Package: IntegratedLearner (via r-universe)

November 25, 2026

Type Package

Title Integrated Multi-Omics Learning for Survival and Other Outcomes

Version 1.0.0

Description Provides a unified interface for multi-omics prediction using early and late fusion for continuous, binary, multiclass, and survival outcomes. It supports both MultiAssayExperiment and PCL-style inputs, performs input validation and feature/sample harmonization across layers, and provides model fitting, prediction, plotting, and variable-importance utilities.

License MIT + file LICENSE

URL https://github.com/himelmallick/IntegratedLearner

BugReports https://github.com/himelmallick/IntegratedLearner/issues

Encoding UTF-8

Depends R (>= 4.5.0)

Imports BART, CoxBoost, bayesplot, bayesreg, bartMachine, stats, utils, cowplot, ggplot2, gbm, glmnetUtils, mboost, nnet, nloptr, progressr, quadprog, randomForest, randomForestSRC, timeROC, SuperLearner, survival, glmnet, ranger, SummarizedExperiment, MultiAssayExperiment, caret, ROCR, dplyr, tibble, tidyr, withr, multiview, xgboost

Suggests BiocStyle, mlbench, prediction, S4Vectors, testthat (>= 3.0.0), knitr, rmarkdown

SystemRequirements Java (optional; required for sl_bart / bartMachine workflows)

VignetteBuilder knitr

biocViews Software, Classification, Survival, Microbiome

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

Config/pak/sysreqs
cmake libglpk-dev make default-jdk libicu-dev libuv1-dev libxml2-dev libx11-dev zlib1g-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:57:35 UTC

RemoteUrl https://github.com/bioc/IntegratedLearner

RemoteRef RELEASE_3_24

RemoteSha 2b6444f3a05bd3aa9c9eb013dc83d4d7ed701e1f

Contents

auc_obj 2
IL_conbin 3
IL_multiclass 7
ILsurv 10
IntegratedLearner 14
NNLS 18
plot.learner 19
predict.learner 20
predict.sl_bart 22
predict.sl_nnls_auc 23
sl_bart 24
sl_enet 26
sl_glmnet2 28
sl_horseshoe 29
sl_lasso 31
sl_nnls_auc 32
update.learner 33

Index **36**

`auc_obj` *Title Meta level Objective function: NNLS for gaussian; Rank loss for binary observations*

Description

Title Meta level Objective function: NNLS for gaussian; Rank loss for binary observations

Usage

```
auc_obj(b, X, Y)
```

Arguments

b	Weights vector
X	Design matrix (data frame)
Y	Outcome variable

Value

1 - AUC

Examples

```
X <- data.frame(m1 = c(0.1, 0.4, 0.8, 0.9), m2 = c(0.2, 0.3, 0.7, 0.95))
Y <- c(0, 0, 1, 1)
auc_obj(b = c(0.5, 0.5), X = X, Y = Y)
```

IL_conbin *Integrated machine learning for multi-omics prediction (continuous/binary outcomes)*

Description

Performs integrated machine learning to predict a **binary or continuous** outcome based on two or more omics layers (views). This function implements the core IntegratedLearner engine for **non-survival outcomes**.

Usage

```
IL_conbin(
  feature_table,
  sample_metadata,
  feature_metadata,
  feature_table_valid = NULL,
  sample_metadata_valid = NULL,
  folds = 5,
  base_learner = "sl_bart",
  base_screener = "All",
  run_screening = FALSE,
  screen_pct = NULL,
  filter_method = NULL,
  filter_pct = NULL,
  prevalence_pct = NULL,
  meta_learner = "sl_nnls_auc",
  run_concat = TRUE,
  run_stacked = TRUE,
  run_intermediate = FALSE,
  cooperative_rho = c(0, 0.1, 0.25, 0.5, 1),
```

```
  cooperative_s = "lambda.min",
  cooperative_type_measure = NULL,
  drop_poor_performing_layers = FALSE,
  refit.stack = FALSE,
  family = stats::gaussian(),
  ...
)
```

Arguments

feature_table	An R data frame containing multi-layer features (in rows) and samples (in columns). Column names of feature_metadata must match the row names of sample_metadata.
sample_metadata	An R data frame of metadata variables (in columns). Must have a column named subjectID describing per-subject unique identifiers. For longitudinal designs, this variable may be non-unique. Additionally, a column named Y must be present which is the continuous or binary outcome of interest. Row names of sample_metadata must match the column names of feature_table.
feature_metadata	An R data frame of feature-specific metadata across views (in columns) and features (in rows). Must have a column named featureID as a unique per-feature identifier and a column named featureType describing the source layers. Row names of feature_metadata must match the row names of feature_table.
feature_table_valid	Feature table from validation set for which prediction is desired. Must have the exact same structure as feature_table. If missing, uses feature_table.
sample_metadata_valid	Sample-specific metadata table from independent validation set when available. Must have the exact same structure as sample_metadata.
folds	How many folds in the V-fold nested cross-validation? Default is 5.
base_learner	Base learner for late fusion and early fusion. Default: sl_bart.
base_screener	Deprecated for this backend; currently ignored and kept only for backward compatibility.
run_screening	Logical; if TRUE, run supervised screening within each training fold (and again on full training data) before fitting base models.
screen_pct	Percentage of features to retain during screening ((0,100]). Applied after any optional filtering.
filter_method	Optional feature-filter method before model fitting. Supported values are 'prevalence' (top detected features by prevalence) and 'variance' (top features by caret-based variance ranking via nearZeroVar + empirical variance). If NULL, defaults to 'prevalence' when filtering is requested.
filter_pct	Optional retention percentage (in (0,100]) for the selected filter_method. Keeps the top filter_pct percent features.
prevalence_pct	Optional retention percentage (in (0,100]) for prevalence-based filtering before model fitting. Deprecated alias of filter_pct with filter_method = 'prevalence'.

`meta_learner` Meta-learner for late fusion (stacked generalization). Defaults to `sl_nnls_auc`.

`run_concat` Should early fusion be run? Default is TRUE.

`run_stacked` Should stacked model (late fusion) be run? Default is TRUE.

`run_intermediate`
Should standalone cooperative learning via **multiview** be run? Default is FALSE. When TRUE, only the cooperative multiview model is fit; base learners, early fusion, and late fusion are skipped.

`cooperative_rho`
Non-negative agreement-penalty values to evaluate for cooperative learning.

`cooperative_s` Penalty value used when predicting from the selected cooperative model. Defaults to `"lambda.min"`.

`cooperative_type_measure`
Optional **multiview** CV loss/score. If `NULL`, an outcome-appropriate default is used.

`drop_poor_performing_layers`
Logical; if `TRUE`, layers with single-layer performance below the screening threshold are removed from early and late fusion only. The threshold is AUC < 0.5 for binary and $R^2 < 0.5$ for continuous outcomes. Single-layer outputs are still retained and reported.

`refit.stack` logical; For late fusion, refit predictions on the entire data are returned if specified. Default FALSE.

`family` Allows `gaussian()` for continuous and `binomial()` for binary outcomes. Survival outcomes must be handled via IL_survival().

`...` Additional arguments (currently unused).

Details

`IL_conbin` takes a training set (`feature_table, sample_metadata, feature_metadata`) and, optionally, a corresponding validation set, and returns predicted values based on the validation set. It also performs V-fold nested cross-validation to estimate the prediction accuracy of various fusion algorithms.

Two integration paradigms are supported: early and late. The software includes multiple ML models based on the `SuperLearner` R package as well as several data exploration capabilities and visualization modules in a unified estimation framework.

Although `IL_conbin()` is typically called internally by `IntegratedLearner()` after extracting multi-view feature tables from `MultiAssayExperiment` objects, advanced users may call `IL_conbin()` directly when they already have multi-layer feature tables and metadata in matrix/data.frame form.

Value

A list-like IntegratedLearner object containing fitted layer-specific, stacked, and concatenated models, cross-validated performance (AUC or R\302\262), and predictions for training and validation sets. When `run_intermediate = TRUE`, coefficient-based cooperative feature-importance scores from the selected **multiview** model are returned in `cooperative_feature_importance`, `cooperative_feature_importanc` `cooperative_feature_importance_signed`, and `cooperative_feature_importance_signed_by_layer`. Supported base learners also return native feature importance in `train_out$early$combined_importance`

and `train_out$late$stacked$feature_importance_by_layer` when the corresponding fusion method is enabled. The `sl_nnls_auc` stack additionally returns `layer_weights` and `combined_importance` under `train_out$late$stacked`.

Progress, summaries, and reproducibility

Call `set.seed()` before fitting to reproduce the data splits and stochastic fitting steps. To display progress, wrap the call in `progressr::with_progress()`. Use `print(fit)` to show a fitted model summary.

Author(s)

Himel Mallick, <him4004@med.cornell.edu>

See Also

IntegratedLearner, IL_survival()

Examples

```
set.seed(1)
n <- 20
feature_table <- as.data.frame(rbind(
  matrix(rnorm(3 * n), nrow = 3, dimnames = list(paste0("L1_F", 1:3), paste0("S", 1:n))),
  matrix(rnorm(2 * n), nrow = 2, dimnames = list(paste0("L2_F", 1:2), paste0("S", 1:n)))
))
sample_metadata <- data.frame(
  subjectID = paste0("ID", 1:n), Y = rnorm(n),
  row.names = colnames(feature_table)
)
feature_metadata <- data.frame(
  featureID = rownames(feature_table),
  featureType = c(rep("Layer1", 3), rep("Layer2", 2)),
  row.names = rownames(feature_table)
)
fit <- IL_conbin(
  feature_table = feature_table,
  sample_metadata = sample_metadata,
  feature_metadata = feature_metadata,
  folds = 3, base_learner = "SL.mean", run_stacked = FALSE,
  run_concat = FALSE, family = stats::gaussian()
)
names(fit)
```

IL_multiclass *Integrated machine learning for multi-omics multiclass classification*

Description

Native multiclass backend used by `IntegratedLearner()` when `family = binomial()` and the outcome has more than two classes. This backend preserves the existing API while using multiclass-native probability modeling and multiclass stacking.

Usage

```
IL_multiclass(
  feature_table,
  sample_metadata,
  feature_metadata,
  feature_table_valid = NULL,
  sample_metadata_valid = NULL,
  folds = 5,
  base_learner = "glmnet",
  base_screener = "All",
  run_screening = FALSE,
  screen_pct = NULL,
  filter_method = NULL,
  filter_pct = NULL,
  prevalence_pct = NULL,
  meta_learner = "glmnet",
  run_concat = TRUE,
  run_stacked = TRUE,
  run_intermediate = FALSE,
  cooperative_rho = c(0, 0.1, 0.25, 0.5, 1),
  cooperative_s = "lambda.min",
  cooperative_type_measure = NULL,
  family = stats::binomial(),
  eps = 1e-15,
  run_majority = TRUE,
  run_weighted = TRUE,
  ...
)
```

Arguments

`feature_table` An R data frame containing multi-layer features (in rows) and samples (in columns). Column names of `feature_metadata` must match the row names of `sample_metadata`.

`sample_metadata` An R data frame of metadata variables (in columns). Must have a column named `subjectID` describing per-subject unique identifiers. For longitudinal designs, this variable may be non-unique. Additionally, a column named `Y` must

be present which is the continuous or binary outcome of interest. Row names of `sample_metadata` must match the column names of `feature_table`.

`feature_metadata`
: An R data frame of feature-specific metadata across views (in columns) and features (in rows). Must have a column named `featureID` as a unique per-feature identifier and a column named `featureType` describing the source layers. Row names of `feature_metadata` must match the row names of `feature_table`.

`feature_table_valid`
: Feature table from validation set for which prediction is desired. Must have the exact same structure as `feature_table`. If missing, uses `feature_table`.

`sample_metadata_valid`
: Sample-specific metadata table from independent validation set when available. Must have the exact same structure as `sample_metadata`.

`folds` Number of subject-level folds for out-of-fold predictions. Default 5. Fully nested evaluation of the fusion pipeline is separate.

`base_learner` Native multiclass base learner. Default `"glmnet"`.

`base_screener` Deprecated for this backend; currently ignored and kept only for backward compatibility.

`run_screening` Logical; if `TRUE`, run supervised screening within each training fold (and again on full training data) before fitting base models.

`screen_pct` Percentage of features to retain during screening (`(0,100]`). Applied after any optional filtering.

`filter_method` Optional feature-filter method before model fitting. Supported values are `'prevalence'` (top detected features by prevalence) and `'variance'` (top features by caret-based variance ranking via `nearZeroVar` + empirical variance). If `NULL`, defaults to `'prevalence'` when filtering is requested.

`filter_pct` Optional retention percentage (in `(0,100]`) for the selected `filter_method`. Keeps the top `filter_pct` percent features.

`prevalence_pct` Optional retention percentage (in `(0,100]`) for prevalence-based filtering before model fitting. Deprecated alias of `filter_pct` with `filter_method = 'prevalence'`.

`meta_learner` Native multiclass stacking learner. Default `"glmnet"`. Legacy `sl_nnls_auc` and `SL.nnls` aliases map to multinomial glmnet, not scalar layer-weight optimization.

`run_concat` Should early fusion be run? Default is TRUE.

`run_stacked` Should stacked model (late fusion) be run? Default is TRUE.

`run_intermediate`
: Ignored for multiclass outcomes with a message; cooperative **multiview** learning is not currently supported for multiclass fits.

`cooperative_rho`
: Non-negative agreement-penalty values to evaluate for cooperative learning.

`cooperative_s` Penalty value used when predicting from the selected cooperative model. Defaults to `"lambda.min"`.

`cooperative_type_measure`
: Optional **multiview** CV loss/score. If `NULL`, an outcome-appropriate default is used.

`family`
: Allows `gaussian()` for continuous and `binomial()` for binary outcomes. Survival outcomes must be handled via IL_survival().

`eps`
: Small positive constant used to stabilize probabilities.

`run_majority`
: Logical; return multiclass majority voting. Default TRUE; ignored for other outcomes. Ties use the first class in training class order.

`run_weighted`
: Logical; return multiclass probability averaging with nonnegative layer weights summing to one, learned by minimizing out-of-fold multiclass log loss. Default TRUE; ignored for other outcomes.

`...`
: Additional arguments passed to native base-model fitting; these are not passed to the stacked meta-model or fusion-weight optimizer.

Details

Majority voting returns vote fractions under `majority`; these are not calibrated class probabilities. Ties use the first class in training class order. Learned averaging returns probabilities under `weighted_average` and stores final layer weights under `model_fits$model_weighted$weights`. Weights minimize multiclass log loss on out-of-fold layer probabilities. Training outputs cross-fit the weights across the existing folds. As with stacking, fully nested evaluation is needed for an unbiased assessment of the entire fitted pipeline. Results are also grouped under `train_out` and `valid_out`, each with `single`, `early`, and `late` entries. The `late` list contains all enabled methods: `stacked`, `majority`, and `weighted_average`. Each method includes split-specific probabilities, classes, and metrics; learned averaging also includes final weights and, for training, the weights fitted outside each fold. Existing flat outputs remain available. `late_methods` lists the returned late-fusion methods. Fusion entries contain `train_prob`, `train_class`, a `metrics` table, and scalar `train_accuracy`, `train_balanced_accuracy`, `train_auc`, and `train_logloss`; validation entries use the corresponding `valid_` names. AUC is the macro average of available one-vs-rest class AUCs. Single-layer entries contain probability lists and class data frames. Without validation data, `valid_out` is `NULL`. Disabled early fusion gives `early = NULL`, and disabled late methods are omitted from `late`.

Feature importance retains a univariate class-association calculation: for each feature and class, the mean in the remaining classes is subtracted from the mean in that class and divided by the feature's overall training standard deviation plus `1e-8`. The signed scores form a feature-by-class matrix in `feature_importance_signed_by_class`. `feature_importance_signed_by_layer_by_class` groups these matrices by omics layer and ranks features by mean absolute score across classes. `feature_importance_global` is the unsigned mean absolute score across classes, ranked highest first. These associations use the training features after optional filtering, including features excluded by screening; they are shared across fusion methods and are not model-specific attributions.

Value

A fitted multiclass IntegratedLearner object containing class probabilities, predicted classes, fitted single-layer and fusion models, and training/validation multiclass performance summaries when available.

Progress, summaries, and reproducibility

Call `set.seed()` before fitting to reproduce the data splits and stochastic fitting steps. To display progress, wrap the call in `progressr::with_progress()`. Use `print(fit)` to show a fitted model summary.

Examples

```
set.seed(1)
n <- 24
feature_table <- as.data.frame(rbind(
  matrix(rnorm(3 * n), nrow = 3, dimnames = list(paste0("L1_F", 1:3), paste0("S", 1:n))),
  matrix(rnorm(2 * n), nrow = 2, dimnames = list(paste0("L2_F", 1:2), paste0("S", 1:n)))
))
y <- rep(c("A", "B", "C"), length.out = n)
sample_metadata <- data.frame(
  subjectID = paste0("ID", 1:n), Y = y,
  row.names = colnames(feature_table)
)
feature_metadata <- data.frame(
  featureID = rownames(feature_table),
  featureType = c(rep("Layer1", 3), rep("Layer2", 2)),
  row.names = rownames(feature_table)
)
fit <- IL_multiclass(
  feature_table = feature_table,
  sample_metadata = sample_metadata,
  feature_metadata = feature_metadata,
  folds = 3, base_learner = "multinom", meta_learner = "multinom",
  run_stacked = FALSE, run_concat = FALSE
)
fit$family
fit$late_methods
fit$train_out$late$weighted_average$weights
head(fit$feature_importance_signed_by_layer_by_class[[1]])
```

ILsurv *IntegratedLearner Survival Engine*

Description

BioC-friendly survival backend used by `IntegratedLearner()` for time-to-event outcomes. This preserves the historical `ILsurv()` interface while using model fitting and fusion routines that do not depend on **mlr3proba/mlr3extralearners**.

Usage

```
ILsurv(
  feature_table,
  sample_metadata,
```

```
  feature_metadata,
  valid_feature_table = NULL,
  valid_sample_metadata = NULL,
  base_learner = "surv.coxph",
  folds = 5,
  run_screening = FALSE,
  screen_pct = NULL,
  drop_poor_performing_layers = FALSE,
  do_early_fusion = TRUE,
  t_vec = NULL,
  t_vec_probs = c(0.05, 0.25, 0.5, 0.75, 0.95),
  layer_score = c("sum", "mean", "l2"),
  eps = 1e-12,
  weight_lambda = 0.02,
  weight_penalty = c("l2_to_uniform", "entropy"),
  weight_cap = 1,
  optim_maxit_cox = 4000,
  optim_maxit_ibs = 300,
  ibs_shrink_to_uniform = 0,
  run_intermediate = FALSE,
  cooperative_rho = c(0, 0.1, 0.25, 0.5, 1),
  cooperative_s = "lambda.min",
  cooperative_type_measure = NULL,
  ...
)

IL_survival(
  feature_table,
  sample_metadata,
  feature_metadata,
  valid_feature_table = NULL,
  valid_sample_metadata = NULL,
  base_learner = "surv.coxph",
  folds = 5,
  run_screening = FALSE,
  screen_pct = NULL,
  drop_poor_performing_layers = FALSE,
  do_early_fusion = TRUE,
  t_vec = NULL,
  t_vec_probs = c(0.05, 0.25, 0.5, 0.75, 0.95),
  layer_score = c("sum", "mean", "l2"),
  eps = 1e-12,
  weight_lambda = 0.02,
  weight_penalty = c("l2_to_uniform", "entropy"),
  weight_cap = 1,
  optim_maxit_cox = 4000,
  optim_maxit_ibs = 300,
  ibs_shrink_to_uniform = 0,
```

```
  run_intermediate = FALSE,
  cooperative_rho = c(0, 0.1, 0.25, 0.5, 1),
  cooperative_s = "lambda.min",
  cooperative_type_measure = NULL,
  ...
)
```

Arguments

feature_table	An R data frame containing multi-layer features (in rows) and samples (in columns). Column names of feature_metadata must match the row names of sample_metadata.
sample_metadata	An R data frame of metadata variables (in columns). Must have a column named subjectID describing per-subject unique identifiers. For longitudinal designs, this variable may be non-unique. Additionally, a column named Y must be present which is the continuous or binary outcome of interest. Row names of sample_metadata must match the column names of feature_table.
feature_metadata	An R data frame of feature-specific metadata across views (in columns) and features (in rows). Must have a column named featureID as a unique per-feature identifier and a column named featureType describing the source layers. Row names of feature_metadata must match the row names of feature_table.
valid_feature_table	Validation feature table (features x samples).
valid_sample_metadata	Validation sample metadata containing time and event.
base_learner	Survival base learner.
folds	How many folds in the V-fold nested cross-validation? Default is 5.
run_screening	Logical; if TRUE, run supervised screening within each training fold (and again on full training data) before fitting base models.
screen_pct	Percentage of features to retain during screening ((0,100]). Applied after any optional filtering.
drop_poor_performing_layers	Logical; if TRUE, layers with single-layer performance below the screening threshold are removed from early and late fusion only. The threshold is AUC < 0.5 for binary and $R^2 < 0.5$ for continuous outcomes. Single-layer outputs are still retained and reported.
do_early_fusion	Logical; run early fusion model.
t_vec	Optional explicit time points for COX-based late-fusion feature construction.
t_vec_probs	Quantiles used to derive t_vec when t_vec is NULL.
layer_score	Layer summary on cumhaz increments: 'sum', 'mean', or 'l2'.
eps	Numerical lower bound for survival probabilities.
weight_lambda	COX-weight optimization regularization strength.
weight_penalty	COX-weight optimization regularizer: 'l2_to_uniform' or 'entropy'.

`weight_cap` Optional cap on any one layer weight (COX method).

`optim_maxit_cox`
Maximum iterations for COX-weight optimization.

`optim_maxit_ibs`
Maximum iterations for IBS-weight optimization.

`ibs_shrink_to_uniform`
Optional convex shrinkage of IBS weights toward uniform.

`run_intermediate`
Should standalone cooperative learning via **multiview** be run? Default is FALSE. When TRUE, only the cooperative multiview Cox model is fit; survival base learners, early fusion, and late fusion are skipped.

`cooperative_rho`
Non-negative agreement-penalty values to evaluate for cooperative learning.

`cooperative_s` Penalty value used when predicting from the selected cooperative model. Defaults to `"lambda.min"`.

`cooperative_type_measure`
Optional **multiview** CV loss/score. If `NULL`, Harrell's C is used.

`...` Additional base-learner hyperparameters passed to `base_learner`. You may also pass `model_args = list(...)` as a named list where each entry is keyed by learner ID.

Details

Risk-to-survival Cox calibration is fitted using training outcomes and OOF risk scores, then held fixed for validation, including coefficients, baseline survival, missing-risk replacement, and the Brier time grid. A training Kaplan-Meier curve is used if Cox calibration is not estimable. Validation outcomes are used only for performance evaluation, including estimating censoring weights, and never to fit a prediction calibration model. IBS fusion is scored directly on its weighted survival curves; single-layer, early-fusion, Cox-fusion, and cooperative models use their frozen training calibration. Reported training probability-error metrics remain internal summaries because calibration and fusion fitting use training outcomes; they are not fully nested evaluation estimates.

Value

List with `train_out` and `valid_out`. Single-layer survival outputs include `train_out$single$metrics`, `train_out$single$train_risk`, and `train_out$single$feature_importance_signed_by_layer`, a named list of unweighted signed feature-importance scores by layer. Late-fusion results are returned under `train_out$late$IBS`, `train_out$late$COX`, and the analogous validation entries. When `run_intermediate = TRUE`, `train_out$cooperative` includes coefficient-based cooperative feature-importance scores from the selected **multiview** Cox model. `brier_time_grid` gives the shared training-derived grid used for training and validation Brier/IBS evaluation. `survival_calibration` stores the training-fitted risk-to-survival mappings. Single, early, late, and cooperative entries include `train_surv`/`valid_surv` probabilities when the corresponding model is fitted (lists for single layers, matrices for fusion models).

Progress, summaries, and reproducibility

Call `set.seed()` before fitting to reproduce the data splits and stochastic fitting steps. To display progress, wrap the call in `progressr::with_progress()`. Use `print(fit)` to show a fitted model summary.

Examples

```
set.seed(1)
n <- 20
feature_table <- as.data.frame(rbind(
  matrix(rnorm(3 * n), nrow = 3, dimnames = list(paste0("L1_F", 1:3), paste0("S", 1:n))),
  matrix(rnorm(2 * n), nrow = 2, dimnames = list(paste0("L2_F", 1:2), paste0("S", 1:n)))
))
sample_metadata <- data.frame(
  subjectID = paste0("ID", 1:n),
  time = rexp(n, rate = 0.1),
  event = rbinom(n, 1, 0.6),
  row.names = colnames(feature_table)
)
sample_metadata$Y <- survival::Surv(sample_metadata$time, sample_metadata$event)
feature_metadata <- data.frame(
  featureID = rownames(feature_table),
  featureType = c(rep("Layer1", 3), rep("Layer2", 2)),
  row.names = rownames(feature_table)
)
fit <- ILsurv(
  feature_table = feature_table,
  sample_metadata = sample_metadata,
  feature_metadata = feature_metadata,
  folds = 3
)
names(fit)
fit$brier_time_grid
```

IntegratedLearner — *Integrated machine learning for multi-omics prediction and classification*

Description

Performs integrated machine learning to predict a binary, continuous, or time-to-event outcome based on two or more omics layers (views). The `IntegratedLearner` function takes a training `MultiAssayExperiment` and, optionally, a validation `MultiAssayExperiment`, inspects standard Bioconductor containers and accessors, extracts multi-layer feature tables, and returns predicted values based on the validation set. It also performs V-fold nested cross-validation to estimate the prediction accuracy of various fusion algorithms. Two integration paradigms are supported: early and late. The software includes multiple ML models based on the `SuperLearner` R package as well as several data exploration capabilities and visualization modules in a unified estimation framework.

Usage

```
IntegratedLearner(
  MAE_train = NULL,
  MAE_valid = NULL,
  PCL_train = NULL,
  PCL_valid = NULL,
  experiment = NULL,
  assay.type = NULL,
  outcome_col = "Y",
  subject_id_col = "subjectID",
  na.rm = FALSE,
  folds = 5,
  base_learner = "sl_bart",
  base_screener = "All",
  run_screening = FALSE,
  screen_pct = NULL,
  filter_method = NULL,
  filter_pct = NULL,
  prevalence_pct = NULL,
  meta_learner = "sl_nnls_auc",
  run_concat = TRUE,
  run_stacked = TRUE,
  run_intermediate = FALSE,
  cooperative_rho = c(0, 0.1, 0.25, 0.5, 1),
  cooperative_s = "lambda.min",
  cooperative_type_measure = NULL,
  drop_poor_performing_layers = FALSE,
  refit.stack = FALSE,
  family = stats::gaussian(),
  run_majority = TRUE,
  run_weighted = TRUE,
  ...
)
```

Arguments

MAE_train	A MultiAssayExperiment containing the training data. Each experiment corresponds to one view (omics layer), usually stored as a SummarizedExperiment or TreeSummarizedExperiment. The colData of every selected experiment must contain a column named by outcome_col describing the outcome of interest (binary, continuous, or survival-encoded), and a subject identifier column named by subject_id_col.
MAE_valid	Optional MultiAssayExperiment containing an independent validation/test set for which prediction is desired. It must contain the same set of experiments used for training, with identical assay names and identical feature (row) names for each experiment. Additional experiments in mae_test are ignored.
PCL_train	Optional list of per-layer feature matrices (legacy PCL mode for backward compatibility).

PCL_valid	Optional validation list of per-layer feature matrices.
experiment	Optional character or integer vector specifying which experiments (layers) to extract from MAE_train (and MAE_valid, if supplied). If NULL, all experiments in MAE_train are used.
assay.type	Optional character vector of assay names, one per experiment. If NULL, each selected experiment must contain exactly one assay; otherwise, the user must supply assay.type.
outcome_col	Outcome column name in MAE/PCL sample metadata. Defaults to "Y" for backward compatibility.
subject_id_col	Subject identifier column name in MAE/PCL sample metadata. Defaults to "subjectID" for backward compatibility.
na.rm	Logical; if TRUE, features with missing values are dropped after extraction.
folds	How many folds in the V-fold nested cross-validation? Default is 5.
base_learner	Base learner for late fusion and early fusion. Check out the SuperLearner package page for all available options. Default is `sl_bart`.
base_screener	Deprecated for IL_conbin and IL_multiclass; kept for backward compatibility and currently ignored in those backends.
run_screening	Logical; if TRUE, run supervised screening on training data within each CV fold (and again on full training data before final fitting), then apply the same selected features to fold-validation or external validation data.
screen_pct	Percentage of features to retain during screening ((0,100]).
filter_method	Optional feature-filter method for non-survival runs. Supported values: 'prevalence' and 'variance'.
filter_pct	Optional retention percentage (in (0,100]) for the selected filter_method. Keeps the top filter_pct percent features.
prevalence_pct	Optional retention percentage (in (0,100]) for prevalence-based feature filtering before model fitting. Deprecated alias of filter_pct with filter_method = 'prevalence'.
meta_learner	Meta-learner for late fusion (stacked generalization). Defaults to `sl_nnls_auc`. For multiclass outcomes this legacy default maps to multinomial glmnet; use run_weighted for learned scalar layer weights. Check out the SuperLearner package page for all available options.
run_concat	Should early fusion be run? Default is TRUE. Uses the specified base_learner as the learning algorithm.
run_stacked	Should stacked model (late fusion) be run? Default is TRUE.
run_intermediate	Should standalone cooperative learning via **multiview** be run? Default is FALSE. When TRUE, IntegratedLearner runs only the cooperative multiview model and skips base learners, early fusion, and late fusion. Supported for continuous, binary, and survival outcomes; ignored with a message for multiclass outcomes.
cooperative_rho	Non-negative agreement-penalty values to evaluate for cooperative learning. A separate cv.multiview() fit is trained for each value, and the best value is selected by cross-validation.

`cooperative_s` Penalty value used when predicting from the selected cooperative model. Defaults to `"lambda.min"`.

`cooperative_type_measure` Optional **multiview** CV loss/score. If `NULL`, an outcome-appropriate default is used.

`drop_poor_performing_layers` Logical; if `TRUE`, layers with poor single-layer performance are removed from early and late fusion only. The thresholds are AUC < 0.5 for binary, R^2 < 0.5 for continuous, and C-index < 0.5 for survival outcomes. Single-layer outputs are still retained and reported.

`refit.stack` logical; For late fusion, post-refit predictions on the entire data are returned if specified. Default is FALSE.

`family` Currently allows `` `gaussian()` `` for continuous, `` `binomial()` `` for binary, or an appropriate Cox/survival family for time-to-event outcomes. Determines whether `IL_conbin` or `IL_survival` is used internally.

`run_majority` Logical; return multiclass majority voting. Default TRUE; ignored for other outcomes. Ties use the first class in training class order.

`run_weighted` Logical; return multiclass probability averaging with nonnegative layer weights summing to one, learned by minimizing out-of-fold multiclass log loss. Default TRUE; ignored for other outcomes.

`...` Additional arguments passed to the underlying engines.

Details

In a typical Bioconductor workflow, users construct assay-level data in `SummarizedExperiment`, combine those assays in `MultiAssayExperiment`, inspect the object with `experiments()`, `assay()`, `colData()`, and `sampleMap()`, and then pass the MAE into `IntegratedLearner()` for supervised multi-omics prediction. Internally, `IntegratedLearner()` converts the `MultiAssayExperiment` into tabular multi-view matrices and then calls `IL_conbin` for continuous/binary outcomes or `IL_survival` for time-to-event outcomes, depending on the specified `family`. Legacy PCL-style inputs remain available for backward compatibility, but the primary user-facing workflow is MAE-based.

Value

A list-like IntegratedLearner object containing the trained model fits (layer-specific, stacked, and/or concatenated models), cross-validated performance estimates, and predicted values for training and, if supplied, validation data. When `run_intermediate = TRUE`, the object also includes coefficient-based cooperative feature-importance scores from the selected **multiview** model. Multiclass fits also provide `train_out` and `valid_out` grouped into `single`, `early`, and `late` results. All enabled late methods (`stacked`, `majority`, `weighted_average`) include probabilities, predicted classes, and metrics; learned averaging also reports layer weights. Existing flat multiclass outputs remain available.

Progress, summaries, and reproducibility

Call `set.seed()` before fitting to reproduce the data splits and stochastic fitting steps. To display progress, wrap the call in `progressr::with_progress()`. Use `print(fit)` to show a fitted model summary.

Author(s)

Himel Mallick, <him4004@med.cornell.edu>

Examples

```
PRISM_MAE <- readRDS(system.file(
  "extdata", "PRISM_MAE.rds", package = "IntegratedLearner", mustWork = TRUE
))
mae <- PRISM_MAE
names(MultiAssayExperiment::experiments(mae))
head(as.data.frame(S4Vectors::DataFrame(MultiAssayExperiment::colData(mae)))[, 1:4])
head(MultiAssayExperiment::sampleMap(mae))
se <- MultiAssayExperiment::experiments(mae)[[1]]
SummarizedExperiment::assayNames(se)
fit <- IntegratedLearner(
  MAE_train = mae,
  experiment = names(MultiAssayExperiment::experiments(mae))[1:2],
  assay.type = rep("abundance", 2),
  folds = 2, base_learner = "SL.mean",
  run_stacked = FALSE, run_concat = FALSE,
  family = stats::binomial()
)
names(fit)
```

NNLS *NNLS function to optimize weights of several base learners*

Description

NNLS function to optimize weights of several base learners

Usage

```
NNLS(x, y, wt)
```

Arguments

x	x
y	y
wt	wt

Value

Solution of the quadratic programming problem

Examples

```
x <- cbind(c(0.1, 0.4, 0.6, 0.9), c(0.2, 0.5, 0.7, 0.8))
y <- c(0.15, 0.45, 0.65, 0.85)
fit <- NNLS(x = x, y = y, wt = rep(1, nrow(x)))
fit$solution
```

`plot.learner` *Plot the summary curves produced by an IntegratedLearner object*

Description

Plots outcome-appropriate performance summaries for the training set and, if available, the validation set produced by an `IntegratedLearner` fit. Depending on the outcome family, this may include ROC curves, R-squared bar plots, multiclass one-vs-rest ROC curves, or survival AUC / Kaplan-Meier panels. Multiclass plots include all enabled single-layer and fusion outputs, including stacking, majority voting, and learned probability averaging. Majority ROC curves use vote fractions, which are not calibrated probabilities.

Usage

```
## S3 method for class 'learner'
plot(
  x,
  y = NULL,
  label_size = 8,
  label_x = 0.3,
  vjust = 0.1,
  rowwise_plot = TRUE,
  ...
)
```

Arguments

`x`	Fitted `IntegratedLearner` object.
`y`	Unused (required for S3 signature).
`label_size`	Optional numeric label size for subplot tags. Default is 8.
`label_x`	Optional single value or vector of x positions for subplot labels, relative to each subplot. Defaults to 0.3 for all labels.
`vjust`	Adjusts the vertical position of each label. More positive values move the label further down on the plot canvas. Can be a single value (applied to all labels) or a vector of values (one for each label). Default is 0.1.
`rowwise_plot`	If both train and test data are available, should the train and test plots be arranged row-wise? Default is `TRUE`. If `FALSE`, plots are aligned column-wise.
`...`	Additional arguments (currently unused).

Value

A list whose $plot entry is a **ggplot2/cowplot** composite object, along with the underlying tabular data used to generate the plot.

Examples

```
PRISM_MAE <- readRDS(system.file(
  "extdata", "PRISM_MAE.rds", package = "IntegratedLearner", mustWork = TRUE
))
mae <- PRISM_MAE
se <- MultiAssayExperiment::experiments(mae)[[1]]
head(SummarizedExperiment::assay(se, SummarizedExperiment::assayNames(se)[1]))
fit <- IntegratedLearner(
  MAE_train = mae,
  experiment = names(MultiAssayExperiment::experiments(mae))[1:2],
  assay.type = rep("abundance", 2),
  folds = 2, base_learner = "SL.mean",
  run_stacked = FALSE, run_concat = FALSE,
  family = stats::binomial()
)
plt <- plot(fit)
names(plt)
```

---

predict.learner *Make predictions using a trained 'IntegratedLearner' model*

Description

This function makes predictions using a trained 'IntegratedLearner' model for new samples for which predictions are to be made

Usage

```
## S3 method for class 'learner'
predict(
  object,
  feature_table_valid = NULL,
  sample_metadata_valid = NULL,
  feature_metadata = NULL,
  outcome_col = NULL,
  subject_id_col = NULL,
  ...
)
```

Arguments

object	Fitted 'IntegratedLearner' object

feature_table_valid
: Feature table from validation set. Must have the exact same structure as feature_table.

sample_metadata_valid
: OPTIONAL (can provide feature_table_valid and not this): Sample-specific metadata table from independent validation set. If provided, it must have the exact same structure as sample_metadata.

feature_metadata
: Matrix containing feature names and their corresponding layers. Must be same as that provided in IntegratedLearner object.

outcome_col
: Optional outcome column name in sample_metadata_valid. If NULL, uses the mapping stored in object$column_map (or falls back to "Y").

subject_id_col
: Optional subject identifier column name in sample_metadata_valid. If NULL, uses the mapping stored in object$column_map (or falls back to "subjectID").

...
: Additional arguments (currently unused)

Value

For gaussian and binomial fits, a list with at least

yhat.test
: A data frame or matrix of predictions for each single layer and, when requested during training, the stacked and concatenated fusion predictions.

Y_test
: Observed validation outcomes, returned only when sample_metadata_valid is supplied.

AUC.test, accuracy.test, balanced_accuracy.test, metrics.test
: Binary-outcome performance summaries, returned only when validation outcomes are supplied for binomial fits.

R2.test
: Named vector of validation-set R^2 values, returned only when validation outcomes are supplied for gaussian fits.

For multiclass fits, returns the output of predict_multiclass.learner(), which contains class probabilities, predicted classes, and optional validation metrics. The flat outputs are prob.test, class.test, and yhat.test, with metrics.test when outcomes are available. Grouped results are returned in valid_out$single, valid_out$early, and valid_out$late. The latter includes all methods enabled during fitting: stacked, majority, and weighted_average, as listed in late_methods. Each fusion method includes valid_prob, valid_class, and a metrics table, plus valid_accuracy, valid_balanced_accuracy, valid_auc, and valid_logloss when labels are available. Without labels, metrics is NULL and scalar metrics are absent. Learned averaging also includes the training-derived layer weights; prediction does not refit them. Majority outputs are vote fractions; ties use training class order, and the fractions are not calibrated probabilities. Disabled early fusion returns early = NULL, and disabled late methods are omitted from late.

Examples

```
set.seed(1)
sample_ids <- paste0("S", seq_len(18))
layer1 <- matrix(rnorm(3 * length(sample_ids)),
  nrow = 3,
  dimnames = list(paste0("L1_F", 1:3), sample_ids)
)
```

```
layer2 <- matrix(rnorm(2 * length(sample_ids)),
  nrow = 2,
  dimnames = list(paste0("L2_F", 1:2), sample_ids)
)
signal <- colMeans(layer1)
sample_metadata <- data.frame(
  Y = as.numeric(signal > stats::median(signal)),
  subjectID = sample_ids,
  row.names = sample_ids
)
feature_table <- as.data.frame(rbind(layer1, layer2))
feature_metadata <- data.frame(
  featureID = rownames(feature_table),
  featureType = c(rep("Layer1", 3), rep("Layer2", 2)),
  row.names = rownames(feature_table)
)
fit <- IntegratedLearner(
  PCL_train = list(
    feature_table = feature_table,
    sample_metadata = sample_metadata,
    feature_metadata = feature_metadata
  ),
  folds = 2, base_learner = "SL.mean",
  run_stacked = FALSE, run_concat = FALSE,
  family = stats::binomial()
)
pred <- predict(
  fit,
  feature_table_valid = feature_table,
  sample_metadata_valid = sample_metadata,
  feature_metadata = feature_metadata
)
names(pred)
```

`predict.sl_bart` *Predict function for sl_bart*

Description

Predict function for sl_bart

Usage

```
## S3 method for class 'sl_bart'
predict(object, newdata, family, X = NULL, Y = NULL, ...)
```

Arguments

`object`	Fitted sl_bart model object
`newdata`	Data frame for prediction

family	Family object passed through (unused)
X	Training design matrix (unused)
Y	Training outcome (unused)
...	Additional arguments (unused)

Value

Prediction from the sl_bart model

Examples

```
if (
  requireNamespace("bartMachine", quietly = TRUE) &&
  nzchar(Sys.which("java")) &&
 any(grepl("jdk.incubator.vector", getOption("java.parameters", character()), fixed = TRUE))
) {
  set.seed(1)
  X <- data.frame(x1 = rnorm(20), x2 = rnorm(20))
  Y <- rnorm(20)
  fit <- sl_bart(
    Y = Y, X = X, newX = X[1:3, ],
    family = stats::gaussian(),
    obsWeights = rep(1, nrow(X)), id = seq_len(nrow(X)),
    num_trees = 5, num_burn_in = 5, num_iterations_after_burn_in = 20
  )
  head(predict(fit$fit, newdata = X[1:3, ], family = stats::gaussian()))
}
```

---

predict.sl_nnls_auc *Predict function for sl_nnls_auc*

Description

Predict function for sl_nnls_auc

Usage

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

Arguments

object	Fitted sl_nnls_auc model
newdata	Validation layer-level predictions
...	Additional arguments (unused)

Value

Prediction from the meta-learner

Examples

```
set.seed(1)
X <- data.frame(m1 = runif(20), m2 = runif(20))
Y <- rnorm(20)
fit <- sl_nnls_auc(
  Y = Y, X = X, newX = X[1:4, ],
  family = stats::gaussian(), obsWeights = rep(1, nrow(X))
)
head(predict(fit$fit, newdata = X[1:4, ]))
```

---

sl_bart *Wrapper for bartMachine learner*

Description

Support bayesian additive regression trees via the bartMachine package.

Usage

```
sl_bart(
  Y,
  X,
  newX,
  family,
  obsWeights,
  id,
  num_trees = 50,
  num_burn_in = 250,
  verbose = FALSE,
  alpha = 0.95,
  beta = 2,
  k = 2,
  q = 0.9,
  nu = 3,
  num_iterations_after_burn_in = 1000,
  serialize = TRUE,
  seed = 5678,
  ...
)
```

Arguments

Y	Outcome variable
X	Covariate dataframe
newX	Optional dataframe to predict the outcome
family	'gaussian' for regression, 'binomial' for binary classification
obsWeights	Optional observation-level weights (supported but not tested)
id	Optional id to group observations from the same unit (not used currently).
num_trees	The number of trees to be grown in the sum-of-trees model.
num_burn_in	Number of MCMC samples to be discarded as 'burn-in'.
verbose	Prints information about progress of the algorithm to the screen.
alpha	Base hyperparameter in tree prior for whether a node is nonterminal or not.
beta	Power hyperparameter in tree prior for whether a node is nonterminal or not.
k	For regression, k determines the prior probability that E(Y|X) is contained in the interval (y_{min}, y_{max}), based on a normal distribution. For example, when k=2, the prior probability is 95 percent. For classification, k determines the prior probability that E(Y|X) is between (-3,3). Note that a larger value of k results in more shrinkage and a more conservative fit.
q	Quantile of the prior on the error variance at which the data-based estimate is placed. Note that the larger the value of q, the more aggressive the fit as you are placing more prior weight on values lower than the data-based estimate. Not used for classification.
nu	Degrees of freedom for the inverse chi^2 prior. Not used for classification.
num_iterations_after_burn_in	Number of MCMC samples to draw from the posterior distribution of f(x).
serialize	If TRUE, bartMachine results can be saved to a file, but will require additional RAM.
seed	Seed for reproducibility passed to bartMachine.
...	Additional arguments (not used)

Value

A list with elements pred (predictions for newX) and fit (the fitted model object).

Examples

```
if (
  requireNamespace("bartMachine", quietly = TRUE) &&
  nzchar(Sys.which("java")) &&
 any(grepl("jdk.incubator.vector", getOption("java.parameters", character()), fixed = TRUE))
) {
  set.seed(1)
  X <- data.frame(x1 = rnorm(20), x2 = rnorm(20))
  Y <- rnorm(20)
  fit <- sl_bart(
```

```
    Y = Y, X = X, newX = X[1:3, ],
    family = stats::gaussian(),
    obsWeights = rep(1, nrow(X)), id = seq_len(nrow(X)),
    num_trees = 5, num_burn_in = 5, num_iterations_after_burn_in = 20
  )
  fit$pred
}
```

`sl_enet` *Elastic net regression with optimized alpha and lambda*

Description

Selects alpha by cross-validation, then uses the shared glmnet fitting implementation.

Usage

```
sl_enet(
  Y,
  X,
  newX,
  family,
  obsWeights,
  id,
  alpha = seq(0, 1, 0.1),
  nfolds = 10,
  nlambda = 100,
  useMin = TRUE,
  loss = "deviance",
  ...
)
```

Arguments

`Y`	Outcome variable
`X`	Covariate dataframe
`newX`	Dataframe to predict the outcome
`family`	'gaussian' for regression, 'binomial' for binary classification. Untested options: 'multinomial' for multiple classification or 'mgaussian' for multiple response, 'poisson' for non-negative outcome with proportional mean and variance, 'cox'.
`obsWeights`	Optional observation-level weights
`id`	Optional id to group observations from the same unit (not used currently).
`alpha`	Numeric grid of elastic net mixing parameters between 0 and 1.
`nfolds`	Number of folds for internal cross-validation to optimize lambda.

nlambda	Number of lambda values to check, recommended to be 100 or more.
useMin	If TRUE use lambda that minimizes risk, otherwise use 1 standard-error rule which chooses a higher penalty with performance within one standard error of the minimum (see Breiman et al. 1984 on CART for background).
loss	Loss function, can be 'deviance', 'mse', or 'mae'. If family = binomial can also be 'auc' or 'class' (misclassification error).
...	Any additional arguments are passed through to cv.glmnet.

Value

A list with elements pred (predictions for newX) and fit (cross-validated glmnet fit metadata).

References

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1), 1.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(1), 55-67.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), 267-288.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301-320.

See Also

predict.SL.glmnet cv.glmnet glmnet

Examples

```
set.seed(1)
X <- data.frame(x1 = rnorm(20), x2 = rnorm(20))
Y <- rnorm(20)
fit <- sl_enet(
  Y = Y, X = X, newX = X[1:3, ], family = stats::gaussian(),
  obsWeights = rep(1, nrow(X)), id = seq_len(nrow(X)),
  alpha = c(0, 0.5, 1), nfolds = 3, nlambda = 10
)
head(fit$pred)
```

`sl_glmnet2` *Elastic net regression, including lasso and ridge with a fixed alpha*

Description

Penalized regression using elastic net. Alpha = 0 corresponds to ridge regression and alpha = 1 corresponds to Lasso.

See `vignette('glmnet_beta', package = 'glmnet')` for a nice tutorial on glmnet.

Usage

```
sl_glmnet2(
  Y,
  X,
  newX,
  family,
  obsWeights,
  id,
  alpha = 0.5,
  nfolds = 10,
  nlambda = 100,
  useMin = TRUE,
  loss = "deviance",
  ...
)
```

Arguments

`Y`	Outcome variable
`X`	Covariate dataframe
`newX`	Dataframe to predict the outcome
`family`	'gaussian' for regression, 'binomial' for binary classification. Untested options: 'multinomial' for multiple classification or 'mgaussian' for multiple response, 'poisson' for non-negative outcome with proportional mean and variance, 'cox'.
`obsWeights`	Optional observation-level weights
`id`	Optional id to group observations from the same unit (not used currently).
`alpha`	Elastic net mixing parameter, range 0 to 1. 0 = ridge regression and 1 = lasso.
`nfolds`	Number of folds for internal cross-validation to optimize lambda.
`nlambda`	Number of lambda values to check, recommended to be 100 or more.
`useMin`	If TRUE use lambda that minimizes risk, otherwise use 1 standard-error rule which chooses a higher penalty with performance within one standard error of the minimum (see Breiman et al. 1984 on CART for background).
`loss`	Loss function, can be 'deviance', 'mse', or 'mae'. If family = binomial can also be 'auc' or 'class' (misclassification error).
`...`	Any additional arguments are passed through to cv.glmnet.

Value

A list with elements `pred` (predictions for `newX`) and `fit` (cross-validated glmnet fit metadata).

References

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1), 1.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(1), 55-67.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), 267-288.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301-320.

See Also

`predict.SL.glmnet` `cv.glmnet` `glmnet`

Examples

```
set.seed(1)
X <- data.frame(x1 = rnorm(20), x2 = rnorm(20))
Y <- rnorm(20)
fit <- sl_glmnet2(
  Y = Y, X = X, newX = X[1:3, ], family = stats::gaussian(),
  obsWeights = rep(1, nrow(X)), id = seq_len(nrow(X)),
  nfolds = 3, nlambda = 10
)
head(fit$pred)
```

`sl_horseshoe` *Horseshoe regression*

Description

Horseshoe regression

Usage

```
sl_horseshoe(
  Y,
  X,
  newX,
  family,
  prior = "horseshoe",
  N = 20000L,
```

```
  burnin = 1000L,
  thinning = 1L,
  ...
)
```

Arguments

Y	Outcome variable
X	Covariate data frame
newX	Dataframe to predict the outcome
family	'gaussian' for regression, 'binomial' for binary classification. Untested options: 'poisson' for for integer or count data
prior	prior for regression coefficients to use. 'Horseshoe' by default. Untested options: ridge regression (prior='rr' or prior='ridge'), lasso regression (prior='lasso') and horseshoe+ regression (prior='hs+' or prior='horseshoe+')
N	Number of posterior samples to generate.
burnin	Number of burn-in samples.
thinning	Desired level of thinning.
...	other parameters passed to bayesreg function

Value

A list with elements `pred` (predictions for `newX`) and `fit` (the fitted Bayesian regression object).

Examples

```
if (
  requireNamespace("bayesreg", quietly = TRUE) &&
  identical(Sys.getenv("INTEGRATEDLEARNER_RUN_BAYESREG_EXAMPLES"), "true")
) {
  set.seed(1)
  X <- data.frame(x1 = rnorm(20), x2 = rnorm(20))
  Y <- rnorm(20)
  fit <- sl_horseshoe(
    Y = Y, X = X, newX = X[1:3, ],
    family = stats::gaussian()
  )
  fit$pred
}
```

`sl_lasso` *Lasso regression with a fixed alpha*

Description

Uses the shared glmnet fitting implementation with `alpha = 1` by default.

Usage

```
sl_lasso(
  Y,
  X,
  newX,
  family,
  obsWeights,
  id,
  alpha = 1,
  nfolds = 10,
  nlambda = 100,
  useMin = TRUE,
  loss = "deviance",
  ...
)
```

Arguments

`Y`	Outcome variable
`X`	Covariate dataframe
`newX`	Dataframe to predict the outcome
`family`	'gaussian' for regression, 'binomial' for binary classification. Untested options: 'multinomial' for multiple classification or 'mgaussian' for multiple response, 'poisson' for non-negative outcome with proportional mean and variance, 'cox'.
`obsWeights`	Optional observation-level weights
`id`	Optional id to group observations from the same unit (not used currently).
`alpha`	Elastic net mixing parameter, range 0 to 1. 0 = ridge regression and 1 = lasso.
`nfolds`	Number of folds for internal cross-validation to optimize lambda.
`nlambda`	Number of lambda values to check, recommended to be 100 or more.
`useMin`	If TRUE use lambda that minimizes risk, otherwise use 1 standard-error rule which chooses a higher penalty with performance within one standard error of the minimum (see Breiman et al. 1984 on CART for background).
`loss`	Loss function, can be 'deviance', 'mse', or 'mae'. If family = binomial can also be 'auc' or 'class' (misclassification error).
`...`	Any additional arguments are passed through to cv.glmnet.

Value

A list with elements `pred` (predictions for `newX`) and `fit` (cross-validated glmnet fit metadata).

References

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1), 1.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(1), 55-67.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), 267-288.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301-320.

See Also

`predict.SL.glmnet` `cv.glmnet` `glmnet`

Examples

```
set.seed(1)
X <- data.frame(x1 = rnorm(20), x2 = rnorm(20))
linpred <- 0.5 * X$x1 - 0.25 * X$x2
Y <- stats::rbinom(nrow(X), 1, stats::plogis(linpred))
fit <- sl_lasso(
  Y = Y, X = X, newX = X[1:3, ], family = stats::binomial(),
  obsWeights = rep(1, nrow(X)), id = seq_len(nrow(X)),
  nfolds = 3, nlambda = 10
)
head(fit$pred)
```

---

sl_nnls_auc *Combined SuperLearner function for both NNLS/AUC maximization*

Description

Combined SuperLearner function for both NNLS/AUC maximization

Usage

```
sl_nnls_auc(Y, X, newX, family, obsWeights, bounds = c(0, Inf), ...)
```

Arguments

Y	Outcome matrix from metalearner
X	Layer-level predictions used to train the metalearner
newX	Layer-level predictions for validation data
family	Family object
obsWeights	Observation weights
bounds	Lower/upper bounds for weights (binomial case)
...	Additional arguments passed through

Value

Estimated meta-learner coefficients and predictions

Examples

```
set.seed(1)
X <- data.frame(m1 = runif(20), m2 = runif(20))
Y <- rnorm(20)
fit <- sl_nnls_auc(
  Y = Y, X = X, newX = X[1:4, ],
  family = stats::gaussian(), obsWeights = rep(1, nrow(X))
)
head(fit$pred)
```

---

update.learner *Update IntegratedLearner fit object based on layers available in the test set*

Description

Allow update of IntegratedLearner if only a subset of omics layers are available in test set. If all layers and features match, it calls predict.learner()

Usage

```
## S3 method for class 'learner'
update(
  object,
  feature_table_valid,
  sample_metadata_valid = NULL,
  feature_metadata_valid,
  outcome_col = NULL,
  subject_id_col = NULL,
  ...
)
```

Arguments

`object`	fitted 'IntegratedLearner' object
`feature_table_valid`	Feature table from validation set. It should be a data frame with features in rows and samples in columns. Feature names should be a subset of training data feature names.
`sample_metadata_valid`	OPTIONAL (can provide feature_table_valid and not this): Sample-specific metadata table from independent validation set. If provided, it must have the exact same structure as sample_metadata. Default is NULL.
`feature_metadata_valid`	Matrix containing feature names and their corresponding layers. Must be subset of feature_metadata provided in IntegratedLearner object.
`outcome_col`	Optional outcome column name in `sample_metadata_valid`. If `NULL`, uses `object$column_map$outcome_col` (or `"Y"`).
`subject_id_col`	Optional subject ID column name in `sample_metadata_valid`. If `NULL`, uses `object$column_map$subject_id_col` (or `"subjectID"`).
`...`	Additional arguments (unused)

Value

An updated IntegratedLearner prediction object with `yhat.test` and, when validation outcomes are supplied, the observed outcomes and corresponding performance summaries.

Progress, summaries, and reproducibility

Call `set.seed()` before fitting to reproduce the data splits and stochastic fitting steps. To display progress, wrap the call in `progressr::with_progress()`. Use `print(fit)` to show a fitted model summary.

Examples

```
set.seed(1)
sample_ids <- paste0("S", seq_len(18))
layer1 <- matrix(rnorm(3 * length(sample_ids)),
  nrow = 3,
  dimnames = list(paste0("L1_F", 1:3), sample_ids)
)
layer2 <- matrix(rnorm(2 * length(sample_ids)),
  nrow = 2,
  dimnames = list(paste0("L2_F", 1:2), sample_ids)
)
signal <- colMeans(layer1)
sample_metadata_valid <- data.frame(
  Y = as.numeric(signal > stats::median(signal)),
  subjectID = sample_ids,
  row.names = sample_ids
)
feature_table_valid <- as.data.frame(rbind(layer1, layer2))
```

```
feature_metadata_valid <- data.frame(
  featureID = rownames(feature_table_valid),
  featureType = c(rep("Layer1", 3), rep("Layer2", 2)),
  row.names = rownames(feature_table_valid)
)
fit <- IntegratedLearner(
  PCL_train = list(
    feature_table = feature_table_valid,
    sample_metadata = sample_metadata_valid,
    feature_metadata = feature_metadata_valid
  ),
  folds = 2, base_learner = "SL.mean",
  run_stacked = FALSE, run_concat = FALSE,
  family = stats::binomial()
)
upd <- update(
  object = fit,
  feature_table_valid = feature_table_valid,
  sample_metadata_valid = sample_metadata_valid,
  feature_metadata_valid = feature_metadata_valid
)
names(upd)
```

Index

∗ **metagenomics**
  IL_conbin, 3
  IntegratedLearner, 14
∗ **microbiome**
  IL_conbin, 3
  IntegratedLearner, 14
∗ **multiomics**
  IL_conbin, 3
  IntegratedLearner, 14
∗ **scRNASeq**
  IL_conbin, 3
  IntegratedLearner, 14
∗ **singlecell**
  IL_conbin, 3
  IntegratedLearner, 14
∗ **tweedie**
  IL_conbin, 3
  IntegratedLearner, 14

auc_obj, 2

cv.glmnet, *27*, *29*, *32*

glmnet, *27*, *29*, *32*

IL_conbin, 3
IL_multiclass, 7
IL_survival *(*ILsurv*)*, 10
ILsurv, 10
IntegratedLearner, *6*, 14

NNLS, 18

plot.learner, 19
predict.learner, 20
predict.sl_bart, 22
predict.sl_nnls_auc, 23

sl_bart, 24
sl_enet, 26
sl_glmnet2, 28
sl_horseshoe, 29
sl_lasso, 31
sl_nnls_auc, 32
SuperLearner, *5*, *14*

update.learner, 33