TRUE,
  n_estimators = 10,
  seed = 123,
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Positive",
  new_negative_label = "Negative"
)
```

Arguments

...	Data frames for analysis. The first is the training dataset; all subsequent arguments are test datasets.
model_names	Character vector specifying which models to use. If NULL (default), uses all registered models.
tune	Logical, enable hyperparameter tuning. Default TRUE.
n_estimators	Integer, number of bootstrap samples for bagging. Default 10.
seed	Integer for reproducibility. Default 123.

positive_label_value
: Value representing positive class. Default 1.

negative_label_value
: Value representing negative class. Default 0.

new_positive_label
: Factor level name for positive class. Default "Positive".

new_negative_label
: Factor level name for negative class. Default "Negative".

Value

Same structure as `int_dia` with additional imbalance-handling models.

Examples

```
## Not run:
imbalanced_results <- int_imbalance(train_imbalanced, test_imbalanced)

## End(Not run)
```

---

int_pro *Comprehensive Prognostic Modeling Pipeline*

Description

Executes a complete prognostic (survival) modeling workflow including single models, bagging, and stacking ensembles. Returns C-index and time-dependent AUROC metrics.

Usage

```
int_pro(
  ...,
  model_names = NULL,
  tune = TRUE,
  n_estimators = 10,
  seed = 123,
  time_unit = "day",
  years_to_evaluate = c(1, 3, 5)
)
```

Arguments

`...`	Data frames for survival analysis. First = training; others = test sets. Format: first column = ID, second = outcome (0/1), third = time, remaining = features.
`model_names`	Character vector specifying which models to use. If NULL (default), uses all registered prognostic models.

tune	Logical, enable tuning. Default TRUE.
n_estimators	Integer, bagging iterations. Default 10.
seed	Integer for reproducibility. Default 123.
time_unit	Time unit in data: "day", "month", or "year". Default "day".
years_to_evaluate	Numeric vector of years for time-dependent AUROC. Default c(1,3,5).

Value

A list with:

- all_results: All model outputs
- cindex_matrix: C-index values (models × datasets)
- avg_auroc_matrix: Average time-dependent AUROC (models × datasets)
- model_categories: Model category labels
- dataset_names: Dataset identifiers

Examples

```
## Not run:
prognosis_results <- int_pro(train_pro, test_pro1, test_pro2)

## End(Not run)
```

---

lasso_dia *Train a Lasso (L1 Regularized Logistic Regression) Model for Classification*

Description

Trains a Lasso-regularized logistic regression model using caret::train (via glmnet method) for binary classification.

Usage

```
lasso_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

X	A data frame of features.
y	A factor vector of class labels.
tune	Logical, whether to perform hyperparameter tuning for lambda (if TRUE) or use a fixed value (if FALSE). alpha is fixed at 1 for Lasso.
cv_folds	An integer, the number of cross-validation folds for caret.

Value

A `caret::train` object representing the trained Lasso model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
lasso_model <- lasso_dia(X_toy, y_toy)
print(lasso_model)
```

---

lasso_pro *Train Lasso Cox Proportional Hazards Model*

Description

Fits a Cox proportional hazards model regularized by the Lasso (L1) penalty. Uses cross-validation to select the optimal lambda.

Usage

```
lasso_pro(X, y_surv, tune = FALSE)
```

Arguments

`X`	A data frame of predictors.
`y_surv`	A `Surv` object containing time and status.
`tune`	Logical. If TRUE, performs internal tuning (currently handled by cv.glmnet automatically).

Value

An object of class `survival_glmnet` and `pro_model`.

Examples

```
library(survival)
# Create dummy data
set.seed(123)
df <- data.frame(time = rexp(50), status = sample(0:1, 50, replace=TRUE),
                 var1 = rnorm(50), var2 = rnorm(50))
y <- Surv(df$time, df$status)
x <- df[, c("var1", "var2")]

model <- lasso_pro(x, y)
print(class(model))
```

`lda_dia` *Train a Linear Discriminant Analysis (LDA) Model for Classification*

Description

Trains a Linear Discriminant Analysis (LDA) model using `caret::train` for binary classification.

Usage

```
lda_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

`X`	A data frame of features.
`y`	A factor vector of class labels.
`tune`	Logical, whether to perform hyperparameter tuning (currently ignored for LDA).
`cv_folds`	An integer, the number of cross-validation folds for `caret`.

Value

A `caret::train` object representing the trained LDA model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
lda_model <- lda_dia(X_toy, y_toy)
print(lda_model)
```

load_and_prepare_data_dia *Load and Prepare Data for Diagnostic Models*

Description

Loads a CSV file containing patient data, extracts features, and converts the label column into a factor suitable for classification models. Handles basic data cleaning like trimming whitespace and type conversion.

Usage

```
load_and_prepare_data_dia(
  data_path,
  label_col_name,
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Positive",
  new_negative_label = "Negative"
)
```

Arguments

data_path	A character string, the file path to the input CSV data. The first column is assumed to be a sample ID.
label_col_name	A character string, the name of the column containing the class labels.
positive_label_value	A numeric or character value that represents the positive class in the raw data.
negative_label_value	A numeric or character value that represents the negative class in the raw data.
new_positive_label	A character string, the desired factor level name for the positive class (e.g., "Positive").
new_negative_label	A character string, the desired factor level name for the negative class (e.g., "Negative").

Value

A list containing:

- X: A data frame of features (all columns except ID and label).
- y: A factor vector of class labels, with levels new_negative_label and new_positive_label.
- sample_ids: A vector of sample IDs (the first column of the input data).
- pos_class_label: The character string used for the positive class factor level.
- neg_class_label: The character string used for the negative class factor level.
- y_original_numeric: The original numeric/character vector of labels.

Examples

```
# Create a dummy CSV file in a temporary directory for demonstration
temp_csv_path <- tempfile(fileext = ".csv")
dummy_data <- data.frame(
  ID = paste0("Patient", 1:50),
  Disease_Status = sample(c(0, 1), 50, replace = TRUE),
  FeatureA = rnorm(50),
  FeatureB = runif(50, 0, 100),
  CategoricalFeature = sample(c("X", "Y", "Z"), 50, replace = TRUE)
)
write.csv(dummy_data, temp_csv_path, row.names = FALSE)

# Load and prepare data from the temporary file
prepared_data <- load_and_prepare_data_dia(
  data_path = temp_csv_path,
  label_col_name = "Disease_Status",
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Case",
  new_negative_label = "Control"
)

# Check prepared data structure
str(prepared_data$X)
table(prepared_data$y)

# Clean up the dummy file
unlink(temp_csv_path)
```

---

min_max_normalize *Min-Max Normalization*

Description

Performs linear transformation of data to the range 0 to 1. Essential for stacking ensembles to normalize risk scores from heterogeneous base learners.

Usage

```
min_max_normalize(x, min_val = NULL, max_val = NULL)
```

Arguments

x	A numeric vector.
min_val	Optional reference minimum value (e.g., from training set).
max_val	Optional reference maximum value (e.g., from training set).

Value

A numeric vector of normalized values.

`mlp_dia` *Train a Multi-Layer Perceptron (Neural Network) Model for Classification*

Description

Trains a Multi-Layer Perceptron (MLP) neural network model using `caret::train` for binary classification.

Usage

```
mlp_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

`X`	A data frame of features.
`y`	A factor vector of class labels.
`tune`	Logical, whether to perform hyperparameter tuning using `caret`'s default grid (if `TRUE`) or a fixed value (if `FALSE`).
`cv_folds`	An integer, the number of cross-validation folds for `caret`.

Value

A `caret::train` object representing the trained MLP model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
mlp_model <- mlp_dia(X_toy, y_toy)
print(mlp_model)
```

`models_dia` *Run Multiple Diagnostic Models*

Description

Trains and evaluates one or more registered diagnostic models on a given dataset.

Usage

```
models_dia(
  data,
  model = "all_dia",
  tune = FALSE,
  seed = 123,
  threshold_choices = "default",
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Positive",
  new_negative_label = "Negative"
)
```

Arguments

`data`	A data frame where the first column is the sample ID, the second is the outcome label, and subsequent columns are features.
`model`	A character string or vector of character strings, specifying which models to run. Use "all_dia" to run all registered models.
`tune`	Logical, whether to enable hyperparameter tuning for individual models.
`seed`	An integer, for reproducibility of random processes.
`threshold_choices`	A character string (e.g., "f1", "youden", "default") or a numeric value (0-1), or a named list/vector allowing different threshold strategies/values for each model.
`positive_label_value`	A numeric or character value in the raw data representing the positive class.
`negative_label_value`	A numeric or character value in the raw data representing the negative class.
`new_positive_label`	A character string, the desired factor level name for the positive class (e.g., "Positive").
`new_negative_label`	A character string, the desired factor level name for the negative class (e.g., "Negative").

Value

A named list, where each element corresponds to a run model and contains its trained `model_object`, `sample_score` data frame, and `evaluation_metrics`.

See Also

initialize_modeling_system_dia, evaluate_model_dia

Examples

```
# This example assumes your package includes a dataset named 'train_dia'.
# If not, you should create a toy data frame similar to the one below.
#
# train_dia <- data.frame(
#   ID = paste0("Patient", 1:100),
#   Disease_Status = sample(c(0, 1), 100, replace = TRUE),
#   FeatureA = rnorm(100),
#   FeatureB = runif(100)
# )

# Ensure the 'train_dia' dataset is available in the environment
# For example, if it is exported by your package:
# data(train_dia)

# Check if 'train_dia' exists, otherwise skip the example
if (exists("train_dia")) {
  # 1. Initialize the modeling system
  initialize_modeling_system_dia()

  # 2. Run selected models
  results <- models_dia(
    data = train_dia,
    model = c("rf", "lasso"), # Run only Random Forest and Lasso
    threshold_choices = list(rf = "f1", lasso = 0.6), # Different thresholds
    positive_label_value = 1,
    negative_label_value = 0,
    new_positive_label = "Case",
    new_negative_label = "Control",
    seed = 42
  )

  # 3. Print summaries
  for (model_name in names(results)) {
    print_model_summary_dia(model_name, results[[model_name]])
  }
}
```

`models_pro` *Run Multiple Prognostic Models*

Description

High-level API to train and evaluate multiple survival models in batch.

Usage

```
models_pro(
  data,
  model = "all_pro",
  tune = FALSE,
  seed = 123,
  time_unit = "day",
  years_to_evaluate = c(1, 3, 5)
)
```

Arguments

data	Input data frame.
model	Character vector of model names or "all_pro".
tune	Logical. Enable hyperparameter tuning?
seed	Random seed.
time_unit	Time unit of input.
years_to_evaluate	Years for AUC calculation.

Value

A list of model results.

---

nb_dia *Train a Naive Bayes Model for Classification*

Description

Trains a Naive Bayes model using `caret::train` for binary classification.

Usage

```
nb_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

X	A data frame of features.
y	A factor vector of class labels.
tune	Logical, whether to perform hyperparameter tuning using `caret`'s default grid (if `TRUE`) or fixed values (if `FALSE`).
cv_folds	An integer, the number of cross-validation folds for `caret`.

Value

A `caret::train` object representing the trained Naive Bayes model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
nb_model <- nb_dia(X_toy, y_toy)
print(nb_model)
```

`plot_integrated_results` *Visualize Integrated Modeling Results*

Description

Creates a heatmap visualization with performance metrics across models and datasets, including category annotations and summary bar plots.

Usage

```
plot_integrated_results(results_obj, metric_name = "AUROC", output_file = NULL)
```

Arguments

`results_obj`	Output from `int_dia`, `int_imbalance`, or `int_pro`.
`metric_name`	Character string for metric used (e.g., "AUROC", "C-index").
`output_file`	Optional file path to save plot. If NULL, plot is displayed.

Value

A ggplot object (invisibly).

Examples

```
## Not run:
results <- int_dia(train_dia, test_dia)
plot_integrated_results(results, "AUROC")

## End(Not run)
```

pls_pro *Train Partial Least Squares Cox (PLS-Cox)*

Description

Fits a Cox model using Partial Least Squares reduction for high-dimensional data.

Usage

```
pls_pro(X, y_surv, tune = FALSE)
```

Arguments

X	A data frame of predictors.
y_surv	A Surv object containing time and status.
tune	Logical. If TRUE, performs internal tuning (currently handled by cv.glmnet automatically).

Value

An object of class survival_plsRcox and pro_model.

predict_pro *Generic Prediction Interface for Prognostic Models*

Description

A unified S3 generic method to generate prognostic risk scores from various trained model objects. This decouples the prediction implementation from the high-level evaluation logic, facilitating extensibility.

Usage

```
predict_pro(object, newdata, ...)
```

Arguments

object	A trained model object with class pro_model.
newdata	A data frame containing features for prediction.
...	Additional arguments passed to specific methods.

Value

A numeric vector representing the prognostic risk score (higher values typically indicate higher risk).

print_model_summary_dia *Print Diagnostic Model Summary*

Description

Prints a formatted summary of the evaluation metrics for a diagnostic model, either from training data or new data evaluation.

Usage

```
print_model_summary_dia(model_name, results_list, on_new_data = FALSE)
```

Arguments

model_name	A character string, the name of the model (e.g., "rf", "Bagging (RF)").
results_list	A list containing model evaluation results, typically an element from the output of models_dia() or the result of bagging_dia(), stacking_dia(), voting_dia(), or imbalance_dia(). It must contain evaluation_metrics and model_object (if applicable).
on_new_data	Logical, indicating whether the results are from applying the model to new, unseen data (TRUE) or from the training/internal validation data (FALSE).

Value

NULL. Prints the summary to the console.

Examples

```
# Example for a successfully evaluated model
successful_results <- list(
  evaluation_metrics = list(
    Threshold_Strategy = "f1",
    `_Threshold` = 0.45,
    AUROC = 0.85, AUROC_95CI_Lower = 0.75, AUROC_95CI_Upper = 0.95,
    AUPRC = 0.80, Accuracy = 0.82, F1 = 0.78,
    Precision = 0.79, Recall = 0.77, Specificity = 0.85
  )
)
print_model_summary_dia("MyAwesomeModel", successful_results)

# Example for a failed model
failed_results <- list(evaluation_metrics = list(error = "Training failed"))
print_model_summary_dia("MyFailedModel", failed_results)
```

print_model_summary_pro *Print Prognostic Model Summary*

Description

Formatted console output of model performance.

Usage

```
print_model_summary_pro(model_name, results_list)
```

Arguments

model_name	Name of the model.
results_list	Result object containing evaluation_metrics.

qda_dia *Train a Quadratic Discriminant Analysis (QDA) Model for Classification*

Description

Trains a Quadratic Discriminant Analysis (QDA) model using caret::train for binary classification.

Usage

```
qda_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

X	A data frame of features.
y	A factor vector of class labels.
tune	Logical, whether to perform hyperparameter tuning (currently ignored for QDA).
cv_folds	An integer, the number of cross-validation folds for caret.

Value

A caret::train object representing the trained QDA model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
qda_model <- qda_dia(X_toy, y_toy)
print(qda_model)
```

`register_model_dia` *Register a Diagnostic Model Function*

Description

Registers a user-defined or pre-defined diagnostic model function with the internal model registry. This allows the function to be called later by its registered name, facilitating a modular model management system.

Usage

```
register_model_dia(name, func)
```

Arguments

`name`	A character string, the unique name to register the model under.
`func`	A function, the R function implementing the diagnostic model. This function should typically accept `X` (features) and `y` (labels) as its first two arguments and return a `caret::train` object.

Value

NULL. The function registers the model function invisibly.

See Also

`get_registered_models_dia`, `initialize_modeling_system_dia`

Examples

```
# Example of a dummy model function for registration
my_dummy_rf_model <- function(X, y, tune = FALSE, cv_folds = 5) {
  message("Training dummy RF model...")
  # This is a placeholder and doesn't train a real model.
  # It returns a list with a structure similar to a caret train object.
  list(method = "dummy_rf")
}

# Initialize the system before registering
initialize_modeling_system_dia()

# Register the new model
register_model_dia("dummy_rf", my_dummy_rf_model)

# Verify that the model is now in the list of registered models
"dummy_rf" %in% names(get_registered_models_dia())
```

---

register_model_pro *Register a Prognostic Model*

Description

Registers a model function into the internal system environment, making it available for batch execution.

Usage

```
register_model_pro(name, func)
```

Arguments

name	String identifier for the model.
func	The model training function.

---

rf_dia *Train a Random Forest Model for Classification*

Description

Trains a Random Forest model using `caret::train` for binary classification.

Usage

```
rf_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

X	A data frame of features.
y	A factor vector of class labels.
tune	Logical, whether to perform hyperparameter tuning using caret's default grid (if TRUE) or use a fixed mtry value (if FALSE).
cv_folds	An integer, the number of cross-validation folds for caret.

Value

A caret::train object representing the trained Random Forest model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
rf_model <- rf_dia(X_toy, y_toy)
print(rf_model)
```

---

ridge_dia *Train a Ridge (L2 Regularized Logistic Regression) Model for Classification*

Description

Trains a Ridge-regularized logistic regression model using caret::train (via glmnet method) for binary classification.

Usage

```
ridge_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

X	A data frame of features.
y	A factor vector of class labels.
tune	Logical, whether to perform hyperparameter tuning for lambda (if TRUE) or use a fixed value (if FALSE). alpha is fixed at 0 for Ridge.
cv_folds	An integer, the number of cross-validation folds for caret.

Value

A `caret::train` object representing the trained Ridge model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
ridge_model <- ridge_dia(X_toy, y_toy)
print(ridge_model)
```

---

ridge_pro *Train Ridge Cox Model*

Description

Fits a Cox model with Ridge (L2) regularization.

Usage

```
ridge_pro(X, y_surv, tune = FALSE)
```

Arguments

`X`	A data frame of predictors.
`y_surv`	A `Surv` object containing time and status.
`tune`	Logical. If TRUE, performs internal tuning (currently handled by cv.glmnet automatically).

Value

An object of class `survival_glmnet` and `pro_model`.

rsf_pro *Train Random Survival Forest (RSF)*

Description

Fits a Random Survival Forest using the log-rank splitting rule. Includes capabilities for hyperparameter tuning via grid search over `ntree`, `nodesize`, and `mtry`.

Usage

```
rsf_pro(X, y_surv, tune = FALSE, tune_params = NULL)
```

Arguments

`X`	A data frame of predictors.
`y_surv`	A `Surv` object containing time and status.
`tune`	Logical. If TRUE, performs grid search for optimal hyperparameters based on C-index.
`tune_params`	Optional data frame containing the grid for tuning.

Value

An object of class `survival_rsf` and `pro_model`.

stacking_dia *Train a Stacking Diagnostic Model*

Description

Implements a Stacking ensemble. It trains multiple base models, then uses their predictions as features to train a meta-model.

Usage

```
stacking_dia(
  results_all_models,
  data,
  meta_model_name,
  top = 5,
  tune_meta = FALSE,
  threshold_choices = "f1",
  seed = 789,
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Positive",
  new_negative_label = "Negative"
)
```

Arguments

results_all_models	A list of results from models_dia(), containing trained base model objects and their evaluation metrics.
data	A data frame where the first column is the sample ID, the second is the outcome label, and subsequent columns are features. Used for training the meta-model.
meta_model_name	A character string, the name of the meta-model to use (e.g., "lasso", "gbm"). This model must be registered.
top	An integer, the number of top-performing base models (ranked by AUROC) to select for the stacking ensemble.
tune_meta	Logical, whether to enable tuning for the meta-model.
threshold_choices	A character string (e.g., "f1", "youden", "default") or a numeric value (0-1) for determining the evaluation threshold for the ensemble.
seed	An integer, for reproducibility.
positive_label_value	A numeric or character value in the raw data representing the positive class.
negative_label_value	A numeric or character value in the raw data representing the negative class.
new_positive_label	A character string, the desired factor level name for the positive class (e.g., "Positive").
new_negative_label	A character string, the desired factor level name for the negative class (e.g., "Negative").

Value

A list containing the model_object, sample_score, and evaluation_metrics.

See Also

models_dia, evaluate_model_dia

Examples

```
# 1. Initialize the modeling system
initialize_modeling_system_dia()

# 2. Create a toy dataset for demonstration
set.seed(42)
data_toy <- data.frame(
  ID = paste0("Sample", 1:60),
  Status = sample(c(0, 1), 60, replace = TRUE),
  Feat1 = rnorm(60),
  Feat2 = runif(60)
```

```
)

# 3. Generate mock base model results (as if from models_dia)
# In a real scenario, you would run models_dia() on your full dataset
base_model_results <- models_dia(
  data = data_toy,
  model = c("rf", "lasso"),
  seed = 123
)

# 4. Run the stacking ensemble
stacking_results <- stacking_dia(
  results_all_models = base_model_results,
  data = data_toy,
  meta_model_name = "gbm",
  top = 2,
  threshold_choices = "f1"
)
print_model_summary_dia("Stacking (GBM)", stacking_results)
```

`stacking_pro` *Train Stacking Ensemble for Prognosis*

Description

Implements a Stacking Ensemble (Super Learner). It uses the risk scores from top-performing base models as meta-features to train a second-level meta-learner.

Usage

```
stacking_pro(
  results_all_models,
  data,
  meta_model_name,
  top = 3,
  tune_meta = FALSE,
  time_unit = "day",
  years_to_evaluate = c(1, 3, 5),
  seed = 789
)
```

Arguments

`results_all_models`	List of results from `models_pro()`.
`data`	Training data.
`meta_model_name`	Name of the meta-learner (e.g., "lasso_pro").

top	Integer. Number of top base models to include based on C-index.
tune_meta	Logical. Tune the meta-learner?
time_unit	Time unit.
years_to_evaluate	Evaluation years.
seed	Integer seed.

Value

A list containing the stacking object and evaluation results.

`stepcox_pro` *Train Stepwise Cox Model (AIC-based)*

Description

Fits a Cox model and performs backward stepwise selection based on AIC.

Usage

```
stepcox_pro(X, y_surv, tune = FALSE)
```

Arguments

X	A data frame of predictors.
y_surv	A Surv object containing time and status.
tune	Logical. If TRUE, performs internal tuning (currently handled by cv.glmnet automatically).

Value

An object of class survival_stepcox and pro_model.

`Surv` *re-export Surv from survival*

Description

re-export Surv from survival

svm_dia *Train a Support Vector Machine (Linear Kernel) Model for Classification*

Description

Trains a Support Vector Machine (SVM) model with a linear kernel using `caret::train` for binary classification.

Usage

```
svm_dia(X, y, tune = FALSE, cv_folds = 5)
```

Arguments

X	A data frame of features.
y	A factor vector of class labels.
tune	Logical, whether to perform hyperparameter tuning using `caret`'s default grid (if `TRUE`) or a fixed value (if `FALSE`).
cv_folds	An integer, the number of cross-validation folds for `caret`.

Value

A `caret::train` object representing the trained SVM model.

Examples

```
set.seed(42)
n_obs <- 50
X_toy <- data.frame(
  FeatureA = rnorm(n_obs),
  FeatureB = runif(n_obs, 0, 100)
)
y_toy <- factor(sample(c("Control", "Case"), n_obs, replace = TRUE),
                levels = c("Control", "Case"))

# Train the model
svm_model <- svm_dia(X_toy, y_toy)
print(svm_model)
```

`test_dia` *Test Data for Diagnostic Models*

Description

A test dataset for evaluating diagnostic models, with a structure identical to `train_dia`.

Usage

```
test_dia
```

Format

A data frame with rows for samples and 22 columns:

sample character. Unique identifier for each sample.

outcome integer. The binary outcome (0 or 1).

AC004637.1 numeric. Gene expression level.

AC008459.1 numeric. Gene expression level.

AC009242.1 numeric. Gene expression level.

AC016735.1 numeric. Gene expression level.

AC090125.1 numeric. Gene expression level.

AC104237.3 numeric. Gene expression level.

AC112721.2 numeric. Gene expression level.

AC246817.1 numeric. Gene expression level.

AL135841.1 numeric. Gene expression level.

AL139241.1 numeric. Gene expression level.

HYMAI numeric. Gene expression level.

KCNIP2.AS1 numeric. Gene expression level.

LINC00639 numeric. Gene expression level.

LINC00922 numeric. Gene expression level.

LINC00924 numeric. Gene expression level.

LINC00958 numeric. Gene expression level.

LINC01028 numeric. Gene expression level.

LINC01614 numeric. Gene expression level.

LINC01644 numeric. Gene expression level.

PRDM16.DT numeric. Gene expression level.

Source

Stored in `data/test_dia.rda`.

`test_pro` *Test Data for Prognostic (Survival) Models*

Description

A test dataset for evaluating prognostic models, with a structure identical to `train_pro`.

Usage

```
test_pro
```

Format

A data frame with rows for samples and 31 columns:

sample character. Unique identifier for each sample.

outcome integer. The event status (0 or 1).

time numeric. The time to event or censoring.

AC004990.1 numeric. Gene expression level.

AC055854.1 numeric. Gene expression level.

AC084212.1 numeric. Gene expression level.

AC092118.1 numeric. Gene expression level.

AC093515.1 numeric. Gene expression level.

AC104211.1 numeric. Gene expression level.

AC105046.1 numeric. Gene expression level.

AC105219.1 numeric. Gene expression level.

AC110772.2 numeric. Gene expression level.

AC133644.1 numeric. Gene expression level.

AL133467.1 numeric. Gene expression level.

AL391845.2 numeric. Gene expression level.

AL590434.1 numeric. Gene expression level.

AL603840.1 numeric. Gene expression level.

AP000851.2 numeric. Gene expression level.

AP001434.1 numeric. Gene expression level.

C9orf163 numeric. Gene expression level.

FAM153CP numeric. Gene expression level.

HOTAIR numeric. Gene expression level.

HYMAI numeric. Gene expression level.

LINC00165 numeric. Gene expression level.

LINC01028 numeric. Gene expression level.

LINC01152 numeric. Gene expression level.

LINC01497 numeric. Gene expression level.

LINC01614 numeric. Gene expression level.

LINC01929 numeric. Gene expression level.

LINC02408 numeric. Gene expression level.

SIRLNT numeric. Gene expression level.

Source

Stored in `data/test_pro.rda`.

`train_dia` *Training Data for Diagnostic Models*

Description

A training dataset for diagnostic models, containing sample IDs, binary outcomes, and gene expression features.

Usage

```
train_dia
```

Format

A data frame with rows for samples and 22 columns:

sample character. Unique identifier for each sample.

outcome integer. The binary outcome, where 1 typically represents a positive case and 0 a negative case.

AC004637.1 numeric. Gene expression level.

AC008459.1 numeric. Gene expression level.

AC009242.1 numeric. Gene expression level.

AC016735.1 numeric. Gene expression level.

AC090125.1 numeric. Gene expression level.

AC104237.3 numeric. Gene expression level.

AC112721.2 numeric. Gene expression level.

AC246817.1 numeric. Gene expression level.

AL135841.1 numeric. Gene expression level.

AL139241.1 numeric. Gene expression level.

HYMAI numeric. Gene expression level.

KCNIP2.AS1 numeric. Gene expression level.

LINC00639 numeric. Gene expression level.

LINC00922 numeric. Gene expression level.

LINC00924 numeric. Gene expression level.

LINC00958 numeric. Gene expression level.

LINC01028 numeric. Gene expression level.

LINC01614 numeric. Gene expression level.

LINC01644 numeric. Gene expression level.

PRDM16.DT numeric. Gene expression level.

Details

This dataset is used to train machine learning models for diagnosis. The column names starting with 'AC', 'AL', 'LINC', etc., are feature variables.

Source

Stored in `data/train_dia.rda`.

`train_pro` *Training Data for Prognostic (Survival) Models*

Description

A training dataset for prognostic models, containing sample IDs, survival outcomes (time and event status), and gene expression features.

Usage

```
train_pro
```

Format

A data frame with rows for samples and 31 columns:

sample character. Unique identifier for each sample.

outcome integer. The event status, where 1 indicates an event occurred and 0 indicates censoring.

time numeric. The time to event or censoring.

AC004990.1 numeric. Gene expression level.

AC055854.1 numeric. Gene expression level.

AC084212.1 numeric. Gene expression level.

AC092118.1 numeric. Gene expression level.

AC093515.1 numeric. Gene expression level.

AC104211.1 numeric. Gene expression level.

AC105046.1 numeric. Gene expression level.

AC105219.1 numeric. Gene expression level.

AC110772.2 numeric. Gene expression level.

AC133644.1 numeric. Gene expression level.

AL133467.1 numeric. Gene expression level.

AL391845.2 numeric. Gene expression level.

AL590434.1 numeric. Gene expression level.

AL603840.1 numeric. Gene expression level.

AP000851.2 numeric. Gene expression level.

AP001434.1 numeric. Gene expression level.

C9orf163 numeric. Gene expression level.

FAM153CP numeric. Gene expression level.

HOTAIR numeric. Gene expression level.

HYMAI numeric. Gene expression level.

LINC00165 numeric. Gene expression level.

LINC01028 numeric. Gene expression level.

LINC01152 numeric. Gene expression level.

LINC01497 numeric. Gene expression level.

LINC01614 numeric. Gene expression level.

LINC01929 numeric. Gene expression level.

LINC02408 numeric. Gene expression level.

SIRLNT numeric. Gene expression level.

Details

This dataset is used to train machine learning models for prognosis. The features are typically gene expression values.

Source

Stored in `data/train_pro.rda`.

voting_dia *Train a Voting Ensemble Diagnostic Model*

Description

Implements a Voting ensemble, combining predictions from multiple base models through soft or hard voting.

Usage

```
voting_dia(
  results_all_models,
  data,
  type = c("soft", "hard"),
  weight_metric = "AUROC",
  top = 5,
  seed = 789,
  threshold_choices = "f1",
  positive_label_value = 1,
  negative_label_value = 0,
  new_positive_label = "Positive",
  new_negative_label = "Negative"
)
```

Arguments

results_all_models	A list of results from models_dia(), containing trained base model objects and their evaluation metrics.
data	A data frame where the first column is the sample ID, the second is the outcome label, and subsequent columns are features. Used for evaluation.
type	A character string, "soft" for weighted average of probabilities or "hard" for majority class voting.
weight_metric	A character string, the metric to use for weighting base models in soft voting (e.g., "AUROC", "F1"). Ignored for hard voting.
top	An integer, the number of top-performing base models (ranked by weight_metric) to include in the ensemble.
seed	An integer, for reproducibility.
threshold_choices	A character string (e.g., "f1", "youden", "default") or a numeric value (0-1) for determining the evaluation threshold for the ensemble.
positive_label_value	A numeric or character value in the raw data representing the positive class.
negative_label_value	A numeric or character value in the raw data representing the negative class.

`new_positive_label`
: A character string, the desired factor level name for the positive class (e.g., "Positive").

`new_negative_label`
: A character string, the desired factor level name for the negative class (e.g., "Negative").

Value

A list containing the `model_object`, `sample_score`, and `evaluation_metrics`.

See Also

`models_dia`, `evaluate_model_dia`

Examples

```
# 1. Initialize the modeling system
initialize_modeling_system_dia()

# 2. Create a toy dataset for demonstration
set.seed(42)
data_toy <- data.frame(
  ID = paste0("Sample", 1:60),
  Status = sample(c(0, 1), 60, replace = TRUE),
  Feat1 = rnorm(60),
  Feat2 = runif(60)
)

# 3. Generate mock base model results (as if from models_dia)
base_model_results <- models_dia(
  data = data_toy,
  model = c("rf", "lasso"),
  seed = 123
)

# 4. Run the soft voting ensemble
soft_voting_results <- voting_dia(
  results_all_models = base_model_results,
  data = data_toy,
  type = "soft",
  weight_metric = "AUROC",
  top = 2,
  threshold_choices = "f1"
)
print_model_summary_dia("Soft Voting", soft_voting_results)
```

xb_dia *Train an XGBoost Tree Model for Classification*

Description

Trains an Extreme Gradient Boosting (XGBoost) model using `caret::train` for binary classification.

Usage

```
xb_dia(X, y, tune = FALSE, cv_folds = 5, tune_length = 20)
```

Arguments

`X`	A data frame of features.
`y`	A factor vector of class labels.
`tune`	Logical, whether to perform hyperparameter tuning using `caret`'s default grid (if `TRUE`) or use fixed values (if `FALSE`).
`cv_folds`	An integer, the number of cross-validation folds for `caret`.
`tune_length`	An integer, the number of random parameter combinations to try when tune=TRUE. Only used when search="random". Default is 20.

Value

A `caret::train` object representing the trained XGBoost model.

xgb_pro *Train XGBoost Cox Model*

Description

Fits an XGBoost model using the Cox proportional hazards objective function.

Usage

```
xgb_pro(X, y_surv, tune = FALSE)
```

Arguments

`X`	A data frame of predictors.
`y_surv`	A `Surv` object containing time and status.
`tune`	Logical. If TRUE, performs internal tuning (currently handled by cv.glmnet automatically).

Value

An object of class `survival_xgboost` and `pro_model`.

Index