Package: sl3 (via r-universe)

November 14, 2024

Title Pipelines for Machine Learning and Super Learning

Version 1.4.5

Maintainer Jeremy Coyle <jeremyrcoyle@gmail.com>

Description A modern implementation of the Super Learner prediction algorithm, coupled with a general purpose framework for composing arbitrary pipelines for machine learning tasks.

Depends R (>= 3.6.0)

Imports data.table, assertthat, origami (>= 1.0.7), R6, uuid, BBmisc, stats, delayed, utils, methods, ggplot2, digest, Rdpack, dplyr, caret, ROCR

Suggests testthat, rmarkdown, devtools, R.rsp, future, knitr, stringr, reticulate, rgl, rJava, arm, bartMachine, cvAUC, e1071, earth, polspline, forecast, glmnet, grf, gbm, hal9001 (>= 0.4.6), h2o, keras, nloptr, nnls, randomForest, ranger, rpart, Rsolnp, rugarch, speedglm, SuperLearner, tsDyn, xgboost, lightgbm, dbarts, gam, haldensify (>= 0.2.7), mgcv, hts, GA, SIS, partykit

Remotes github::nhejazi/haldensify

Additional_repositories https://tlverse.r-universe.dev

License GPL-3

Language en-US

URL https://tlverse.org/sl3/

BugReports https://github.com/tlverse/sl3/issues

Encoding UTF-8

LazyData yes

LazyLoad yes

VignetteBuilder knitr, R.rsp

Roxygen list(markdown = TRUE, old_usage = TRUE, r6 = FALSE)

RoxygenNote 7.3.2

RdMacros Rdpack

Config/pak/sysreqs libglpk-dev make libicu-dev libxml2-dev

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

RemoteUrl https://github.com/tlverse/sl3

RemoteRef fix-tests

RemoteSha 4bb008e655592123e11cc0135fa1c52d19d2e37a

Contents

`args_to_list` *Get all arguments of parent call (both specified and defaults) as list*

Description

Get all arguments of parent call (both specified and defaults) as list

Usage

```
args_to_list()
```

Value

A `list` of all arguments for the parent function call.

`bsds` *Bicycle sharing time series dataset*

Description

Bicycle sharing time series dataset from the UCI Machine Learning Repository.

Usage

```
data(bsds)
```

Source

https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset

Fanaee-T, Hadi, and Gama, Joao, 'Event labeling combining ensemble detectors and background knowledge', Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg

Examples

```
data(bsds)
head(bsds)
#
```

---

cpp *Subset of growth data from the collaborative perinatal project (CPP)*

Description

Subset of growth data from the collaborative perinatal project (CPP). `cpp_imputed` drops observations for which the `haz` column is `NA`, and imputes all other observations as 0. This is only for the purposes of simplifying testing and examples.

Usage

```
data(cpp)

data(cpp_imputed)
```

Format

A data frame with 1,912 repated-measures observations and 500 unique subjects:

subjid Subject ID

agedays Age since birth at examination (days)

wtkg Weight (kg)

htcm Standing height (cm)

lencm Recumbent length (cm)

bmi BMI (kg/m**2)

waz Weight for age z-score

haz Length/height for age z-score

whz Weight for length/height z-score

baz BMI for age z-score

siteid Investigational Site ID

sexn Sex (num)

sex Sex

feedingn Maternal breastfeeding status (num)

feeding Maternal breastfeeding status

gagebrth Gestational age at birth (days)

birthwt Birth weight (gm)

birthlen Birth length (cm)

apgar1 APGAR Score 1 min after birth

apgar5 APGAR Score 5 min after birth

mage Maternal age at birth of child (yrs)

mracen Maternal race (num)

mrace Maternal race

mmaritn Mothers marital status (num)

mmarit Mothers marital status

meducyrs Mother, years of education

sesn Socio-economic status (num)

ses Socio-economic status

parity Maternal parity

gravida Maternal num pregnancies

smoked Maternal smoking status

mcignum Num cigarettes mom smoked per day

comprisk Maternal risk factors

Source

https://catalog.archives.gov/id/606622

Broman, Sarah. 'The collaborative perinatal project: an overview.' Handbook of longitudinal research 1 (1984): 185-227.

Examples

```
data(cpp)
head(cpp)
#
```

cpp_1yr *Subset of growth data from the collaborative perinatal project (CPP)*

Description

Subset of growth data from the collaborative perinatal project (CPP) at single time-point. The rows in original cpp data were subset for agedays==366. See ?cpp for the description of the variables.

Usage

```
data(cpp_1yr)
```

Source

https://catalog.archives.gov/id/606622

Broman, Sarah. 'The collaborative perinatal project: an overview.' Handbook of longitudinal research 1 (1984): 185-227.

Examples

```
data(cpp_1yr)
head(cpp_1yr)
table(cpp_1yr[["agedays"]])
#
```

Custom_chain *Customize chaining for a learner*

Description

This function wraps a learner in such a way that the behavior of `learner$chain` is modified to use a new function definition. `learner$train` and `learner$predict` are unaffected.

Usage

```
customize_chain(learner, chain_fun)
```

Arguments

`learner`	A `sl3` learner to modify.
`chain_fun`	A function with arguments `learner` and `task` that defines the new chain behavior.

Format

`R6Class` object.

Value

Lrnr_base object with methods for training and prediction

Fields

params A list of learners to chain.

Methods

new(...) This method is used to create a pipeline of learners. Arguments should be individual Learners, in the order they should be applied.

See Also

Other Learners: Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

---

cv_risk *Cross-validated Risk Estimation*

Description

Estimates the cross-validated risk for a given learner and evaluation function, which can be either a loss or a risk function.

Usage

```
cv_risk(learner, eval_fun = NULL, coefs = NULL)
```

Arguments

learner	A trained learner object.
eval_fun	A valid loss or risk function. See loss_functions and risk_functions.
coefs	A numeric vector of coefficients.

cv_sl *Cross-validated Super Learner*

Description

Cross-validated Super Learner

Usage

```
cv_sl(lrnr_sl, eval_fun)
```

Arguments

lrnr_sl	a Lrnr_sl object specifying the Super Learner. Note that the cv_control argument of Lrnr_sl can be specified to control the inner cross-validation of lrnr_sl, as shown in the example.
eval_fun	the evaluation function, either a loss or risk function, for evaluating the Super Learner's predictions.

Value

A list of containing the following: the table of cross-validated risk estimates of the super learner and the candidate learners used to construct it, and either a matrix of coefficients for the super learner on each fold or a list for the metalearner fit on each fold.

Examples

```
## Not run:
data(cpp_imputed)
cpp_task <- sl3_Task$new(
  data = cpp_imputed,
  covariates = c("apgar1", "apgar5", "parity", "gagebrth", "mage"),
  outcome = "haz"
)
glm_lrn <- Lrnr_glm$new()
ranger_lrn <- Lrnr_ranger$new()
lasso_lrn <- Lrnr_glmnet$new()
sl <- Lrnr_sl$new(
  learners = list(glm_lrn, ranger_lrn, lasso_lrn),
  cv_control = list(V = 5),
  verbose = FALSE
)
cv_sl_object <- cv_sl(
  lrnr_sl = sl, eval_fun = loss_squared_error
)

## End(Not run)
```

debug_train *Helper functions to debug sl3 Learners*

Description

Helper functions to debug sl3 Learners

Usage

```
debug_train(learner, once = FALSE)

debugonce_train(learner)

debug_predict(learner, once = FALSE)

debugonce_predict(learner)

sl3_debug_mode(enabled = TRUE)

undebug_learner(learner)
```

Arguments

learner	the learner to debug
once	if true, use debugonce instead of debug
enabled	enable/disable the use of future (debugging is easier without futures)

default_metalearner *Automatically Defined Metalearner*

Description

A sensible metalearner is chosen based on the outcome type.

Usage

```
default_metalearner(outcome_type)
```

Arguments

outcome_type	a Variable_Type object

Details

For binary and continuous outcome types, the default metalearner is non-negative least squares (NNLS) regression (Lrnr_nnls), and for others the metalearner is Lrnr_solnp with an appropriate loss and combination function, shown in the table below.

Outcome Type	Combination Function	Loss Function
categorical	metalearner_linear_multinomial	loss_loglik_multinomial
multivariate	metalearner_linear_multivariate	loss_squared_error_multivariate

---

`define_h2o_X` *h2o Model Definition*

Description

Definition of h2o type models. This function is for internal use only. This function uploads input data into an h2o.Frame, allowing the data to be subset to the task$X data.table by a smaller set of covariates if spec'ed in params.

This learner provides faster fitting procedures for generalized linear models by using the h2o package and the h2o.glm method. The h2o Platform fits GLMs in a computationally efficient manner. For details on the procedure, consult the documentation of the h2o package.

Usage

```
define_h2o_X(task, outcome_type = NULL)
```

Arguments

task	An object of type Lrnr_base as defined in this package.
outcome_type	An object of type Variable_Tyoe for use in formatting the outcome

Format

R6Class object.

Value

Learner object with methods for training and prediction. See Lrnr_base for documentation on learners.

Parameters

`intercept=TRUE` If `TRUE`, and intercept term is included.

`standardize=TRUE` Standardize covariates to have mean = 0 and SD = 1.

`lambda=0` Lasso Parameter.

`max_iterations=100` Maximum number of iterations.

`ignore_const_columns=FALSE` If `TRUE`, drop constant covariate columns

`missing_values_handling="Skip"` How to handle missing values.

`...` Other arguments passed to `h2o.glm`.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `undocumented_learner`

Examples

```
## Not run:
library(h2o)
suppressWarnings(h2o.init())

# load example data
data(cpp_imputed)

# create sl3 task
task <- sl3_Task$new(
```

```
  cpp_imputed,
  covariates = c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs"),
  outcome = "haz"
)

# train h2o glm learner and make predictions
lrnr_h2o <- Lrnr_h2o_glm$new()
lrnr_h2o_fit <- lrnr_h2o$train(task)
lrnr_h2o_pred <- lrnr_h2o_fit$predict()

## End(Not run)
```

`delayed_make_learner` *Learner helpers*

Description

Learner helpers

Usage

```
delayed_make_learner(learner_class, ...)

learner_train(learner, task, trained_sublearners)

delayed_learner_train(learner, task, name = NULL)

learner_fit_predict(learner_fit, task = NULL)

delayed_learner_fit_predict(learner_fit, task = NULL)

learner_fit_chain(learner_fit, task = NULL)

delayed_learner_fit_chain(learner_fit, task = NULL)

learner_subset_covariates(learner, task)

learner_process_formula(learner, task)

delayed_learner_subset_covariates(learner, task)

delayed_learner_process_formula(learner, task)
```

Arguments

`learner_class`	The learner class to instantiate.
`...`	Parameters with which to instantiate the learner.

learner	A learner object to fit to the task.
task	The task on which to fit.
trained_sublearners	Any data obtained from a train_sublearners step.
name	a more detailed name for this delayed task, if necessary
learner_fit	a learner object that has already been fit

---

density_dat *Simulated data with continuous exposure*

Description

Simulated data with continuous exposure, used with examples of conditional density estimation.

Usage

```
data(density_dat)
```

Examples

```
data(density_dat)
head(density_dat)
#
```

---

factor_to_indicators *Convert Factors to indicators*

Description

replicates the functionality of model.matrix, but faster

Replicates the functionality of model.matrix, but faster

Usage

```
factor_to_indicators(x, ind_ref_mat = NULL)

dt_expand_factors(dt)
```

Arguments

x	the factor to expand
ind_ref_mat	a matrix used for expansion, if NULL generated automatically
dt	the dt to expand

`importance` *Importance Extract variable importance measures produced by* `randomForest` *and order in decreasing order of importance.*

Description

Function that takes a cross-validated fit (i.e., cross-validated learner that has already been trained on a task), which could be a cross-validated single learner or super learner, and generates a risk-based variable importance score for either each covariate or each group of covariates in the task. This function outputs a `data.table`, where each row corresponds to the risk difference or the risk ratio between the following two risks: the risk when a covariate (or group of covariates) is permuted or removed, and the original risk (i.e., when all covariates are included as they were in the observed data). A higher risk ratio/difference corresponds to a more important covariate/group. A plot can be generated from the returned `data.table` by calling companion function `importance_plot`.

Usage

```
importance(fit, eval_fun = NULL, fold_number = "validation",
  type = c("remove", "permute"), importance_metric = c("difference",
  "ratio"), covariate_groups = NULL)

importance(fit, eval_fun = NULL, fold_number = "validation",
  type = c("remove", "permute"), importance_metric = c("difference",
  "ratio"), covariate_groups = NULL)
```

Arguments

`fit`	A trained cross-validated (CV) learner (such as a CV stack or super learner), from which cross-validated predictions can be generated.
`eval_fun`	The evaluation function (risk or loss function) for evaluating the risk. Defaults vary based on the outcome type, matching defaults in `default_metalearner`. See `loss_functions` and `risk_functions` for options. Default is `NULL`.
`fold_number`	The fold number to use for obtaining the predictions from the fit. Either a positive integer for obtaining predictions from a specific fold's fit; `"full"` for obtaining predictions from a fit on all of the data, or `"validation"` (default) for obtaining cross-validated predictions, where the data used for training and prediction never overlaps across the folds. Note that if a positive integer or `"full"` is supplied here then there will be overlap between the data used for training and validation, so `fold_number ="validation"` is recommended.
`type`	Which method should be used to obscure the relationship between each covariate / covariate group and the outcome? When `type` is `"remove"` (default), each covariate / covariate group is removed one at a time from the task; the cross-validated learner is refit to this modified task; and finally, predictions are obtained from this refit. When `type` is `"permute"`, each covariate / covariate group is permuted (sampled without replacement) one at a time, and then predictions are obtained from this modified data.

`importance_metric`
: Either `"ratio"` or `"difference"` (default). For each covariate / covariate group, `"ratio"` returns the risk of the permuted/removed covariate / covariate group divided by observed/original risk (i.e., the risk with all covariates as they existed in the sample) and `"difference"` returns the difference between the risk with the permuted/removed covariate / covariate group and the observed risk.

`covariate_groups`
: Optional named list covariate groups which will invoke variable importance evaluation at the group-level, by removing/permuting all covariates in the same group together. If covariates in the task are not specified in the list of groups, then those covariates will be added as additional single-covariate groups.

Value

A `data.table` of variable importance for each covariate.

Examples

```
## Not run:
# define ML task
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# build relatively fast learner library (not recommended for real analysis)
lasso_lrnr <- Lrnr_glmnet$new()
glm_lrnr <- Lrnr_glm$new()
ranger_lrnr <- Lrnr_ranger$new()
lrnrs <- c(lasso_lrnr, glm_lrnr)
names(lrnrs) <- c("lasso", "glm")
lrnr_stack <- make_learner(Stack, lrnrs)

# instantiate SL with default metalearner
sl <- Lrnr_sl$new(lrnr_stack)
sl_fit <- sl$train(task)

importance_result <- importance(sl_fit)
importance_result

# importance with groups of covariates
groups <- list(
  scores = c("apgar1", "apgar5"),
  maternal = c("parity", "mage", "meducyrs")
)
importance_result_groups <- importance(sl_fit, covariate_groups = groups)
importance_result_groups

## End(Not run)
```

`importance_plot` *Variable Importance Plot*

Description

Variable Importance Plot

Usage

```
importance_plot(x, nvar = min(30, nrow(x)))
```

Arguments

`x`	The two-column `data.table` returned by `importance`, where the first column is the covariate/groups and the second column is the importance score.
`nvar`	The maximum number of predictors to be plotted. Defaults to the minimum between 30 and the number of rows in `x`.

Value

A `ggplot` of variable importance.

Examples

```
## Not run:
# define ML task
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# build relatively fast learner library (not recommended for real analysis)
lasso_lrnr <- Lrnr_glmnet$new()
glm_lrnr <- Lrnr_glm$new()
ranger_lrnr <- Lrnr_ranger$new()
lrnrs <- c(lasso_lrnr, glm_lrnr)
names(lrnrs) <- c("lasso", "glm")
lrnr_stack <- make_learner(Stack, lrnrs)

# instantiate SL with default metalearner
sl <- Lrnr_sl$new(lrnr_stack)
sl_fit <- sl$train(task)
importance_result <- importance(sl_fit)
importance_plot(importance_result)

## End(Not run)
```

inverse_sample *Inverse CDF Sampling*

Description

Inverse CDF Sampling

Usage

```
inverse_sample(n_samples, cdf = NULL, pdf = NULL)
```

Arguments

n_samples	If true, remove entries after failure time for each observation.
cdf	A list with x and y representing the cdf
pdf	A list with x and y representing the pdf

loss_functions *Loss Function Definitions*

Description

Loss functions for use in evaluating learner fits.

Usage

```
loss_squared_error(pred, observed)

loss_loglik_true_cat(pred, observed)

loss_loglik_binomial(pred, observed)

loss_loglik_multinomial(pred, observed)

loss_squared_error_multivariate(pred, observed)
```

Arguments

pred	A vector of predicted values
observed	A vector of observed values

Value

A vector of loss values

Note

Assumes predicted probabilities are "packed" into a single vector.

`Lrnr_arima` *Univariate ARIMA Models*

Description

This learner supports autoregressive integrated moving average model for univariate time-series.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

- `order`: An optional specification of the non-seasonal part of the ARIMA model; the three integer components (p, d, q) are the AR order, the degree of differencing, and the MA order. If order is specified, then `arima` will be called; otherwise, `auto.arima` will be used to fit the "best" ARIMA model according to AIC (default), AIC or BIC. The information criterion to be used in `auto.arima` model selection can be modified by specifying `ic` argument.
- `num_screen = 5`: The top n number of "most impotant" variables to retain.
- `...`: Other parameters passed to `arima` or `auto.arima` function, depending on whether or not `order` argument is provided.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
library(origami)
data(bsds)

folds <- make_folds(bsds,
  fold_fun = folds_rolling_window, window_size = 500,
  validation_size = 100, gap = 0, batch = 50
)

task <- sl3_Task$new(
  data = bsds,
  folds = folds,
  covariates = c(
    "weekday", "temp"
  ),
  outcome = "cnt"
)

arima_lrnr <- make_learner(Lrnr_arima)

train_task <- training(task, fold = task$folds[[1]])
valid_task <- validation(task, fold = task$folds[[1]])

arima_fit <- arima_lrnr$train(train_task)
arima_preds <- arima_fit$predict(valid_task)
```

Lrnr_bartMachine *bartMachine: Bayesian Additive Regression Trees (BART)*

Description

This learner implements Bayesian Additive Regression Trees via **bartMachine** (described in Kapelner and Bleich (2016)) and the function `bartMachine`.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- . . . : Parameters passed to `bartMachine`. See it's documentation for details.

References

Kapelner A, Bleich J (2016). "bartMachine: Machine Learning with Bayesian Additive Regression Trees." *Journal of Statistical Software*, **70**(4), 1–40. doi:10.18637/jss.v070.i04.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
# set up ML task
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# fit a bartMachine model and predict from it
bartMachine_learner <- make_learner(Lrnr_bartMachine)
bartMachine_fit <- bartMachine_learner$train(task)
preds <- bartMachine_fit$predict()

## End(Not run)
```

Lrnr_base *Base Class for all sl3 Learners*

Description

Generally this base learner class should not be instantiated. Intended to be an abstract class, although abstract classes are not explicitly supported by **R6**. All learners support the methods and fields documented below. For more information on a particular learner, see its help file.

Usage

```
make_learner(learner_class, ...)
```

Arguments

`learner_class`	The learner class to instantiate.
`...`	Parameters with which to instantiate the learner. See Parameters section below.

Format

`R6Class` object.

Value

Learner object with methods for training and prediction

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

User Methods

`train(task)` Trains learner to a task using `delayed`. Returns a fit object

- `task`: The task to use for training

`base_train(task, trained_sublearners = NULL)` Trains learner to a task. Returns a fit object

- `task`: The task to use for training
- `trained_sublearners`: Any sublearners previous trained. Almost always `NULL`

`predict(task=NULL)` Generates predictions using `delayed`. Returns a prediction vector or matrix.

- `task`: The task to use for prediction. If no task is provided, it will use the task used for training.

`base_predict(task=NULL)` Generates predictions. Returns a prediction vector or matrix.

- `task`: The task to use for prediction. If no task is provided, it will use the task used for training.

`chain(task=NULL)` Generates a chained task using `delayed`

- `task`: The task to use for chaining If no task is provided, it will use the task used for training.

`base_chain(task=NULL)` Generates a chained task

- `task`: The task to use for chaining If no task is provided, it will use the task used for training.

Fields

`is_trained` TRUE if this is a learner fit, not an untrained learner

`fit_object` The internal fit object

`name` The learner name

`learner_uuid` A unique identifier of this learner, but common to all fits of this learner

`fit_uuid` A unique identifier of this learner fit. NULL if this is an untrained learner

`params` A list of learner parameters, as specified on construction

`training_task` The task used for training. NULL if this is an untrained learner

`training_outcome_type` The outcome_type of the task used for training. NULL if this is an untrained learner

`properties` The properties supported by this learner

`coefficients` Fit coefficients, if this learner has coefficients. NULL otherwise, or if this is an untrained learner

Internal Methods

These methods are primiarily for internal use only. They're not recommended for public consumption.

`subset_covariates(task)` Returns a task with covariates subsetted using the `covariates` parameter.

- `task`: The task to subset

`get_outcome_type(task)` Mediates between the task outcome_type and parameter outcome_type. If a parameter outcome_type was specified, returns that. Otherwise, returns the task$outcome_type.

- `task`: The task for which to determine the outcome_type

`train_sublearners(task)` Trains sublearners to a task using `delayed`. Returns a delayed sublearner fit.

- `task`: The task to use for training

`set_train(fit_object, training_task)` Converts a learner to a learner fit.

- `fit_object`: The fit object generated by a call to `private$.train`
- `training_task`: The task used for training

`assert_trained()` Throws an error if this learner does not have a `fit_object`

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task,

Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

---

`Lrnr_bayesglm` *Bayesian Generalized Linear Models*

Description

This learner provides fitting procedures for bayesian generalized linear models (GLMs) from **ar** using `bayesglm.fit`. The GLMs fitted in this way can incorporate independent normal, t, or Cauchy prior distribution for the coefficients.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `intercept = TRUE`: A `logical` specifying whether an intercept term should be included in the fitted null model.
- `...`: Other parameters passed to `bayesglm.fit`. See it's documentation for details.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(cpp_imputed)
covars <- c(
  "apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs", "sexn"
)
outcome <- "haz"
task <- sl3_Task$new(cpp_imputed,
  covariates = covars,
  outcome = outcome
)
# fit and predict from a bayesian GLM
bayesglm_lrnr <- make_learner(Lrnr_bayesglm)
bayesglm_fit <- bayesglm_lrnr$train(task)
bayesglm_preds <- bayesglm_fit$predict(task)
```

Lrnr_bound *Bound Predictions*

Description

This learner bounds predictions. Intended for use as part of Pipeline.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- bound: Either a vector of length two, with lower and upper bounds, or a vector of length 1 with a lower bound, and the upper will be set symmetrically as 1 - the lower bound. Both bounds must be provided when the variable type of the task's outcome is continuous.

Examples

```
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

lasso_lrnr <- Lrnr_glmnet$new()
glm_lrnr <- Lrnr_glm$new()
lrnr_stack <- make_learner(Stack, lasso_lrnr, glm_lrnr)
lrnr_bound <- Lrnr_bound$new(c(-2, 2))
stack_bounded_preds <- Pipeline$new(lrnr_stack, lrnr_bound)
metalrnr_discrete_MSE <- Lrnr_cv_selector$new(loss_squared_error)
discrete_sl <- Lrnr_sl$new(
  learners = stack_bounded_preds, metalearner = metalrnr_discrete_MSE
)
discrete_sl_fit <- discrete_sl$train(task)
```

```
preds <- discrete_sl_fit$predict()
range(preds)
```

`Lrnr_caret` *Caret (Classification and Regression) Training*

Description

This learner uses the **caret** package's `train` function to automatically tune a predictive model. It does this by defining a grid of model-specific tuning parameters; fitting the model according to each tuning parameter specification, to establish a set of models fits; calculating a resampling-based performance measure each variation; and then selecting the model with the best performance.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `method`: A string specifying which **caret** classification or regression model to use. Possible models can be found using `names(caret::getModelInfo())`. Information about a model, including the parameters that are tuned, can be found using `caret::modelLookup()`, e.g., `caret::modelLookup("xgbLinear")`. Consult the `caret` package's documentation on `train` for more details.
- `metric = NULL`: An optional string specifying the summary metric to be used to select the optimal model. If not specified, it will be set to "RMSE" for continuous outcomes and "Accuracy" for categorical and binary outcomes. Other options include "MAE", "Kappa", "Rsquared" and "logLoss". Regression models are defined when `metric` is set as "RMSE", "logLoss", "Rsquared", or "MAE". Classification models are defined when `metric` is set as "Accuracy" or "Kappa". Custom performance metrics can also be used. Consult the `caret` package's `train` documentation for more details.
- `trControl = list(method = "cv", number = 10)`: A list for specifying the arguments for `trainControl` object. If not specified, it will consider "cv" with 10 folds as the resampling method, instead of `caret`'s default resampling method, "boot". For a detailed description, consult the `caret` package's documentation for `train` and `trainControl`.
- `factor_binary_outcome = TRUE`: Logical indicating whether a binary outcome should be defined as a factor instead of a numeric. This only needs to be modified to `FALSE` in the following uncommon instance: when `metric` is specified by the user, `metric` defines a regression model, and the task's outcome is binary. Note that `train` could throw warnings/errors when regression models are considered for binary outcomes; this argument should only be modified by advanced users in niche settings.
- `...`: Other parameters passed to `train` and additional arguments defined in `Lrnr_base`, such as `params` like `formula`.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")
autotuned_RF_lrnr <- Lrnr_caret$new(method = "rf")
set.seed(693)
autotuned_RF_fit <- autotuned_RF_lrnr$train(task)
autotuned_RF_predictions <- autotuned_RF_fit$predict()

## End(Not run)
```

Lrnr_cv *Fit/Predict a learner with Cross Validation*

Description

A wrapper around any learner that generates cross-validate predictions

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`learner` The learner to wrap

`folds=NULL` An `origami` folds object. If `NULL`, folds from the task are used

`full_fit=FALSE` If `TRUE`, also fit the underlying learner on the full data. This can then be accessed with predict_fold(task, fold_number="full")

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
library(origami)

# load example data
data(cpp_imputed)
covars <- c(
  "apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs", "sexn"
)
outcome <- "haz"

# create sl3 task
task <- sl3_Task$new(cpp_imputed, covariates = covars, outcome = outcome)
glm_learner <- Lrnr_glm$new()
cv_glm <- Lrnr_cv$new(glm_learner, folds = make_folds(cpp_imputed, V = 10))

# train cv learner
cv_glm_fit <- cv_glm$train(task)
preds <- cv_glm_fit$predict()
```

Lrnr_cv_selector *Cross-Validated Selector*

Description

This learner is the cross-validated (CV) selector, and it is intended for use as the `metalearner` in Lrnr_sl. `Lrnr_cv_selector` selects the candidate with the best CV predictive performance (i.e., lowest CV risk). Specifically, it aims to optimize the CV risk, and it is defined by a constrained weighted combination: the weights can either be zero or one, and they must sum to one. `Lrnr_cv_selector` optimizes the CV predictive performance under these constraints by assigning the candidate with the best CV predictive performance a weight of one and all others a weight of zero. Thus, `Lrnr_cv_selector` and its predictions will be identical to the best-performing candidate learner and its predictions; this is why we say `Lrnr_cv_selector` "selects" the candidate with the best CV predictive performance.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- eval_function = loss_squared_error: A function that takes as input a vector of predicted values as its first argument and a vector of observed outcome values as its second argument, and then returns a vector of losses or a numeric risk. See loss_functions and risk_functions for options.
- folds = NULL: Optional **origami**-structured cross-validation folds from the task for training Lrnr_sl, e.g., task$folds. This argument is only required and utilized when eval_function is not a loss function, since the risk has to be calculated on each validation set separately and then averaged across them in order to estimate the cross-validated risk. This argument is ignored when eval_function is a loss.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

hal_lrnr <- Lrnr_hal9001$new(
  max_degree = 1, num_knots = c(20, 10), smoothness_orders = 0
)
lasso_lrnr <- Lrnr_glmnet$new()
glm_lrnr <- Lrnr_glm$new()
ranger_lrnr <- Lrnr_ranger$new()
lrnrs <- c(hal_lrnr, lasso_lrnr, glm_lrnr, ranger_lrnr)
names(lrnrs) <- c("hal", "lasso", "glm", "ranger")
```

```
lrnr_stack <- make_learner(Stack, lrnrs)
metalrnr_discrete_MSE <- Lrnr_cv_selector$new(loss_squared_error)
discrete_sl <- Lrnr_sl$new(
  learners = lrnr_stack, metalearner = metalrnr_discrete_MSE
)
discrete_sl_fit <- discrete_sl$train(task)
discrete_sl_fit$cv_risk(loss_squared_error)

## End(Not run)
```

`Lrnr_dbarts` *Discrete Bayesian Additive Regression Tree sampler*

Description

This learner implements BART algorithm in C++, using the `dbarts` package. BART is a Bayesian sum-of-trees model in which each tree is constrained by a prior to be a weak learner.

Format

`R6Class` object.

Value

Learner object with methods for training and prediction. See `Lrnr_base` for documentation on learners.

Parameters

`x.test` Explanatory variables for test (out of sample) data. `bart` will generate draws of $f(x)$ for each x which is a row of `x.test`.

`sigest` For continuous response models, an estimate of the error variance, σ^2, used to calibrate an inverse-chi-squared prior used on that parameter. If not supplied, the least-squares estimate is derived instead. See `sigquant` for more information. Not applicable when y is binary.

`sigdf` Degrees of freedom for error variance prior. Not applicable when y is binary.

`sigquant` The quantile of the error variance prior that the rough estimate (`sigest`) is placed at. The closer the quantile is to 1, the more aggresive the fit will be as you are putting more prior weight on error standard deviations (σ) less than the rough estimate. Not applicable when y is binary.

`k` For numeric y, `k` is the number of prior standard deviations $E(Y|x) = f(x)$ is away from ± 0.5. The response (`y.train`) is internally scaled to range from -0.5 to 0.5. For binary y, `k` is the number of prior standard deviations $f(x)$ is away from ± 3. In both cases, the bigger k is, the more conservative the fitting will be.

`power` Power parameter for tree prior.

`base` Base parameter for tree prior.

`binaryOffset` sed for binary y. When present, the model is $P(Y = 1 \mid x) = \Phi(f(x) + \text{binaryOffset})$, allowing fits with probabilities shrunk towards values other than 0.5.

`weights` An optional vector of weights to be used in the fitting process. When present, BART fits a model with observations $y \mid x \sim N(f(x), \sigma^2/w)$, where $f(x)$ is the unknown function.

`ntree` The number of trees in the sum-of-trees formulation.

`ndpost` The number of posterior draws after burn in, `ndpost / keepevery` will actually be returned.

`nskip` Number of MCMC iterations to be treated as burn in.

`printevery` As the MCMC runs, a message is printed every `printevery` draws.

`keepevery` Every `keepevery` draw is kept to be returned to the user. Useful for "thinning" samples.

`keeptrainfits` If `TRUE` the draws of $f(x)$ for x corresponding to the rows of `x.train` are returned.

`usequants` When `TRUE`, determine tree decision rules using estimated quantiles derived from the `x.train` variables. When `FALSE`, splits are determined using values equally spaced across the range of a variable. See details for more information.

`numcut` The maximum number of possible values used in decision rules (see `usequants`, details). If a single number, it is recycled for all variables; otherwise must be a vector of length equal to `ncol(x.train)`. Fewer rules may be used if a covariate lacks enough unique values.

`printcutoffs` The number of cutoff rules to printed to screen before the MCMC is run. Given a single integer, the same value will be used for all variables. If 0, nothing is printed.

`verbose` Logical; if `FALSE` supress printing.

`nchain` Integer specifying how many independent tree sets and fits should be calculated.

`nthread` Integer specifying how many threads to use. Depending on the CPU architecture, using more than the number of chains can degrade performance for small/medium data sets. As such some calculations may be executed single threaded regardless.

`combinechains` Logical; if `TRUE`, samples will be returned in arrays of dimensions equal to `nchain` $\times$ `ndpost` $\times$ number of observations.

`keeptrees` Logical; must be `TRUE` in order to use `predict` with the result of a `bart` fit.

`keepcall` Logical; if `FALSE`, returned object will have `call` set to `call("NULL")`, otherwise the call used to instantiate BART.

`serializeable` Logical; if `TRUE`, loads the trees into R memory so the fit object can be saved and loaded. See the section on "Saving" in `bart` `NB: This is not currently working`

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
set.seed(123)

# load example data
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")

# create sl3 task
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")
dbart_learner <- make_learner(Lrnr_dbarts, ndpost = 200)

# train dbart learner and make predictions
dbart_fit <- dbart_learner$train(task)
preds <- dbart_fit$predict()
```

Lrnr_define_interactions

Define interactions terms

Description

This learner adds interactions to its chained task. Intended for use in a Pipeline, defining a coupling of the interactions with the learner.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- `interactions`: A list whose elements are a character vector of covariates from which to create interaction terms.
- `warn_on_existing`: If `TRUE`, produce a warning if there is already a column with a name matching this given interaction term.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(cpp_imputed)
covars <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs", "sexn")
outcome <- "haz"
task <- sl3_Task$new(cpp_imputed, covariates = covars, outcome = outcome)
interactions <- list(c("apgar1", "parity"), c("apgar5", "parity"))
lrnr_interact <- Lrnr_define_interactions$new(
  list(c("apgar1", "parity"), c("apgar5", "parity"))
)
lrnr_glm <- Lrnr_glm$new()
interaction_pipeline_glm <- make_learner(Pipeline, lrnr_interact, lrnr_glm)
fit <- interaction_pipeline_glm$train(task)
```

Lrnr_density_discretize

Density from Classification

Description

This learner discretizes a continuous density and then fits a categorical learner

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

categorical_learner The learner to wrap.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by Lrnr_base, and shared by all learners.

covariates A character vector of covariates. The learner will use this to subset the covariates for any specified task

outcome_type A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

... All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
# load example data
data(cpp_imputed)

# create sl3 task
task <- sl3_Task$new(
  cpp_imputed,
  covariates = c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs"),
  outcome = "haz"
)

# train density discretize learner and make predictions
lrnr_discretize <- Lrnr_density_discretize$new(
  categorical_learner = Lrnr_glmnet$new()
)
```

```
lrnr_discretize_fit <- lrnr_discretize$train(task)
lrnr_discretize_pred <- lrnr_discretize_fit$predict()
```

`Lrnr_density_hse` *Density Estimation With Mean Model and Homoscedastic Errors*

Description

This learner assumes a mean model with homoscedastic errors: Y ~ E(Y|W) + epsilon. E(Y|W) is fit using any mean learner, and then the errors are fit with kernel density estimation.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`binomial_learner` The learner to wrap.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
# load example data
data(cpp_imputed)

# create sl3 task
task <- sl3_Task$new(
  cpp_imputed,
  covariates = c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs"),
  outcome = "haz"
)

# train density hse learner and make predictions
lrnr_density_hse <- Lrnr_density_hse$new(mean_learner = Lrnr_glm$new())
fit_density_hse <- lrnr_density_hse$train(task)
preds_density_hse <- fit_density_hse$predict()
```

Lrnr_density_semiparametric

Density Estimation With Mean Model and Homoscedastic Errors

Description

This learner assumes a mean model with homoscedastic errors: Y ~ E(Y|W) + epsilon. E(Y|W) is fit using any mean learner, and then the errors are fit with kernel density estimation.

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`binomial_learner` The learner to wrap.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
# load example data
data(cpp_imputed)

# create sl3 task
task <- sl3_Task$new(
  cpp_imputed,
  covariates = c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs"),
  outcome = "haz"
)

# train density hse learner and make predictions
lrnr_density_semi <- Lrnr_density_semiparametric$new(
  mean_learner = Lrnr_glm$new()
)
lrnr_density_semi_fit <- lrnr_density_semi$train(task)
lrnr_density_semi_pred <- lrnr_density_semi_fit$predict()
```

Lrnr_earth *Earth: Multivariate Adaptive Regression Splines*

Description

This learner provides fitting procedures for building regression models thru the spline regression techniques described in Friedman (1991) and Friedman (1993), via **earth** and the function earth.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- `degree`: A `numeric` specifying the maximum degree of interactions to be used in the model. This defaults to 2, specifying up through one-way interaction terms. Note that this differs from the default of `earth`.
- `penalty`: Generalized Cross Validation (GCV) penalty per knot. Defaults to 3 as per the recommendation for `degree` > 1 in the documentation of `earth`. Special values (for use by knowledgeable users): The value 0 penalizes only terms, not knots. The value -1 translates to no penalty.
- `pmethod`: Pruning method, defaulting to `"backward"`. Other options include `"none"`, `"exhaustive"`, `"forward"`, `"seqrep"`, `"cv"`.
- `nfold`: Number of cross-validation folds. The default is 0, for no cross-validation.
- `ncross`: Only applies if `nfold` > 1, indicating the number of cross-validation rounds. Each cross-validation has `nfold` folds. Defaults to 1.
- `minspan`: Minimum number of observations between knots.
- `endspan`: Minimum number of observations before the first and after the final knot.
- `...`: Other parameters passed to `earth`. See its documentation for details.

References

Friedman JH (1991). "Multivariate adaptive regression splines." *The Annals of Statistics*, 1–67.

Friedman JH (1993). "Fast MARS." Stanford University. https://doi.org/10.1214/aos/1176347963.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
data(cpp_imputed)
covars <- c(
  "apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs", "sexn"
)
outcome <- "haz"
task <- sl3_Task$new(cpp_imputed,
  covariates = covars,
```

```
  outcome = outcome
)
# fit and predict from a MARS model
earth_lrnr <- make_learner(Lrnr_earth)
earth_fit <- earth_lrnr$train(task)
earth_preds <- earth_fit$predict(task)
```

`Lrnr_expSmooth` *Exponential Smoothing state space model*

Description

This learner supports exponential smoothing models using `ets`.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

- `model="ZZZ"`: Three-character string identifying method. In all cases, "N"=none, "A"=additive, "M"=multiplicative, and "Z"=automatically selected. The first letter denotes the error type, second letter denotes the trend type, third letter denotes the season type. For example, "ANN" is simple exponential smoothing with additive errors, "MAM" is multiplicative Holt-Winters' methods with multiplicative errors, etc.
- `damped=NULL`: If TRUE, use a damped trend (either additive or multiplicative). If NULL, both damped and non-damped trends will be tried and the best model (according to the information criterion ic) returned.
- `alpha=NULL`: Value of alpha. If NULL, it is estimated.
- `beta=NULL`: Value of beta. If NULL, it is estimated.
- `gamma=NULL`: Value of gamma. If NULL, it is estimated.
- `phi=NULL`: Value of phi. If NULL, it is estimated.
- `lambda=NULL`: Box-Cox transformation parameter. Ignored if `NULL`. When lambda is specified, `additive.only` is set to `TRUE`.
- `additive.only=FALSE`: If `TRUE`, will only consider additive models.
- `biasadj=FALSE`: Use adjusted back-transformed mean for Box-Cox transformations.
- `lower=c(rep(1e-04, 3), 0.8)`: Lower bounds for the parameters (alpha, beta, gamma, phi).
- `upper=c(rep(0.9999,3), 0.98)`: Upper bounds for the parameters (alpha, beta, gamma, phi)
- `opt.crit="lik"`: Optimization criterion.

- `nmse=3`: Number of steps for average multistep MSE (1 <= nmse <= 30).
- `bounds="both""` Type of parameter space to impose: "usual" indicates all parameters must lie between specified lower and upper bounds; "admissible" indicates parameters must lie in the admissible space; "both" (default) takes the intersection of these regions.
- `ic="aic"`: Information criterion to be used in model selection.
- `restrict=TRUE`: If TRUE, models with infinite variance will not be allowed.
- `allow.multiplicative.trend=FALSE`: If TRUE, models with multiplicative trend are allowed when searching for a model.
- `use.initial.values=FALSE`: If `TRUE` and model is of class "ets", then the initial values in the model are also not re-estimated.
- `n.ahead`: The forecast horizon. If not specified, returns forecast of size `task$X`.
- `freq=1`: the number of observations per unit of time.
- `...`: Other parameters passed to `ets`.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
library(origami)
data(bsds)

folds <- make_folds(bsds,
  fold_fun = folds_rolling_window, window_size = 500,
  validation_size = 100, gap = 0, batch = 50
)

task <- sl3_Task$new(
  data = bsds,
  folds = folds,
  covariates = c(
    "weekday", "temp"
  ),
  outcome = "cnt"
)
```

```
expSmooth_lrnr <- make_learner(Lrnr_expSmooth)

train_task <- training(task, fold = task$folds[[1]])
valid_task <- validation(task, fold = task$folds[[1]])

expSmooth_fit <- expSmooth_lrnr$train(train_task)
expSmooth_preds <- expSmooth_fit$predict(valid_task)
```

`Lrnr_ga` *Nonlinear Optimization via Genetic Algorithm (GA)*

Description

This metalearner provides fitting procedures for any pairing of loss or risk function and metalearner function, subject to constraints. The optimization problem is solved by making use of the `ga` function in the **GA** R package. For further consult the documentation of this package.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `learner_function` = `metalearner_linear`: A function(alpha, X) that takes a vector of covariates and a matrix of data and combines them into a vector of predictions. See `metalearners` for options.
- `eval_function` = `loss_squared_error`: A function(pred, truth) that takes prediction and truth vectors and returns a loss vector or a risk scalar. See `loss_functions` and `risk_functions` for options and more detail.
- `make_sparse` = `TRUE`: If `TRUE`, zeros out small alpha values.
- `convex_combination` = `TRUE`: If `TRUE`, constrain alpha to sum to 1.
- `maxiter` = `100`: The maximum number of iterations to run before the GA search is halted.
- `run` = `10`: The number of consecutive generations without any improvement in the best fitness value before the GA is stopped.
- `optim` = `TRUE`: A logical determining whether or not a local search using general-purpose optimization algorithms should be used. Argument `optimArgs` of `ga` provides further details and finer control.
- `...`: Additional arguments to `ga` and/or `Lrnr_base`.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
# define ML task
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# build relatively fast learner library (not recommended for real analysis)
lasso_lrnr <- Lrnr_glmnet$new()
glm_lrnr <- Lrnr_glm$new()
lrnrs <- c(lasso_lrnr, glm_lrnr)
names(lrnrs) <- c("lasso", "glm")
lrnr_stack <- make_learner(Stack, lrnrs)

# instantiate SL with GA metalearner
ga <- Lrnr_ga$new(maxiter=10)
sl <- Lrnr_sl$new(lrnr_stack, ga)
sl_fit <- sl$train(task)
```

Lrnr_gam *GAM: Generalized Additive Models*

Description

This learner provides fitting procedures for generalized additive models, using the routines from **mgcv** through a call to the function gam. The **mgcv** package and the use of GAMs are described thoroughly (with examples) in Wood (2017), while Hastie and Tibshirani (1990) also provided an earlier quite thorough look at GAMs.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- formula: An optional argument specifying the formula of GAM. Input type can be formula or string, or a list of them. If not specified, continuous covariates will be smoothened with the smooth terms represented using "penalized thin plate regression splines". For a more detailed description, please consult the documentation for gam.
- family: An optional argument specifying the family of the GAM. See family and family.mgcv for a list of available family functions. If left unspecified, it will be inferred depending on the detected type of the outcome. For now, GAM supports binomial and gaussian outcome types, if formula is unspecified. For a more detailed description of this argument, please consult the documentation of gam.
- method: An optional argument specifying the method for smoothing parameter selection. The default is global cross-validation (GCV). For more detaileds on this argument, consult the documentation of gam.
- ...: Other parameters passed to gam. See its documentation for details.

References

Hastie TJ, Tibshirani RJ (1990). *Generalized additive models*, volume 43. CRC press.

Wood SN (2017). *Generalized additive models: an introduction with R*. CRC press.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(cpp_imputed)
# create task for prediction
cpp_task <- sl3_Task$new(
  data = cpp_imputed,
  covariates = c("bmi", "parity", "mage", "sexn"),
```

```
  outcome = "haz"
)
# initialization, training, and prediction with the defaults
gam_lrnr <- Lrnr_gam$new()
gam_fit <- gam_lrnr$train(cpp_task)
gam_preds <- gam_fit$predict()
```

`Lrnr_gbm` *GBM: Generalized Boosted Regression Models*

Description

This learner provides fitting procedures for generalized boosted regression trees, using the routines from **gbm**, through a call to the function `gbm.fit`. Though a variety of gradient boosting strategies have seen popularity in machine learning, a few of the early methodological descriptions were given by Friedman (2001) and Friedman (2002).

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `n.trees`: An integer specifying the total number of trees to fit. This is equivalent to the number of iterations and the number of basis functions in the additive expansion. The default is 10000.
- `interaction.depth`: An integer specifying the maximum depth of each tree (i.e., the highest level of allowed variable interactions). A value of 1 implies an additive model, while a value of 2 implies a model with up to 2-way interactions, etc. The default is 2.
- `shrinkage`: A shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction; values of 0.001 to 0.1 have been found to usually work, but a smaller learning rate typically requires more trees. The default is 0.001.
- `...`: Other parameters passed to `gbm`. See its documentation for details.

References

Friedman JH (2001). "Greedy function approximation: a gradient boosting machine." *Annals of statistics*, 1189–1232.

Friedman JH (2002). "Stochastic gradient boosting." *Computational statistics & data analysis*, **38**(4), 367–378.

See Also

Lrnr_xgboost for the extreme gradient boosted tree models from the Xgboost framework (via the **xgboost** package) and Lrnr_lightgbm for the faster and more efficient gradient boosted trees from the LightGBM framework (via the **lightgbm** package).

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(cpp_imputed)
# create task for prediction
cpp_task <- sl3_Task$new(
  data = cpp_imputed,
  covariates = c("apgar1", "apgar5", "parity", "gagebrth", "mage", "sexn"),
  outcome = "haz"
)

# initialization, training, and prediction with the defaults
gbm_lrnr <- Lrnr_gbm$new()
gbm_fit <- gbm_lrnr$train(cpp_task)
gbm_preds <- gbm_fit$predict()
```

Lrnr_glm *Generalized Linear Models*

Description

This learner provides fitting procedures for generalized linear models using the **stats** package glm.fit function.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- `intercept = TRUE`: Should an intercept be included in the model?
- `...`: Other parameters passed to `glm` or `glm.fit`.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# simple, main-terms GLM
lrnr_glm <- make_learner(Lrnr_glm)
glm_fit <- lrnr_glm$train(task)
glm_preds <- glm_fit$predict()

# We can include interaction terms by 'piping' them into this learner.
# Note that both main terms and the specified interactions will be included
# in the regression model.
interaction <- list(c("apgar1", "parity"))
lrnr_interaction <- Lrnr_define_interactions$new(interactions = interaction)
lrnr_glm_w_interaction <- make_learner(Pipeline, lrnr_interaction, lrnr_glm)
fit <- lrnr_glm_w_interaction$train(task)
coefs <- coef(fit$learner_fits$Lrnr_glm_TRUE)
```

`Lrnr_glmnet` *GLMs with Elastic Net Regularization*

Description

This learner provides fitting procedures for elastic net models, including both lasso (L1) and ridge (L2) penalized regression, using the **glmnet** package. The function `cv.glmnet` is used to select an appropriate value of the regularization parameter lambda. For details on these regularized regression models and **glmnet**, consider consulting Friedman et al. (2010)).

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- lambda = NULL: An optional vector of lambda values to compare.
- type.measure = "deviance": The loss to use when selecting lambda. Options documented in cv.glmnet.
- nfolds = 10: Number of k-fold/V-fold cross-validation folds for cv.glmnet to consider when selecting the optimal lambda with cross-validation. Smallest nfolds value allowed by glmnet is 3. For further details, consult the documentation of cv.glmnet.
- alpha = 1: The elastic net parameter: alpha = 0 is Ridge (L2-penalized) regression, while alpha = 1 specifies Lasso (L1-penalized) regression. Values in the closed unit interval specify a weighted combination of the two penalties. For further details, consult the documentation of glmnet.
- nlambda = 100: The number of lambda values to fit. Comparing fewer values will speed up computation, but may hurt the statistical performance. For further details, consult the documentation of cv.glmnet.
- use_min = TRUE: If TRUE, the smallest value of the lambda regularization parameter is used for prediction (i.e., lambda = cv_fit$lambda.min); otherwise, a larger value is used (i.e., lambda = cv_fit$lambda.1se). The distinction between the two variants is clarified in the documentation of cv.glmnet.
- nfolds = 10: Number of folds (default is 10). Smallest value allowable by glmnet is 3.
- ...: Other parameters passed to cv.glmnet and glmnet, and additional arguments defined in Lrnr_base, such as params like formula.

References

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

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation,

Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(mtcars)
mtcars_task <- sl3_Task$new(
  data = mtcars,
  covariates = c(
    "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am",
    "gear", "carb"
  ),
  outcome = "mpg"
)
# simple prediction with lasso penalty
lasso_lrnr <- Lrnr_glmnet$new()
lasso_fit <- lasso_lrnr$train(mtcars_task)
lasso_preds <- lasso_fit$predict()

# simple prediction with ridge penalty
ridge_lrnr <- Lrnr_glmnet$new(alpha = 0)
ridge_fit <- ridge_lrnr$train(mtcars_task)
ridge_preds <- ridge_fit$predict()
```

Lrnr_glmtree *Generalized Linear Model Trees*

Description

This learner uses glmtree from **partykit** to fit recursive partitioning and regression trees in a generalized linear model.

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

- formula: An optional object of class formula (or one that can be coerced to that class), which a symbolic description of the generalized linear model to be fit. If not specified a main terms regression model will be supplied, with each covariate included as a term. Please consult glmtree documentation for more information on its use of formula, and for a description on formula syntax consult the details of the glm documentation.
- ...: Other parameters passed to mob_control or glmtree that are not already specified in the sl3_Task. See its documentation for details.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(cpp_imputed)
# create task for prediction
cpp_task <- sl3_Task$new(
  data = cpp_imputed,
  covariates = c("bmi", "parity", "mage", "sexn"),
  outcome = "haz"
)
# initialization, training, and prediction with the defaults
glmtree_lrnr <- Lrnr_glmtree$new()
glmtree_fit <- glmtree_lrnr$train(cpp_task)
glmtree_preds <- glmtree_fit$predict()
```

Lrnr_glm_fast *Computationally Efficient Generalized Linear Model (GLM) Fitting*

Description

This learner provides faster procedures for fitting linear and generalized linear models than Lrnr_glm with a minimal memory footprint. This learner uses the internal fitting function provided by **speedglm** package, speedglm.wfit. See Enea (2009) for more detail. The glm.fit function is used as a fallback, if speedglm.wfit fails.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- `intercept = TRUE`: Should an intercept be included in the model?
- `method = "Cholesky"`: The method to check for singularity.
- `...`: Other parameters to be passed to `speedglm.wfit`.

References

Enea M (2009). "Fitting linear models and generalized linear models with large data sets in R." *Statistical Methods for the Analysis of Large Datasets: book of short papers*, 411–414.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# simple, main-terms GLM
lrnr_glm_fast <- Lrnr_glm_fast$new(method = "eigen")
glm_fast_fit <- lrnr_glm_fast$train(task)
glm_fast_preds <- glm_fast_fit$predict()
```

Lrnr_glm_semiparametric

Semiparametric Generalized Linear Models

Description

This learner provides fitting procedures for semiparametric generalized linear models using a specified baseline learner and `glm.fit`. Models of the form `linkfun(E[Y|A,W]) = linkfun(E[Y|A=0,W]) + A * f(W)` are supported, where `A` is a binary or continuous interaction variable, `W` are all of the covariates in the task excluding the interaction variable, and `f(W)` is a user-specified parametric function of the non-interaction-variable covariates (e.g., `f(W) = model.matrix(formula_sp, W)`).

The baseline function E[Y|A=0,W] is fit using a user-specified learner, possibly pooled over values of interaction variable A, and then projected onto the semiparametric model.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- formula_parametric = NULL: A formula object specifying the parametric function of the non-interaction-variable covariates.
- lrnr_baseline: A baseline learner for estimation of the nonparametric component. This can be pooled or unpooled by specifying return_matrix_predictions.
- interaction_variable = NULL: An interaction variable name present in the task's data that will be used to multiply by the design matrix generated by formula_sp. If NULL (default) then the interaction variable is treated identically 1. When this learner is used for estimation of the outcome regression in an effect estimation procedure (e.g., when using sl3 within package tmle3), it is recommended that interaction_variable be set as the name of the treatment variable.
- family = NULL: A family object whose link function specifies the type of semiparametric model. For partially-linear least-squares regression, partially-linear logistic regression, and partially-linear log-linear regression family should be set to guassian(), binomial(), and poisson(), respectively.
- append_interaction_matrix = TRUE: Whether lrnr_baseline should be fit on cbind(task$X,A*V), where A is the interaction_variable and V is the design matrix obtained from formula_sp. Note that if TRUE (default) the resulting estimator will be projected onto the semiparametric model using glm.fit. If FALSE and interaction_variable is binary, the semiparametric model is learned by stratifying on interaction_variable; Specifically, lrnr_baseline is used to estimate E[Y|A=0,W] by subsetting to only observations with A = 0, i.e., subsetting to only observations with interaction_variable = 0, and where W are the other covariates in the task that are not the interaction_variable. In the binary interaction_variable case, setting append_interaction_matrix = TRUE allows one to pool the learning across treatment arms and can enhance performance of additive models.
- return_matrix_predictions = FALSE: Whether to return a matrix output with three columns being E[Y|A=0,W], E[Y|A=1,W], E[Y|A,W] in the learner's fit_object, where A is the interaction_variable and W are the other covariates in the task that are not the interaction_variable. Only used if the interaction_variable is binary.
- ...: Any additional parameters that can be considered by Lrnr_base.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions,

Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
# simulate some data
set.seed(459)
n <- 200
W <- runif(n, -1, 1)
A <- rbinom(n, 1, plogis(W))
Y_continuous <- rnorm(n, mean = A + W, sd = 0.3)
Y_binary <- rbinom(n, 1, plogis(A + W))
Y_count <- rpois(n, exp(A + W))
data <- data.table::data.table(W, A, Y_continuous, Y_binary, Y_count)

# Make tasks
task_continuous <- sl3_Task$new(
  data,
  covariates = c("A", "W"), outcome = "Y_continuous"
)
task_binary <- sl3_Task$new(
  data,
  covariates = c("A", "W"), outcome = "Y_binary"
)
task_count <- sl3_Task$new(
  data,
  covariates = c("A", "W"), outcome = "Y_count",
  outcome_type = "continuous"
)

formula_sp <- ~ 1 + W

# fit partially-linear regression with append_interaction_matrix = TRUE
set.seed(100)
lrnr_glm_sp_gaussian <- Lrnr_glm_semiparametric$new(
  formula_sp = formula_sp, family = gaussian(),
  lrnr_baseline = Lrnr_glm$new(),
  interaction_variable = "A", append_interaction_matrix = TRUE
)
lrnr_glm_sp_gaussian <- lrnr_glm_sp_gaussian$train(task_continuous)
preds <- lrnr_glm_sp_gaussian$predict(task_continuous)
beta <- lrnr_glm_sp_gaussian$fit_object$coefficients
# in this case, since append_interaction_matrix = TRUE, it is equivalent to:
```

```
V <- model.matrix(formula_sp, task_continuous$data)
X <- cbind(task_continuous$data[["W"]], task_continuous$data[["A"]] * V)
X0 <- cbind(task_continuous$data[["W"]], 0 * V)
colnames(X) <- c("W", "A", "A*W")
Y <- task_continuous$Y
set.seed(100)
beta_equiv <- coef(glm(X, Y, family = "gaussian"))[c(3, 4)]
# actually, the glm fit is projected onto the semiparametric model
# with glm.fit, no effect in this case
print(beta - beta_equiv)
# fit partially-linear regression w append_interaction_matrix = FALSE`
set.seed(100)
lrnr_glm_sp_gaussian <- Lrnr_glm_semiparametric$new(
  formula_sp = formula_sp, family = gaussian(),
  lrnr_baseline = Lrnr_glm$new(family = gaussian()),
  interaction_variable = "A",
  append_interaction_matrix = FALSE
)
lrnr_glm_sp_gaussian <- lrnr_glm_sp_gaussian$train(task_continuous)
preds <- lrnr_glm_sp_gaussian$predict(task_continuous)
beta <- lrnr_glm_sp_gaussian$fit_object$coefficients
# in this case, since append_interaction_matrix = FALSE, it is equivalent to
# the following
cntrls <- task_continuous$data[["A"]] == 0 # subset to control arm
V <- model.matrix(formula_sp, task_continuous$data)
X <- cbind(rep(1, n), task_continuous$data[["W"]])
Y <- task_continuous$Y
set.seed(100)
beta_Y0W <- lrnr_glm_sp_gaussian$fit_object$lrnr_baseline$fit_object$coefficients
# subset to control arm
beta_Y0W_equiv <- coef(
  glm.fit(X[cntrls, , drop = F], Y[cntrls], family = gaussian())
)
EY0 <- X %*% beta_Y0W
beta_equiv <- coef(glm.fit(A * V, Y, offset = EY0, family = gaussian()))
print(beta_Y0W - beta_Y0W_equiv)
print(beta - beta_equiv)

# fit partially-linear logistic regression
lrnr_glm_sp_binomial <- Lrnr_glm_semiparametric$new(
  formula_sp = formula_sp, family = binomial(),
  lrnr_baseline = Lrnr_glm$new(), interaction_variable = "A",
  append_interaction_matrix = TRUE
)
lrnr_glm_sp_binomial <- lrnr_glm_sp_binomial$train(task_binary)
preds <- lrnr_glm_sp_binomial$predict(task_binary)
beta <- lrnr_glm_sp_binomial$fit_object$coefficients

# fit partially-linear log-link (relative-risk) regression
# Lrnr_glm$new(family = "poisson") setting requires that lrnr_baseline
# predicts nonnegative values. It is recommended to use poisson
# regression-based learners.
lrnr_glm_sp_poisson <- Lrnr_glm_semiparametric$new(
```

```
  formula_sp = formula_sp, family = poisson(),
  lrnr_baseline = Lrnr_glm$new(family = "poisson"),
  interaction_variable = "A",
  append_interaction_matrix = TRUE
)
lrnr_glm_sp_poisson <- lrnr_glm_sp_poisson$train(task_count)
preds <- lrnr_glm_sp_poisson$predict(task_count)
beta <- lrnr_glm_sp_poisson$fit_object$coefficients

## End(Not run)
```

`Lrnr_grf` *Generalized Random Forests Learner*

Description

This learner implements Generalized Random Forests, using the **grf** package. This is a pluggable package for forest-based statistical estimation and inference. GRF currently provides nonparametric methods for least-squares regression, quantile regression, and treatment effect estimation (optionally using instrumental variables). Current implementation trains a regression forest that can be used to estimate quantiles of the conditional distribution of (Y|X=x).

Format

`R6Class` object.

Value

Learner object with methods for training and prediction. See `Lrnr_base` for documentation on learners.

Parameters

`num.trees = 2000` Number of trees grown in the forest. NOTE: Getting accurate confidence intervals generally requires more trees than getting accurate predictions.

`quantiles = c(0.1, 0.5, 0.9)` Vector of quantiles used to calibrate the forest.

`regression.splitting = FALSE` Whether to use regression splits when growing trees instead of specialized splits based on the quantiles (the default). Setting this flag to `TRUE` corresponds to the approach to quantile forests from Meinshausen (2006).

`clusters = NULL` Vector of integers or factors specifying which cluster each observation corresponds to.

`equalize.cluster.weights = FALSE` If `FALSE`, each unit is given the same weight (so that bigger clusters get more weight). If `TRUE`, each cluster is given equal weight in the forest. In this case, during training, each tree uses the same number of observations from each drawn cluster: If the smallest cluster has K units, then when we sample a cluster during training, we only give a random K elements of the cluster to the tree-growing procedure. When estimating average treatment effects, each observation is given weight 1/cluster size, so that the total weight of each cluster is the same.

`sample.fraction = 0.5` Fraction of the data used to build each tree. NOTE: If `honesty = TRUE`, these subsamples will further be cut by a factor of `honesty.fraction.`.

`mtry = NULL` Number of variables tried for each split. By default, this is set based on the dimensionality of the predictors.

`min.node.size = 5` A target for the minimum number of observations in each tree leaf. Note that nodes with size smaller than `min.node.size` can occur, as in the **randomForest** package.

`honesty = TRUE` Whether or not honest splitting (i.e., sub-sample splitting) should be used.

`alpha = 0.05` A tuning parameter that controls the maximum imbalance of a split.

`imbalance.penalty = 0` A tuning parameter that controls how harshly imbalanced splits are penalized.

`num.threads = 1` Number of threads used in training. If set to `NULL`, the software automatically selects an appropriate amount.

`quantiles_pred` Vector of quantiles used to predict. This can be different than the vector of quantiles used for training.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
# load example data
data(cpp_imputed)
```

```
# create sl3 task
task <- sl3_Task$new(
  cpp_imputed,
  covariates = c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs"),
  outcome = "haz"
)

# train grf learner and make predictions
lrnr_grf <- Lrnr_grf$new(seed = 123)
lrnr_grf_fit <- lrnr_grf$train(task)
lrnr_grf_pred <- lrnr_grf_fit$predict()
```

`Lrnr_grfcate` *Generalized Random Forests for Conditional Average Treatment Effects*

Description

This learner implements the so-called "Causal Forests" estimator of the conditional average treatment effect (CATE) using the **grf** package function `causal_forest`. This learner is intended for use in the `tmle3mopttx` package, where it is necessary to fit the CATE, and then predict CATE values from new covariate data. As such, this learner requires a treatment/exposure node to be specified (`A`).

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `A`: Column name in the `sl3_Task`'s `covariates` that indicates the treatment/exposure of interest. The treatment assignment must be a binary or real numeric vector with no NAs.
- `...`: Other parameters passed to `causal_forest`. See its documentation for details.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`,

Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(mtcars)
mtcars_task <- sl3_Task$new(
  data = mtcars,
  covariates = c("cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am"),
  outcome = "mpg"
)
# simple prediction with lasso penalty
grfcate_lrnr <- Lrnr_grfcate$new(A = "vs")
grfcate_fit <- grfcate_lrnr$train(mtcars_task)
grf_cate_predictions <- grfcate_fit$predict()
```

Lrnr_gru_keras *Recurrent Neural Network with Gated Recurrent Unit (GRU) with Keras*

Description

This learner supports Recurrent Neural Networks (RNNs) with Gated Recurrent Units (GRU). This learner leverages the same principles as LSTM networks but is more streamlined and thus cheaper to run, at the expense of some loss in representational power. This learner uses the **keras** package. Note that all preprocessing, such as differencing and seasonal effects for time series, should be addressed before using this learner. Desired lags of the time series should be added as predictors before using the learner.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- batch_size: How many times should the training data be used to train the neural network?
- units: Positive integer, dimensionality of the output space.
- dropout: Float between 0 and 1. Fraction of the input units to drop.
- recurrent_dropout: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

- `activation`: Activation function to use. If you pass NULL, no activation is applied (e.g., "linear" activation: `a(x) = x`).
- `recurrent_activation`: Activation function to use for the recurrent step.
- `recurrent_out`: Activation function to use for the output step.
- `epochs`: Number of epochs to train the model.
- `lr`: Learning rate.
- `layers`: How many LSTM layers. Only allows for 1 or 2.
- `callbacks`: List of callbacks, which is a set of functions to be applied at given stages of the training procedure. Default callback function `callback_early_stopping` stops training if the validation loss does not improve across `patience` number of epochs.
- `...`: Other parameters passed to `keras`.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
## Not run:
library(origami)
data(bsds)

# make folds appropriate for time-series cross-validation
folds <- make_folds(bsds,
  fold_fun = folds_rolling_window, window_size = 500,
  validation_size = 100, gap = 0, batch = 50
)

# build task by passing in external folds structure
task <- sl3_Task$new(
  data = bsds,
  folds = folds,
  covariates = c(
    "weekday", "temp"
  ),
  outcome = "cnt"
)
```

```
# create tasks for taining and validation (simplifed example)
train_task <- training(task, fold = task$folds[[1]])
valid_task <- validation(task, fold = task$folds[[1]])

# instantiate learner, then fit and predict (simplifed example)
gru_lrnr <- Lrnr_gru_keras$new(batch_size = 1, epochs = 200)
gru_fit <- gru_lrnr$train(train_task)
gru_preds <- gru_fit$predict(valid_task)

## End(Not run)
```

`Lrnr_h2o_grid` *Grid Search Models with h2o*

Description

`Lrnr_h2o_grid` - This learner provides facilities for fitting various types of models with support for grid search over the hyperparameter space of such models, using an interface to the H2O platform. For details on the procedures available and any limitations, consult the documentation of the `h2o` package.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`algorithm` An h2o ML algorithm. For a list, please see https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science.html#.

`seed=1` RNG see to use when fitting.

`distribution=NULL` Specifies the loss function for GBM, Deep Learning, and XGBoost.

`intercept=TRUE` If `TRUE`, and intercept term is included.

`standardize=TRUE` Standardize covariates to have mean = 0 and SD = 1.

`lambda=0` Lasso Parameter.

`max_iterations=100` Maximum number of iterations.

`ignore_const_columns=FALSE` If `TRUE`, drop constant covariate columns

`missing_values_handling="Skip"` How to handle missing values.

`...` Other arguments passed to the h2o algorithm of choice. See https://docs.h2o.ai/h2o/latest-stable/h2o-docs/parameters.html for a list.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by Lrnr_base, and shared by all learners.

covariates A character vector of covariates. The learner will use this to subset the covariates for any specified task

outcome_type A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

... All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
library(h2o)
suppressWarnings(h2o.init())
set.seed(1)

# load example data
data(cpp_imputed)
covars <- c(
  "apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs",
  "sexn"
)
outcome <- "haz"
cpp_imputed <- cpp_imputed[1:150, ]

# create sl3 task
task <- sl3_Task$new(cpp_imputed, covariates = covars, outcome = outcome)

# h2o grid search hyperparameter alpha
h2o_glm_grid <- Lrnr_h2o_grid$new(
  algorithm = "glm",
  hyper_params = list(alpha = c(0, 0.5))
)
```

```
h2o_glm_grid_fit <- h2o_glm_grid$train(task)
pred <- h2o_glm_grid_fit$predict()

## End(Not run)
```

`Lrnr_hal9001` *Scalable Highly Adaptive Lasso (HAL)*

Description

The Highly Adaptive Lasso (HAL) is a nonparametric regression function that has been demonstrated to optimally estimate functions with bounded (finite) variation norm. The algorithm proceeds by first building an adaptive basis (i.e., the HAL basis) based on indicator basis functions (or higher-order spline basis functions) representing covariates and interactions of the covariates up to a pre-specified degree. The fitting procedures included in this learner use `fit_hal` from the **hal9001** package. For details on HAL regression, consider consulting the following Benkeser and van der Laan (2016)), Coyle et al. (2020)), Hejazi et al. (2020)).

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `max_degree = 2`: An integer specifying the highest order of interaction terms for which basis functions ought to be generated.
- `smoothness_orders = 1`: An integer specifying the smoothness of the basis functions. See details of `hal9001` package's `fit_hal` function for more information.
- `num_knots = 5`: An integer vector of length 1 or of length `max_degree`, specifying the maximum number of knot points (i.e., bins) for each covariate. If `num_knots` is a unit-length vector, then the same `num_knots` are used for each degree. See details of `hal9001` package's `fit_hal` function for more information.
- `fit_control`: List of arguments, including those specified in `fit_hal`'s `fit_control` documentation, and any additional arguments to be passed to `cv.glmnet` or `glmnet`. See the `hal9001` package `fit_hal` function fdocumentation or more information.
- `...`: Other parameters passed to `fit_hal` and additional arguments defined in `Lrnr_base`, such as `params` like `formula`.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# instantiate with max 2-way interactions, 0-order splines, and binning
# (i.e., num_knots) that decreases with increasing interaction degree
hal_lrnr <- Lrnr_hal9001$new(max_degree = 2, num_knots = c(5, 3))
hal_fit <- hal_lrnr$train(task)
hal_preds <- hal_fit$predict()
```

Lrnr_haldensify *Conditional Density Estimation with the Highly Adaptive LASSO*

Description

Conditional Density Estimation with the Highly Adaptive LASSO

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- grid_type = "equal_range": A character indicating the strategy to be used in creating bins along the observed support of A. For bins of equal range, use "equal_range"; consult the documentation of cut_interval for further information. To ensure each bin has the same number of observations, use "equal_mass"; consult the documentation of cut_number for

details. The default is "equal_range" since this has been found to provide better performance in simulation experiments; however, both types may be specified (i.e., c("equal_range", "equal_mass")) together, in which case cross-validation will be used to select the optimal binning strategy.

- n_bins = c(3, 5): This numeric value indicates the number of bins into which the support of A is to be divided. As with grid_type, multiple values may be specified, in which cross-validation will be used to select the optimal number of bins.
- lambda_seq = exp(seq(-1, -13, length = 1000L)): A numeric sequence of regularization parameter values of Lasso regression, which are passed to fit_hal via its argument lambda, itself passed to glmnet.
- trim_dens = 1/sqrt(n): A numeric giving the minimum allowed value of the resultant density predictions. Any predicted density values below this tolerance threshold are set to the indicated minimum. The default is to use the inverse of the square root of the sample size of the prediction set, i.e., 1/sqrt(n); another notable choice is 1/sqrt(n)/log(n). If there are observations in the prediction set with values of new_A outside of the support of the training set, their predictions are similarly truncated.
- ...: Other arguments to be passed directly to haldensify. See its documentation for details.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
library(dplyr)
data(cpp_imputed)
covars <- c("parity", "sexn")
outcome <- "haz"

# create task
task <- cpp_imputed %>%
  slice(seq(1, nrow(.), by = 3)) %>%
  filter(agedays == 1) %>%
  sl3_Task$new(
    covariates = covars,
    outcome = outcome
  )
```

```
# instantiate the learner
hal_dens <- Lrnr_haldensify$new(
  grid_type = "equal_range",
  n_bins = c(3, 5),
  lambda_seq = exp(seq(-1, -13, length = 100))
)

# fit and predict densities
hal_dens_fit <- hal_dens$train(task)
hal_dens_preds <- hal_dens_fit$predict()

## End(Not run)
```

`Lrnr_HarmonicReg` *Harmonic Regression*

Description

This learner fits first harmonics in a Fourier expansion to one or more time series. Fourier decomposition relies on `fourier`, and the time series is fit using `tslm`. For further details on working with harmonic regression for time-series with package **forecast**, consider consulting Hyndman et al. (2021)) and Hyndman and Khandakar (2008)).

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `K`: Maximum order of the fourier terms. Passed to `fourier`.
- `freq`: The frequency of the time series.
- `...`: Other parameters passed to `fourier`.

References

Hyndman R, Athanasopoulos G, Bergmeir C, Caceres G, Chhay L, O'Hara-Wild M, Petropoulos F, Razbash S, Wang E, Yasmeen F (2021). *forecast: Forecasting functions for time series and linear models*. R package version 8.14, https://pkg.robjhyndman.com/forecast/.

Hyndman RJ, Khandakar Y (2008). "Automatic time series forecasting: the forecast package for R." *Journal of Statistical Software*, **26**(3), 1–22. https://www.jstatsoft.org/article/view/v027i03.

See Also

Other Learners: Custom_chain, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
library(origami)
library(data.table)
data(bsds)

# make folds appropriate for time-series cross-validation
folds <- make_folds(bsds,
  fold_fun = folds_rolling_window, window_size = 500,
  validation_size = 100, gap = 0, batch = 50
)

# build task by passing in external folds structure
task <- sl3_Task$new(
  data = bsds,
  folds = folds,
  covariates = c(
    "weekday", "temp"
  ),
  outcome = "cnt"
)

# create tasks for taining and validation
train_task <- training(task, fold = task$folds[[1]])
valid_task <- validation(task, fold = task$folds[[1]])

# instantiate learner, then fit and predict
HarReg_learner <- Lrnr_HarmonicReg$new(K = 7, freq = 105)
HarReg_fit <- HarReg_learner$train(train_task)
HarReg_preds <- HarReg_fit$predict(valid_task)
```

`Lrnr_independent_binomial`

Classification from Binomial Regression

Description

This learner provides converts a binomial learner into a multinomial learner using a series of independent binomials. The procedure is modeled on https://en.wikipedia.org/wiki/Multinomial_logistic_regression#As_a_set_of_independent_binary_regressions

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`binomial_learner` The learner to wrap.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by Lrnr_base, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
library(dplyr)

# load example data
data(cpp)
```

```
cpp <- cpp %>%
  select(c(bmi, agedays, feeding)) %>%
  mutate(feeding = as.factor(feeding)) %>%
  na.omit()

# create sl3 task
task <- make_sl3_Task(cpp,
  covariates = c("agedays", "bmi"),
  outcome = "feeding"
)

# train independent binomial learner and make predictions
lrnr_indbinomial <- make_learner(Lrnr_independent_binomial)
fit <- lrnr_indbinomial$train(task)
preds <- fit$predict(task)
```

`Lrnr_lightgbm` *LightGBM: Light Gradient Boosting Machine*

Description

This learner provides fitting procedures for `lightgbm` models, using the **lightgbm** package, via `lgb.train`. These gradient boosted decision tree models feature faster training speed and efficiency, lower memory usage than competing frameworks (e.g., from the **xgboost** package), better prediction accuracy, and improved handling of large-scale data. For details on the fitting procedure and its tuning parameters, consult the documentation of the **lightgbm** package. The LightGBM framework was introduced in Ke et al. (2017)).

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `num_threads = 1L`: Number of threads for hyperthreading.
- `...`: Other arguments passed to `lgb.train`. See its documentation for further details.

References

Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu T (2017). "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." In *Advances in Neural Information Processing Systems*, volume 30, 3146–3154.

See Also

Lrnr_gbm for standard gradient boosting models (via the **gbm** package) and Lrnr_xgboost for the extreme gradient boosted tree models from the Xgboost framework (via the **xgboost** package).

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
# currently disabled since LightGBM crashes R on Windows
# more info at https://github.com/tlverse/sl3/issues/344
data(cpp_imputed)
# create task for prediction
cpp_task <- sl3_Task$new(
  data = cpp_imputed,
  covariates = c("bmi", "parity", "mage", "sexn"),
  outcome = "haz"
)

# initialization, training, and prediction with the defaults
lgb_lrnr <- Lrnr_lightgbm$new()
lgb_fit <- lgb_lrnr$train(cpp_task)
lgb_preds <- lgb_fit$predict()

# get feature importance from fitted model
lgb_varimp <- lgb_fit$importance()

## End(Not run)
```

Lrnr_lstm_keras *Long short-term memory Recurrent Neural Network (LSTM) with Keras*

Description

This learner supports long short-term memory (LSTM) recurrent neural network algorithm. This learner uses the keras package. Note that all preprocessing, such as differencing and seasonal effects for time series should be addressed before using this learner. Desired lags of the time series should be added as predictors before using the learner.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- batch_size: How many times should the training data be used to train the neural network?
- units: Positive integer, dimensionality of the output space.
- dropout: Float between 0 and 1. Fraction of the input units to drop.
- recurrent_dropout: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.
- activation: Activation function to use. If you pass NULL, no activation is applied (e.g., "linear" activation: a(x) = x).
- recurrent_activation: Activation function to use for the recurrent step.
- recurrent_out: Activation function to use for the output step.
- epochs: Number of epochs to train the model.
- lr: Learning rate.
- layers: How many LSTM layers. Only allows for 1 or 2.
- callbacks: List of callbacks, which is a set of functions to be applied at given stages of the training procedure. Default callback function callback_early_stopping stops training if the validation loss does not improve across patience number of epochs.
- ...: Other parameters passed to keras.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
library(origami)
data(bsds)

# make folds appropriate for time-series cross-validation
folds <- make_folds(bsds,
  fold_fun = folds_rolling_window, window_size = 500,
  validation_size = 100, gap = 0, batch = 50
)

# build task by passing in external folds structure
task <- sl3_Task$new(
  data = bsds,
  folds = folds,
  covariates = c(
    "weekday", "temp"
  ),
  outcome = "cnt"
)

# create tasks for taining and validation (simplifed example)
train_task <- training(task, fold = task$folds[[1]])
valid_task <- validation(task, fold = task$folds[[1]])

# instantiate learner, then fit and predict (simplifed example)
lstm_lrnr <- Lrnr_lstm_keras$new(batch_size = 1, epochs = 200)
lstm_fit <- lstm_lrnr$train(train_task)
lstm_preds <- lstm_fit$predict(valid_task)

## End(Not run)
```

Lrnr_mean *Fitting Intercept Models*

Description

This learner provides fitting procedures for intercept models. Such models predict the outcome variable simply as the mean of the outcome vector.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- . . . : Not used.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# simple, main-terms GLM
lrnr_mean <- make_learner(Lrnr_mean)
mean_fit <- lrnr_mean$train(task)
mean_preds <- mean_fit$predict()
```

Lrnr_multiple_ts *Stratify univariable time-series learners by time-series*

Description

Stratify univariable time-series learners by time-series

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`learner="learner"` An initialized Lrnr_* object.

`variable_stratify="variable_stratify"` A `character` giving the variable in the covariates on which to stratify. Supports only variables with discrete levels coded as `numeric`.

`...` Other parameters passed directly to `learner$train`. See its documentation for details.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
library(origami)
library(dplyr)

set.seed(123)

# Simulate simple AR(2) process
data <- matrix(arima.sim(model = list(ar = c(0.9, -0.2)), n = 200))
id <- c(rep("Series_1", 50), rep("Series_2", 50), rep("Series_3", 50), rep("Series_4", 50))
data <- data.frame(data)
data$id <- as.factor(id)
data <- data %>%
  group_by(id) %>%
  dplyr::mutate(time = 1:n())

data$W1 <- rbinom(200, 1, 0.6)
data$W2 <- rbinom(200, 1, 0.2)

folds <- origami::make_folds(data,
  t = max(data$time),
  id = data$id,
  time = data$time,
  fold_fun = folds_rolling_window_pooled,
  window_size = 20,
  validation_size = 15,
  gap = 0,
  batch = 10
)
```

```
task <- sl3_Task$new(
  data = data, outcome = "data",
  time = "time", id = "id",
  covariates = c("W1", "W2"),
  folds = folds
)

train_task <- training(task, fold = task$folds[[1]])
valid_task <- validation(task, fold = task$folds[[1]])

lrnr_arima <- Lrnr_arima$new()
multiple_ts_arima <- Lrnr_multiple_ts$new(learner = lrnr_arima)

multiple_ts_arima_fit <- multiple_ts_arima$train(train_task)
multiple_ts_arima_preds <- multiple_ts_arima_fit$predict(valid_task)
```

`Lrnr_multivariate` *Multivariate Learner*

Description

This learner applies a univariate outcome learner across a vector of outcome variables, effectively transforming it into a multivariate outcome learner

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`learner` The learner to wrap.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
library(data.table)

# simulate data
set.seed(123)
n <- 1000
p <- 5
pY <- 3
W <- matrix(rnorm(n * p), nrow = n)
colnames(W) <- sprintf("W%d", seq_len(p))
Y <- matrix(rnorm(n * pY, 0, 0.2) + W[, 1], nrow = n)
colnames(Y) <- sprintf("Y%d", seq_len(pY))
data <- data.table(W, Y)
covariates <- grep("W", names(data), value = TRUE)
outcomes <- grep("Y", names(data), value = TRUE)

# make sl3 task
task <- sl3_Task$new(data.table::copy(data),
  covariates = covariates,
  outcome = outcomes
)

# train multivariate learner and make predictions
mv_learner <- make_learner(Lrnr_multivariate, make_learner(Lrnr_glm_fast))
mv_fit <- mv_learner$train(task)
mv_pred <- mv_fit$predict(task)
mv_pred <- unpack_predictions(mv_pred)
```

Lrnr_nnet *Feed-Forward Neural Networks and Multinomial Log-Linear Models*

Description

This learner provides feed-forward neural networks with a single hidden layer, and for multinomial log-linear models.

Format

R6Class object.

Value

Learner object with methods for both training and prediction. See Lrnr_base for documentation on learners.

Parameters

formula A formula of the form class ~ x1 + x2 + ...

weights (case) weights for each example – if missing defaults to 1

size number of units in the hidden layer. Can be zero if there are skip-layer units.

entropy switch for entropy (= maximum conditional likelihood) fitting. Default by least-squares.

decay parameter for weight decay. Default 0.

maxit maximum number of iterations. Default 100.

linout switch for linear output units. Default logistic output units.

... Other parameters passed to nnet.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by Lrnr_base, and shared by all learners.

covariates A character vector of covariates. The learner will use this to subset the covariates for any specified task

outcome_type A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

... All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
set.seed(123)

# load example data
data(cpp_imputed)
covars <- c("bmi", "parity", "mage", "sexn")
outcome <- "haz"

# create sl3 task
task <- sl3_Task$new(cpp_imputed, covariates = covars, outcome = outcome)

# train neural networks and make predictions
lrnr_nnet <- Lrnr_nnet$new(linout = TRUE, size = 10, maxit = 1000)
fit <- lrnr_nnet$train(task)
preds <- fit$predict(task)
```

Lrnr_nnls *Non-negative Linear Least Squares*

Description

This learner provides fitting procedures for models via non-negative linear least squares regression, using **nnls** package's `nnls` function.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `convex = FALSE`: Normalize the coefficients to be a convex combination.
- `...`: Other parameters passed to `nnls`.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`,

Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

lrnr_nnls <- make_learner(Lrnr_nnls)
nnls_fit <- lrnr_nnls$train(task)
nnls_preds <- nnls_fit$predict()

# NNLS is commonly used as a metalearner in a super learner (i.e., Lrnr_sl)
lrnr_glm <- make_learner(Lrnr_glm)
lrnr_glmnet <- Lrnr_glmnet$new()
lrnr_mean <- Lrnr_mean$new()
learners <- c(lrnr_glm, lrnr_glmnet, lrnr_mean)
names(learners) <- c("glm", "lasso", "mean") # optional, renaming learners
simple_learner_stack <- make_learner(Stack, learners)
sl <- Lrnr_sl$new(learners = simple_learner_stack, metalearner = lrnr_nnls)
sl_fit <- sl$train(task)
sl_preds <- sl_fit$predict()
```

Lrnr_optim *Optimize Metalearner according to Loss Function using optim*

Description

This meta-learner provides fitting procedures for any pairing of loss function and metalearner function, subject to constraints. The optimization problem is solved by making use of `optim`, For further details, consult the documentation of `optim`.

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`learner_function=metalearner_linear` A function(alpha, X) that takes a vector of covariates and a matrix of data and combines them into a vector of predictions. See metalearners for options.

loss_function=loss_squared_error A function(pred, truth) that takes prediction and truth vectors and returns a loss vector. See loss_functions for options.

intercept=FALSE If true, X includes an intercept term.

init_0=FALSE If true, alpha is initialized to all 0's, useful for TMLE. Otherwise, it is initialized to equal weights summing to 1, useful for Super Learner.

... Not currently used.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by Lrnr_base, and shared by all learners.

covariates A character vector of covariates. The learner will use this to subset the covariates for any specified task

outcome_type A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

... All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

---

Lrnr_pca *Principal Component Analysis and Regression*

Description

This learner provides facilities for performing principal components analysis (PCA) to reduce the dimensionality of a data set to a pre-specified value. For further details, consult the documentation of prcomp from the core package stats. This learner object is primarily intended for use with other learners as part of a pre-processing pipeline.

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

n_comp A numeric value indicating the number of components to be produced as a result of the PCA dimensionality reduction. For convenience, this defaults to two (2) components.

center A logical value indicating whether the input data matrix should be centered before performing PCA. This defaults to TRUE since that is the recommended practice. Consider consulting the documentation of prcomp for details.

scale. A logical value indicating whether the input data matrix should be scaled (to unit variance) before performing PCA. Consider consulting the documentation of prcomp for details.

... Other optional parameters to be passed to prcomp. Consider consulting the documentation of prcomp for details.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by Lrnr_base, and shared by all learners.

covariates A character vector of covariates. The learner will use this to subset the covariates for any specified task

outcome_type A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

... All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
set.seed(37912)

# load example data
ncomp <- 3
```

```
data(cpp_imputed)
covars <- c(
  "apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs",
  "sexn"
)
outcome <- "haz"

# create sl3 task
task <- sl3_Task$new(cpp_imputed, covariates = covars, outcome = outcome)

# define learners
glm_fast <- Lrnr_glm_fast$new(intercept = FALSE)
pca_sl3 <- Lrnr_pca$new(n_comp = ncomp, center = TRUE, scale. = TRUE)
pcr_pipe_sl3 <- Pipeline$new(pca_sl3, glm_fast)

# create stacks + train and predict
pcr_pipe_sl3_fit <- pcr_pipe_sl3$train(task)
pcr_pred <- pcr_pipe_sl3_fit$predict()
```

`Lrnr_pkg_SuperLearner` *Use SuperLearner Wrappers, Screeners, and Methods, in sl3*

Description

These learners provide an interface to the wrapper functions, screening algorithms, and combination methods provided by the `SuperLearner` package. These components add support for a range of algorithms not currently implemented natively in `sl3`.

`Lrnr_pkg_SuperLearner` - Interface for `SuperLearner` wrapper functions. Use `SuperLearner::listWrappers("SL")` for a list.

Use `SuperLearner::listWrappers("method")` for a list of options.

Use `SuperLearner::listWrappers("screen")` for a list of options.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`SL_wrapper` The wrapper function to use.

`...` Currently not used.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by Lrnr_base, and shared by all learners.

- `covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task
- `outcome_type` A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified
- `...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

---

Lrnr_polspline — *Polyspline - multivariate adaptive polynomial spline regression (polymars) and polychotomous regression and multiple classification (polyclass)*

Description

This learner provides fitting procedures for an adaptive regression procedure using piecewise linear splines to model the response, using the the **polspline** package' functions polymars (for continuous outcome prediction) or polyclass (for binary or categorical outcome prediction).

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- ...: Other parameters passed to polymars, polyclass, or additional arguments defined in Lrnr_base (such as params like formula). See their documentation for details.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")
polspline_lrnr <- Lrnr_caret$new(method = "rf")
set.seed(693)
polspline_lrnr_fit <- polspline_lrnr$train(task)
polspline_lrnr_predictions <- polspline_lrnr_fit$predict()

## End(Not run)
```

Lrnr_pooled_hazards *Classification from Pooled Hazards*

Description

This learner provides converts a binomial learner into a multinomial learner using a pooled hazards model.

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`binomial_learner` The learner to wrap.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
library(data.table)
set.seed(74294)

n <- 500
x <- rnorm(n)
epsilon <- rnorm(n)
y <- 3 * x + epsilon
data <- data.table(x = x, y = y)
task <- sl3_Task$new(data, covariates = c("x"), outcome = "y")

# instantiate learners
hal <- Lrnr_hal9001$new(
  lambda = exp(seq(-1, -13, length = 100)),
  max_degree = 6,
  smoothness_orders = 0
)
hazard_learner <- Lrnr_pooled_hazards$new(hal)
density_learner <- Lrnr_density_discretize$new(
  hazard_learner,
```

```
  type = "equal_range",
  n_bins = 5
)

# fit discrete density model to pooled hazards data
set.seed(74294)
fit_density <- density_learner$train(task)
pred_density <- fit_density$predict()
```

`Lrnr_randomForest` *Random Forests*

Description

This learner provides fitting procedures for random forest models, using the `randomForest` package, using `randomForest` function.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

- `ntree = 500`: Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.
- `keep.forest = TRUE`: If `TRUE`, forest is stored, which is required for prediction.
- `nodesize = 5`: Minimum number of observations in a terminal node.
- `...`: Other parameters passed to `randomForest`.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
data(cpp_imputed)
# create task for prediction
cpp_task <- sl3_Task$new(
  data = cpp_imputed,
  covariates = c("bmi", "parity", "mage", "sexn"),
  outcome = "haz"
)
# initialization, training, and prediction with the defaults
rf_lrnr <- Lrnr_randomForest$new()
rf_fit <- rf_lrnr$train(cpp_task)
rf_preds <- rf_fit$predict()
```

`Lrnr_ranger` *Ranger: Fast(er) Random Forests*

Description

This learner provides fitting procedures for a faster implementation of Random Forests, using the routines from **ranger** (described in Wright and Ziegler (2017)) through a call to the function `ranger`. Variable importance functionality is also provided through invocation of the `importance` method.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `num.trees = 500`: Number of trees to be used in growing the forest.
- `write.forest = TRUE`: If `TRUE`, forest is stored, which is required for prediction. Set to `FALSE` to reduce memory usage if downstream prediction is not intended.
- `importance = "none"`: Variable importance mode, one of "none", "impurity", "impurity_corrected", "permutation". The "impurity" measure is the Gini index for classification, the variance of the responses for regression, and the sum of test statistics (for survival analysis, see the `splitrule` argument of `ranger`).
- `num.threads = 1`: Number of threads.
- `...`: Other parameters passed to `ranger`. See its documentation for details.

References

Wright MN, Ziegler A (2017). "ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R." *Journal of Statistical Software*, **77**(1), 1–17. doi:10.18637/jss.v077.i01.

See Also

Lrnr_randomForest for a similar learner using **randomForest**

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(mtcars)
# create task for prediction
mtcars_task <- sl3_Task$new(
  data = mtcars,
  covariates = c(
    "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am",
    "gear", "carb"
  ),
  outcome = "mpg"
)
# initialization, training, and prediction with the defaults
ranger_lrnr <- Lrnr_ranger$new()
ranger_fit <- ranger_lrnr$train(mtcars_task)
ranger_preds <- ranger_fit$predict()

# variable importance
ranger_lrnr_importance <- Lrnr_ranger$new(importance = "impurity_corrected")
ranger_fit_importance <- ranger_lrnr_importance$train(mtcars_task)
ranger_importance <- ranger_fit_importance$importance()

# screening based on variable importance, example in glm pipeline
ranger_importance_screener <- Lrnr_screener_importance$new(
  learner = ranger_lrnr_importance, num_screen = 3
)
glm_lrnr <- make_learner(Lrnr_glm)
ranger_screen_glm_pipe <- Pipeline$new(ranger_importance_screener, glm_lrnr)
ranger_screen_glm_pipe_fit <- ranger_screen_glm_pipe$train(mtcars_task)
```

Lrnr_revere_task *Learner that chains into a revere task*

Description

A wrapper around a revere generator that produces a revere task on chain

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`revere_function` The revere generator function to wrap

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

---

Lrnr_rpart *Learner for Recursive Partitioning and Regression Trees*

Description

This learner uses rpart from the **rpart** package to fit recursive partitioning and regression trees.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- `factor_binary_outcome = TRUE`: Logical indicating whether a binary outcome should be defined as a factor instead of a numeric. This only needs to be modified to `FALSE` when the user has a binary outcome and they would like to use the mean squared error (MSE) as the splitting metric.
- `...`: Other parameters to be passed directly to `rpart` (see its documentation for details), and additional arguments defined in `Lrnr_base`, such as `formula`.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")
rpart_lrnr <- Lrnr_rpart$new()
set.seed(693)
rpart_fit <- rpart_lrnr$train(task)
```

Lrnr_rugarch *Univariate GARCH Models*

Description

This learner supports autoregressive fractionally integrated moving average and various flavors of generalized autoregressive conditional heteroskedasticity models for univariate time-series. All the models are fit using `ugarchfit`.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- variance.model: List containing variance model specification. This includes model, GARCH order, submodel, external regressors and variance tageting. Refer to ugarchspec for more information.
- mean.model: List containing the mean model specification. This includes ARMA model, whether the mean should be included, and external regressors among others.
- distribution.model: Conditional density to be used for the innovations.
- start.pars:List of staring parameters for the optimization routine.
- fixed.pars:List of parameters which are to be kept fixed during the optimization routine.
- ...: Other parameters passed to ugarchfit.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
library(origami)
library(data.table)
data(bsds)

# make folds appropriate for time-series cross-validation
folds <- make_folds(bsds,
  fold_fun = folds_rolling_window, window_size = 500,
  validation_size = 100, gap = 0, batch = 50
)

# build task by passing in external folds structure
task <- sl3_Task$new(
  data = bsds,
  folds = folds,
  covariates = c(
```

```
    "weekday", "temp"
  ),
  outcome = "cnt"
)

# create tasks for taining and validation
train_task <- training(task, fold = task$folds[[1]])
valid_task <- validation(task, fold = task$folds[[1]])

# instantiate learner, then fit and predict
HarReg_learner <- Lrnr_HarmonicReg$new(K = 7, freq = 105)
HarReg_fit <- HarReg_learner$train(train_task)
HarReg_preds <- HarReg_fit$predict(valid_task)
```

`Lrnr_screener_augment` *Augmented Covariate Screener*

Description

This learner augments a set of screened covariates with covariates that should be included by default, even if the screener did not select them.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`screener` An instantiated screener.

`default_covariates` Vector of covariate names to be automatically added to the vector selected by the screener, regardless of whether or not these covariates were selected by the screener.

`...` Other parameters passed to `screener`.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
library(data.table)

# load example data
data(cpp_imputed)
setDT(cpp_imputed)
cpp_imputed[, parity_cat := factor(ifelse(parity < 4, parity, 4))]
covars <- c(
  "apgar1", "apgar5", "parity_cat", "gagebrth", "mage", "meducyrs",
  "sexn"
)
outcome <- "haz"

# create sl3 task
task <- sl3_Task$new(data.table::copy(cpp_imputed),
  covariates = covars,
  outcome = outcome
)

screener_cor <- make_learner(
  Lrnr_screener_correlation,
  type = "rank",
  num_screen = 2
)
screener_augment <- Lrnr_screener_augment$new(screener_cor, covars)
screener_fit <- screener_augment$train(task)
selected <- screener_fit$fit_object$selected
screener_selected <- screener_fit$fit_object$screener_selected
```

Lrnr_screener_coefs *Coefficient Magnitude Screener*

Description

This learner provides screening of covariates based on the magnitude of their estimated coefficients in a (possibly regularized) GLM.

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`learner` An instantiated learner to use for estimating coefficients used in screening.

`threshold = 1e-3` Minimum size of coefficients to be kept.

`max_screen = NULL` Maximum number of covariates to be kept.

`min_screen = 2` Maximum number of covariates to be kept. Only applicable when supplied `learner` is a `Lrnr_glmnet`.

`...` Other parameters passed to `learner`.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
library(data.table)

# load example data
data(cpp_imputed)
setDT(cpp_imputed)
cpp_imputed[, parity_cat := factor(ifelse(parity < 4, parity, 4))]
covars <- c(
  "apgar1", "apgar5", "parity_cat", "gagebrth", "mage", "meducyrs",
  "sexn"
)
outcome <- "haz"

# create sl3 task
task <- sl3_Task$new(data.table::copy(cpp_imputed),
  covariates = covars,
  outcome = outcome
)

lrnr_glmnet <- make_learner(Lrnr_glmnet)
lrnr_glm <- make_learner(Lrnr_glm)
lrnr_mean <- make_learner(Lrnr_mean)
lrnrs <- make_learner(Stack, lrnr_glm, lrnr_mean)

glm_screener <- make_learner(Lrnr_screener_coefs, lrnr_glm, max_screen = 2)
glm_screener_pipeline <- make_learner(Pipeline, glm_screener, lrnrs)
```

```
fit_glm_screener_pipeline <- glm_screener_pipeline$train(task)
preds_glm_screener_pipeline <- fit_glm_screener_pipeline$predict()
```

Lrnr_screener_correlation

Correlation Screening Procedures

Description

This learner provides covariate screening procedures by running a test of correlation (Pearson default) with the `cor.test` function, and then selecting the (1) top ranked variables (default), or (2) the variables with a pvalue lower than some pre-specified threshold.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`method = 'pearson'` Correlation coefficient used for test.

`type = c('rank', 'threshold')` Screen covariates by (1) rank (default), which chooses the top `num_screen` correlated covariates; or (2) threshold, which chooses covariates with a correlation-test- based pvalue lower the threshold and a minimum of `min_screen` covariates.

`num_screen = 5` Number of covariates to select.

`pvalue_threshold = 0.1` Maximum p-value threshold. Covariates with a pvalue lower than this threshold will be retained, and at least `min_screen` most significant covariates will be selected.

`min_screen = 2` Minimum number of covariates to select. Used in pvalue_threshold screening procedure.

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_solnp_density`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Examples

```
library(data.table)

# load example data
data(cpp_imputed)
setDT(cpp_imputed)
cpp_imputed[, parity_cat := factor(ifelse(parity < 4, parity, 4))]
covars <- c(
  "apgar1", "apgar5", "parity_cat", "gagebrth", "mage", "meducyrs",
  "sexn"
)
outcome <- "haz"

# create sl3 task
task <- sl3_Task$new(data.table::copy(cpp_imputed),
  covariates = covars,
  outcome = outcome
)

lrnr_glmnet <- make_learner(Lrnr_glmnet)
lrnr_glm <- make_learner(Lrnr_glm)
lrnr_mean <- make_learner(Lrnr_mean)
lrnrs <- make_learner(Stack, lrnr_glm, lrnr_mean)

screen_corP <- make_learner(Lrnr_screener_correlation, type = "threshold")
corP_pipeline <- make_learner(Pipeline, screen_corP, lrnrs)
fit_corP <- corP_pipeline$train(task)
preds_corP_screener <- fit_corP$predict()
```

Lrnr_screener_importance

Variable Importance Screener

Description

This learner screens covariates based on their variable importance, where the importance values are obtained from the `learner`. Any learner with an `importance` method can be used. The set of learners with support for `importance` can be found with `sl3_list_learners("importance")`. Like all other screeners, this learner is intended for use in a `Pipeline`, so the output from this learner (i.e., the selected covariates) can be used as input for the next learner in the pipeline.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `learner`: An instantiated learner that supports variable importance. The set of learners with this support can be obtained via `sl3_list_learners("importance")`.
- `num_screen = 5`: The top n number of "most impotant" variables to retain.
- `...`: Other parameters passed to the `learner`'s `importance` function.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(mtcars)
mtcars_task <- sl3_Task$new(
  data = mtcars,
  covariates = c(
    "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am",
    "gear", "carb"
  ),
  outcome = "mpg"
)
glm_lrnr <- make_learner(Lrnr_glm)

# screening based on \code{\link{Lrnr_ranger}} variable importance
ranger_lrnr_importance <- Lrnr_ranger$new(importance = "impurity_corrected")
ranger_importance_screener <- Lrnr_screener_importance$new(
  learner = ranger_lrnr_importance, num_screen = 3
)
ranger_screen_glm_pipe <- Pipeline$new(ranger_importance_screener, glm_lrnr)
ranger_screen_glm_pipe_fit <- ranger_screen_glm_pipe$train(mtcars_task)

# screening based on \code{\link{Lrnr_randomForest}} variable importance
rf_lrnr <- Lrnr_randomForest$new()
rf_importance_screener <- Lrnr_screener_importance$new(
  learner = rf_lrnr, num_screen = 3
)
rf_screen_glm_pipe <- Pipeline$new(rf_importance_screener, glm_lrnr)
rf_screen_glm_pipe_fit <- rf_screen_glm_pipe$train(mtcars_task)

# screening based on \code{\link{Lrnr_randomForest}} variable importance
```

```
xgb_lrnr <- Lrnr_xgboost$new()
xgb_importance_screener <- Lrnr_screener_importance$new(
  learner = xgb_lrnr, num_screen = 3
)
xgb_screen_glm_pipe <- Pipeline$new(xgb_importance_screener, glm_lrnr)
xgb_screen_glm_pipe_fit <- xgb_screen_glm_pipe$train(mtcars_task)
```

`Lrnr_sl` *The Super Learner Algorithm*

Description

Learner that encapsulates the Super Learner algorithm. Fits metalearner on cross-validated predictions from learners. Then forms a pipeline with the learners.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `learners`: The "library" of user-specified algorithms for the super learner to consider as candidates.
- `metalearner = "default"`: The metalearner to be fit on c cross-validated predictions from the candidates. If `"default"`, the `default_metalearner` is used to construct a metalearner based on the `outcome_type` of the training `task`.
- `cv_control = NULL`: Optional list of arguments that will be used to define a specific cross-validation fold structure for fitting the super learner. Intended for use in a nested cross-validation scheme, such as cross-validated super learner (`cv_sl`) or when `Lrnr_sl` is considered in the list of candidate `learners` in another `Lrnr_sl`. Includes the arguments listed below, and any others to be passed to `fold_funs`:
  - `strata = NULL`: Discrete covariate or outcome name to define stratified cross-validation folds. If `NULL` and if `task$outcome_type$type` is binary or categorical, then the default behavior is to consider stratified cross-validation, where the strata are defined with respect to the outcome. To override the default behavior, i.e., to not consider stratified cross-validation when `strata = NULL` and `task$outcome_type$type` is binary or categorical is not `NULL`, set `strata = "none"`.
  - `cluster_by_id = TRUE`: Logical to specify clustered cross-validation scheme according to `id` in `task`. Specifically, if `task$nodes$id` is not `NULL` and if `cluster_by_id = TRUE` (default) then `task$nodes$id` is used to define a clustered cross-validation scheme, so dependent units are placed together in the same training sets and validation set. To override the default behavior, i.e., to not consider clustered cross-validation when `task$nodes$id` is not `NULL`, set `cluster_by_id = FALSE`.

  - fold_fun = NULL: A function indicating the **origami** cross-validation scheme to use, such as folds_vfold for V-fold cross-validation. See fold_funs for a list of possibilities. If NULL (default) and if other cv_control arguments are specified, e.g., V, strata or cluster_by_id, then the default behavior is to set fold_fun = origami::folds_vfold.
  - ...: Other arguments to be passed to fold_fun, such as V for fold_fun = folds_vfold. See fold_funs for a list fold-function-specific possible arguments.
- keep_extra = TRUE: Stores all sub-parts of the super learner computation. When FALSE, the resulting object has a memory footprint that is significantly reduced through the discarding of intermediary data structures.
- verbose = NULL: Whether to print cv_control-related messages. Warnings and errors are always printed. When verbose = NULL, verbosity specified by option sl3.verbose will be used, and the default sl3.verbose option is FALSE. (Note: to turn on sl3.verbose option, set options("sl3.verbose" = TRUE).)
- ...: Any additional parameters that can be considered by Lrnr_base.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
## Not run:
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")
# this is just for illustrative purposes, not intended for real applications
# of the super learner!
glm_lrn <- Lrnr_glm$new()
ranger_lrn <- Lrnr_ranger$new()
lasso_lrn <- Lrnr_glmnet$new()
eSL <- Lrnr_sl$new(learners = list(glm_lrn, ranger_lrn, lasso_lrn))
eSL_fit <- eSL$train(task)
# example with cv_control, where Lrnr_sl included as a candidate
eSL_nested5folds <- Lrnr_sl$new(
  learners = list(glm_lrn, ranger_lrn, lasso_lrn),
  cv_control = list(V = 5),
  verbose = FALSE
)
```

```
dSL <- Lrnr_sl$new(
  learners = list(glm_lrn, ranger_lrn, lasso_lrn, eSL_nested5folds),
  metalearner = Lrnr_cv_selector$new(loss_squared_error)
)
dSL_fit <- dSL$train(task)
# example with cv_control, where we use cross-validated super learner
cvSL_fit <- cv_sl(
  lrnr_sl = eSL_nested5folds, task = task, eval_fun = loss_squared_error
)

## End(Not run)
```

`Lrnr_solnp` *Nonlinear Optimization via Augmented Lagrange*

Description

This meta-learner provides fitting procedures for any pairing of loss or risk function and metalearner function, subject to constraints. The optimization problem is solved by making use of `solnp`, using Lagrange multipliers. An important note from the `solnp` documentation states that the control parameters `tol` and `delta` are key in getting any possibility of successful convergence, therefore it is suggested that the user change these appropriately to reflect their problem specification. For further details, consult the documentation of the **Rsolnp** package.

Format

An `R6Class` object inheriting from `Lrnr_base`.

Value

A learner object inheriting from `Lrnr_base` with methods for training and prediction. For a full list of learner functionality, see the complete documentation of `Lrnr_base`.

Parameters

- `learner_function` = `metalearner_linear`: A function(alpha, X) that takes a vector of covariates and a matrix of data and combines them into a vector of predictions. See `metalearners` for options.
- `eval_function` = `loss_squared_error`: A function(pred, truth) that takes prediction and truth vectors and returns a loss vector or a risk scalar. See `loss_functions` and `risk_functions` for options and more detail.
- `make_sparse` = `TRUE`: If `TRUE`, zeros out small alpha values.
- `convex_combination` = `TRUE`: If `TRUE`, constrain alpha to sum to 1.
- `init_0` = `FALSE`: If `TRUE`, alpha is initialized to all 0's, useful for TMLE. Otherwise, it is initialized to equal weights summing to 1, useful for Super Learner.

- rho = 1: This is used as a penalty weighting scaler for infeasibility in the augmented objective function. The higher its value the more the weighting to bring the solution into the feasible region (default 1). However, very high values might lead to numerical ill conditioning or significantly slow down convergence.
- outer.iter = 400: Maximum number of major (outer) iterations.
- inner.iter = 800: Maximum number of minor (inner) iterations.
- delta = 1e-7:Relative step size in forward difference evaluation.
- tol = 1e-8: Relative tolerance on feasibility and optimality.
- trace = FALSE: The value of the objective function and the parameters are printed at every major iteration.
- ...: Additional arguments defined in Lrnr_base, such as params (like formula) and name.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
# define ML task
data(cpp_imputed)
covs <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs")
task <- sl3_Task$new(cpp_imputed, covariates = covs, outcome = "haz")

# build relatively fast learner library (not recommended for real analysis)
lasso_lrnr <- Lrnr_glmnet$new()
glm_lrnr <- Lrnr_glm$new()
ranger_lrnr <- Lrnr_ranger$new()
lrnrs <- c(lasso_lrnr, glm_lrnr)
names(lrnrs) <- c("lasso", "glm")
lrnr_stack <- make_learner(Stack, lrnrs)

# instantiate SL with solnp metalearner
solnp_meta <- Lrnr_solnp$new()
sl <- Lrnr_sl$new(lrnr_stack, solnp_meta)
sl_fit <- sl$train(task)
```

`Lrnr_solnp_density` *Nonlinear Optimization via Augmented Lagrange*

Description

This meta-learner provides fitting procedures for density estimation, finding convex combinations of candidate density estimators by minimizing the cross-validated negative log-likelihood loss of each candidate density. The optimization problem is solved by making use of `solnp`, using Lagrange multipliers. For further details, consult the documentation of the `Rsolnp` package.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`...` Not currently used.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: `Custom_chain`, `Lrnr_HarmonicReg`, `Lrnr_arima`, `Lrnr_bartMachine`, `Lrnr_base`, `Lrnr_bayesglm`, `Lrnr_caret`, `Lrnr_cv`, `Lrnr_cv_selector`, `Lrnr_dbarts`, `Lrnr_define_interactions`, `Lrnr_density_discretize`, `Lrnr_density_hse`, `Lrnr_density_semiparametric`, `Lrnr_earth`, `Lrnr_expSmooth`, `Lrnr_ga`, `Lrnr_gam`, `Lrnr_gbm`, `Lrnr_glm`, `Lrnr_glm_fast`, `Lrnr_glm_semiparametric`, `Lrnr_glmnet`, `Lrnr_glmtree`, `Lrnr_grf`, `Lrnr_grfcate`, `Lrnr_gru_keras`, `Lrnr_h2o_grid`, `Lrnr_hal9001`, `Lrnr_haldensify`, `Lrnr_independent_binomial`, `Lrnr_lightgbm`, `Lrnr_lstm_keras`, `Lrnr_mean`, `Lrnr_multiple_ts`, `Lrnr_multivariate`, `Lrnr_nnet`, `Lrnr_nnls`, `Lrnr_optim`, `Lrnr_pca`, `Lrnr_pkg_SuperLearner`, `Lrnr_polspline`, `Lrnr_pooled_hazards`, `Lrnr_randomForest`, `Lrnr_ranger`, `Lrnr_revere_task`, `Lrnr_rpart`, `Lrnr_rugarch`, `Lrnr_screener_augment`, `Lrnr_screener_coefs`, `Lrnr_screener_correlation`, `Lrnr_screener_importance`, `Lrnr_sl`, `Lrnr_solnp`, `Lrnr_stratified`, `Lrnr_subset_covariates`, `Lrnr_svm`, `Lrnr_tsDyn`, `Lrnr_ts_weights`, `Lrnr_xgboost`, `Pipeline`, `Stack`, `define_h2o_X()`, `undocumented_learner`

Lrnr_stratified *Stratify learner fits by a single variable*

Description

Stratify learner fits by a single variable

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`learner="learner"` An initialized Lrnr_* object.

`variable_stratify="variable_stratify"` `character` giving the variable in the covariates on which to stratify. Supports only variables with discrete levels coded as `numeric`.

`...` Other parameters passed directly to `learner$train`. See its documentation for details.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
library(data.table)

# load example data set
data(cpp_imputed)
setDT(cpp_imputed)

# use covariates of intest and the outcome to build a task object
covars <- c("apgar1", "apgar5", "sexn")
task <- sl3_Task$new(cpp_imputed, covariates = covars, outcome = "haz")
```

```
hal_lrnr <- Lrnr_hal9001$new(fit_control = list(n_folds = 3))
stratified_hal <- Lrnr_stratified$new(
  learner = hal_lrnr,
  variable_stratify = "sexn"
)

# stratified learner
set.seed(123)
stratified_hal_fit <- stratified_hal$train(task)
stratified_prediction <- stratified_hal_fit$predict(task = task)
```

`Lrnr_subset_covariates`

Learner with Covariate Subsetting

Description

This learner provides fitting procedures for subsetting covariates. It is a convenience utility for reducing the number of covariates to be fit.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`...` Not currently used.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
# load example data
data(cpp_imputed)
covars <- c("apgar1", "apgar5", "parity", "gagebrth", "mage", "meducyrs", "sexn")
outcome <- "haz"

# create sl3 task
task <- sl3_Task$new(data.table::copy(cpp_imputed),
  covariates = covars,
  outcome = outcome,
  folds = origami::make_folds(cpp_imputed, V = 3)
)

glm_learner <- Lrnr_glm$new()
glmnet_learner <- Lrnr_glmnet$new()
subset_apgar <- Lrnr_subset_covariates$new(covariates = c("apgar1", "apgar5"))
learners <- list(glm_learner, glmnet_learner, subset_apgar)
sl <- make_learner(Lrnr_sl, learners, glm_learner)

sl_fit <- sl$train(task)
sl_pred <- sl_fit$predict()
```

Lrnr_svm *Support Vector Machines*

Description

This learner provides fitting procedures for support vector machines, using the routines from **e1071** (described in Meyer et al. (2021) and Chang and Lin (2011), the core library to which **e1071** is an interface) through a call to the function svm.

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- scale = TRUE: A logical vector indicating the variables to be scaled. For a detailed description, please consult the documentation for svm.
- type = NULL: SVMs can be used as a classification machine, as a a regression machine, or for novelty detection. Depending of whether the outcome is a factor or not, the default setting for this argument is "C-classification" or "eps-regression", respectively. This may be overwritten by setting an explicit value. For a full set of options, please consult the documentation for svm.
- kernel = "radial": The kernel used in training and predicting. You may consider changing some of the optional parameters, depending on the kernel type. Kernel options include: "linear", "polynomial", "radial" (the default), "sigmoid". For a detailed description, consult the documentation for svm.
- fitted = TRUE: Logical indicating whether the fitted values should be computed and included in the model fit object or not.
- probability = FALSE: Logical indicating whether the model should allow for probability predictions.
- ...: Other parameters passed to svm. See its documentation for details.

References

Chang C, Lin C (2011). "LIBSVM: A library for support vector machines." *ACM Transactions on Intelligent Systems and Technology*, **2**(3), 27:1–27:27. Software available at https://www.csie.ntu.edu.tw/~cjlin/libsvm/.

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2021). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.7-6, https://CRAN.R-project.org/package=e1071.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(mtcars)
# create task for prediction
mtcars_task <- sl3_Task$new(
  data = mtcars,
  covariates = c(
    "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am",
    "gear", "carb"
  ),
  outcome = "mpg"
)
# initialization, training, and prediction with the defaults
svm_lrnr <- Lrnr_svm$new()
svm_fit <- svm_lrnr$train(mtcars_task)
svm_preds <- svm_fit$predict()
```

Lrnr_tsDyn *Nonlinear Time Series Analysis*

Description

This learner supports various forms of nonlinear autoregression, including additive AR, neural nets, SETAR and LSTAR models, threshold VAR and VECM.

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`learner` Available built-in time series models. Currently available can be listed with availableModels() function.

`m = 1` embedding dimension.

`...` Additional learner-specific arguments.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner,

Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

---

Lrnr_ts_weights *Time-specific weighting of prediction losses*

Description

A wrapper around any learner that reweights observations. This reweighted is intended for time series, and ultimately assigns weights to losses. This learner is particularly useful as a metalearner wrapper. It can be used to create a time-adaptive ensemble, where a super learner is created in a manner that places more weight (with max weight of 1) on recent losses, and less weight is placed on losses further in the past.

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction

Parameters

`learner` The learner to wrap

`folds=NULL` An `origami` folds object. If `NULL`, folds from the task are used

`full_fit=FALSE` If `TRUE`, also fit the underlying learner on the full data. This can then be accessed with `predict_fold(task, fold_number="full")`

`window` Observations corresponding to times outside of the window are assigned weight of 0, and obervations corresponding to times within the window are assigned weight of 1. The window is defined with respect to the difference from the maximum time, where all times are obtained from the task node for time. For example, if the maximum time is 100 and the window is 10, then obervations corresponding to times 90-100 are assigned weight 1 and obervations for times 1-89 are assigned weight 0. If `rate` is provided with `window`, then times within the window are assigned according to the `rate` argument (and potentially `delay_decay`), and the times outside of the window are still assigned weight of 0.

`rate` A rate of decay to apply to the losses, where the decay function is (1-rate)^lag and the lag is the difference from all times to the maximum time.

`delay_decay` The amount of time to delay decaying weights, for optional use with `rate` argument. The delay decay is subtracted from the lags, such that lags less than the delay decay have lag of 0 and thus weight of 1. For example, a delay decay of 10 assigns weight 1 to observations that are no more than 10 time points away from the maximum time; and for observations that are more than 10 time points away from the maximum time, the weight is assigned according to

the decay function. In this example, observations corresponding to 11 time points away from the maximum time would be assigned lag=1, 11-10, when setting the weights with respect to (1-rate)^lag.

... Not currently used.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_xgboost, Pipeline, Stack, define_h2o_X(), undocumented_learner

---

Lrnr_xgboost *xgboost: eXtreme Gradient Boosting*

Description

This learner provides fitting procedures for xgboost models, using the **xgboost** package, via xgb.train. Such models are classification and regression trees with extreme gradient boosting. For details on the fitting procedure, consult the documentation of the **xgboost** and Chen and Guestrin (2016)).

Format

An R6Class object inheriting from Lrnr_base.

Value

A learner object inheriting from Lrnr_base with methods for training and prediction. For a full list of learner functionality, see the complete documentation of Lrnr_base.

Parameters

- nrounds=20: Number of fitting iterations.
- ...: Other parameters passed to xgb.train.

References

Chen T, Guestrin C (2016). "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 785–794.

See Also

Lrnr_gbm for standard gradient boosting models (via the **gbm** package) and Lrnr_lightgbm for the faster and more efficient gradient boosted trees from the LightGBM framework (via the **lightgbm** package).

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Pipeline, Stack, define_h2o_X(), undocumented_learner

Examples

```
data(mtcars)
mtcars_task <- sl3_Task$new(
  data = mtcars,
  covariates = c(
    "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am",
    "gear", "carb"
  ),
  outcome = "mpg"
)

# initialization, training, and prediction with the defaults
xgb_lrnr <- Lrnr_xgboost$new()
xgb_fit <- xgb_lrnr$train(mtcars_task)
xgb_preds <- xgb_fit$predict()

# get feature importance from fitted model
xgb_varimp <- xgb_fit$importance()
```

---

make_learner_stack *Make a stack of sl3 learners*

Description

Produce a stack of learners by passing in a list with IDs for the learners. The resultant stack of learners may then be used as normal.

Usage

```
make_learner_stack(...)
```

Arguments

...	Each argument is a list that will be passed to make_learner

Value

An sl3 Stack consisting of the learners passed in as arguments the list argument to this function. This Stack has all of the standard methods associated with such objects.

Examples

```
# constructing learners with default settings
sl_stack_easy <- make_learner_stack(
  "Lrnr_mean", "Lrnr_glm_fast",
  "Lrnr_xgboost"
)

# constructing learners with arguments passed in
sl_stack <- make_learner_stack(
  "Lrnr_mean",
  list("Lrnr_hal9001",
    n_folds = 10,
    use_min = TRUE
  )
)
```

---

metalearners *Combine predictions from multiple learners*

Description

Combine predictions from multiple learners

Usage

```
metalearner_logistic_binomial(alpha, X, trim)

metalearner_linear(alpha, X)

metalearner_linear_multivariate(alpha, X)

metalearner_linear_multinomial(alpha, X)
```

Arguments

alpha	a vector of combination coefficients
X	a matrix of predictions
trim	a value use to trim predictions away from 0 and 1.

pack_predictions *Pack multidimensional predictions into a vector (and unpack again)*

Description

Pack multidimensional predictions into a vector (and unpack again)

Usage

```
pack_predictions(pred_matrix)

unpack_predictions(x)

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

normalize_rows(x)
```

Arguments

pred_matrix	a matrix of prediciton values
x	a packed prediction list
...	ignored

Pipeline *Pipeline (chain) of learners.*

Description

A Pipeline of learners is a way to "chain" Learners together, where the output of one learner is used as output for the next learner. This can be used for things like screening, two stage machine learning methods, and Super Learning. A pipeline is fit by fitting the first `Learner`, calling `chain()` to create the next task, which becomes the training data for the next `Learner`. Similarly, for prediction, the predictions from the first `Learner` become the data to predict on for the next `Learner`.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

... Parameters should be individual `Learners`, in the order they should be applied.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by Lrnr_base, and shared by all learners.

covariates A character vector of covariates. The learner will use this to subset the covariates for any specified task

outcome_type A variable_type object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

... All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Stack, define_h2o_X(), undocumented_learner

---

pooled_hazard_task *Generate A Pooled Hazards Task from a Failure Time (or Categorical) Task*

Description

Generate A Pooled Hazards Task from a Failure Time (or Categorical) Task

Usage

```
pooled_hazard_task(task, trim = TRUE)
```

Arguments

task	A sl3_Task where the outcome is failure time.
trim	If true, remove entries after failure time for each observation.

prediction_plot *Plot predicted and true values for diganostic purposes*

Description

If a Lrnr_sl fit is provided, predictions will be generated from the cross-validated learner fits and final metalearner fit. Otherwise, non cross-validated predictions will be used an an error will be thrown

Usage

```
prediction_plot(learner_fit)
```

Arguments

learner_fit	A fit sl3 learner object. Ideally from a Lrnr_sl

Value

A ggplot2 object

predict_classes *Predict Class from Predicted Probabilities*

Description

Returns the most likely class label for each row of predicted class probabilities

Usage

```
predict_classes(predictions)
```

Arguments

predictions	the nxc matrix where each row are predicted probabilities for one observation for each of c classes.

Value

a vector of length n, the predicted class labels as a factor variable

process_data *Process Data*

Description

A function called upon creating a task that uses the data provided to the task in order to process the covariates and identify missingness in the outcome. See parameters and details for more information.

Usage

```
process_data(data, nodes, column_names, flag = TRUE,
  drop_missing_outcome = FALSE)
```

Arguments

data	A data.table containing the analytic dataset. In creating the sl3_Task, the data passed to the task is supplied for this argument.
nodes	A list of character vectors for covariates, outcome, id, weights, and offset, which is generated when creating the sl3_Task if not already specified as an argument to make_sl3_Task.
column_names	A named list of column names in the data, which is generated when creating the sl3_Task if not already specified as an argument to make_sl3_Task.
flag	Logical (default TRUE) indicating whether to notify the user when there are outcomes that are missing, which can be modified when creating the sl3_Task by setting flag = FALSE.
drop_missing_outcome	Logical (default FALSE) indicating whether to drop observations with missing outcomes, which can be modified when creating the sl3_Task by setting drop_missing_outcome = TRUE.

Details

If the data provided to the task contains missing covariate values, then a few things will happen. First, for each covariate with missing values, if the proportion of missing values is greater than getOption("sl3.max_p_missing"), the covariate will be dropped. (The default option "sl3.max_p_missing" is 0.5 and it can be modified to say, 0.75, by setting options("sl3.max_p_missing" = 0.75)). Also, for each covariate with missing values that was not dropped, a so-called "missingness indicator" (that takes the name of the covariate with prefix "delta_") will be added as an additional covariate. The missingness indicator will take a value of 0 if the covariate value was missing and 1 if not. Also, imputation will be performed for each covariate with missing values: continuous covariates are imputed with the median, and discrete covariates are imputed with the mode. This coupling of imputation and missingness indicators removes the missing covariate values, while preserving the pattern of missingness, respectively. To avoid this default imputation, users can perform imputation on their analytic dataset before supplying it to make_sl3_Task. We generally recommend the missingness indicators be added regardless of the imputation strategy, unless missingness is very rare.

This function also coverts any character covariates to factors, and one-hot encodes factor covariates.

Lastly, if the `outcome` is supplied in creating the `sl3_Task` and if missing outcome values are detected in `data`, then a warning will be thrown. If `drop_missing_outcome = TRUE` then observations with missing outcomes will be dropped.

Value

A list of processed data, nodes and column names

`risk` *Risk Estimation*

Description

Estimates a risk for a given set of predictions and loss function.

Usage

```
risk(pred, observed, loss = loss_squared_error, weights = NULL)
```

Arguments

`pred`	A vector of predicted values.
`observed`	A vector of observed values.
`loss`	A loss function. For options, see loss_functions.
`weights`	A vector of weights.

`risk_functions` *FACTORY RISK FUNCTION FOR ROCR PERFORMANCE MEASURES WITH BINARY OUTCOMES*

Description

Factory function for estimating an ROCR-based risk for a given ROCR measure, and the risk is defined as one minus the performance measure.

Usage

```
custom_ROCR_risk(measure, cutoff = 0.5, name = NULL, ...)
```

Arguments

measure	A character indicating which ROCR performance measure to use for evaluation. The measure must be either cutoff-dependent so a single value can be selected (e.g., "tpr"), or it's value is a scalar (e.g., "aucpr"). For more information, see performance.
cutoff	A numeric value specifying the cutoff for choosing a single performance measure from the returned set. Only used for performance measures that are cutoff-dependent and default is 0.5. See performance for more detail.
name	An optional character string for user to supply their desired name for the performance measure, which will be used for naming subsequent risk-related tables and metrics (e.g., cv_risk column names). When name is not supplied, the measure will be used for naming.
...	Optional arguments to specific ROCR performance measures. See performance for more detail.

Note

This risk does not take into account weights. In order to use this risk, it must first be instantiated with respect to the **ROCR** performance measure of interest, and then the user-defined function can be used.

---

safe_dim *dim that works for vectors too*

Description

safe_dim tries to get dimensions from dim and falls back on length if dim returns NULL

Usage

```
safe_dim(x)
```

Arguments

x	the object to get dimensions from

Shared_Data *Container Class for data.table Shared Between Tasks*

Description

Mostly to deal with alloc.col shallow copies, but also nice to have a bit more abstraction.

sl3Options *Querying/setting a single* sl3 *option*

Description

To list all sl3 options, just run this function without any parameters provided. To query only one value, pass the first parameter. To set that, use the value parameter too.

Usage

```
sl3Options(o, value)
```

Arguments

o	Option name (string).
value	Value to assign (optional)

Examples

```
## Not run:
sl3Options()
sl3Options("sl3.verbose")
sl3Options("sl3.temp.dir")
sl3Options("sl3.verbose", TRUE)

## End(Not run)
#
```

sl3_list_properties *List sl3 Learners*

Description

Lists learners in sl3 (defined as objects that start with Lrnr_ and inherit from Lrnr_base)

Usage

```
sl3_list_properties()

sl3_list_learners(properties = c())
```

Arguments

properties	a vector of properties that learners must match to be returned

Value

a vector of learner names that match the property list

sl3_revere_Task *Revere (SplitSpecific) Task*

Description

A task that has different realizations in different folds Useful for Revere CV operations

Details

Learners with property "cv" must use these tasks correctly

Other learners will treat this as the equivalent of the "full" task.

sl3_Task *Define a Machine Learning Task*

Description

An increasingly thick wrapper around a `data.table` containing the data for a prediction task. This contains metadata about the particular machine learning problem, including which variables are to be used as covariates and outcomes.

Usage

```
make_sl3_Task(...)
```

Arguments

`...`	Passes all arguments to the constructor. See documentation for Constructor below.

Format

`R6Class` object.

Value

`sl3_Task` object

Constructor

`make_sl3_Task(data, covariates, outcome = NULL, outcome_type = NULL, outcome_levels = NULL, id = NULL, weights = NULL, offset = NULL, nodes = NULL, column_names = NULL, folds = NULL, drop_missing_outcome = FALSE, flag = TRUE)`

`data` A `data.frame` or `data.table` containing the analytic dataset.

`covariates` A character vector of variable names that define the set of covariates.

`outcome` A character vector of variable names that define the set of outcomes. Usually just one variable, although some learners support multivariate outcomes. Use `sl3_list_learners("multivariate_outcome")` to find such learners.

`outcome_type` A `Variable_type` object that defines the variable type of the outcome. Alternatively, a character specifying such a type. See `variable_type` for details on defining variable types.

`outcome_levels` A vector of levels expected for the outcome variable. If `outcome_type` is a character, this will be used to construct an appropriate `variable_type` object.

`id` A character indicating which variable (if any) to be used as an identifier for independent observations, which would be necessary if there are clusters of dependent units in the data (e.g., repeated measures on the same individual). The `id` is used to define a clustered cross-validation scheme (if `folds` is not already supplied to `make_sl3_Task`), for learners that use cross-validation as part of their fitting procedure. Use `sl3_list_learners("ids")` to find learners whose fitting procedures support clustered observations, and use `sl3_list_learners("cv")` to find learners whose fitting procedures involve cross-validation.

`weights` A character indicating which variable (if any) to be used as observation weights, for learners that support that. Use `sl3_list_learners("weights")` to find such learners.

`offset` A character indicating which variable (if any) to be used as an observation offset, for learners that support that. Use `sl3_list_learners("offset")` to find such learners.

`nodes` A list of character vectors as nodes. This will override the `covariates`, `outcome`, `id`, `weights`, and `offset` arguments if specified, serving as an alternative way to specify those arguments.

`column_names` A named list of characters that maps between column names in `data` and how those variables are referenced in `sl3_Task` functions.

`drop_missing_outcome` Logical indicating whether to drop outcomes that are missing.

`flag` Logical indicating whether to notify the user when there are outcomes that are missing.

`folds` An optional origami fold object, as generated by `make_folds`, specifying a cross-validation scheme. If `NULL` (default), a V-fold cross-validation scheme with V = 10 will be considered for learners that use cross-validation as part of their fitting procedure. Also, if `NULL` (default) and `id` is specified, then a clustered V-fold cross-validation procedure with 10 folds will be considered. Use `sl3_list_learners("cv")` to find learners whose fitting procedures involve cross-validation.

Methods

`add_interactions(interactions, warn_on_existing = TRUE)` Adds interaction terms to task, returns a task with interaction terms added to covariate list.

- interactions: A list of lists, where each sublist describes one interaction term, listing the variables that comprise it
- warn_on_existing: If TRUE, produce a warning if there is already a column with a name matching this interaction term

add_columns(fit_uuid, new_data, global_cols=FALSE) Add columns to internal data, returning an updated vector of column_names

- fit_uuid: A uuid character that is used to generate unique internal column names. This prevents two added columns with the same name overwriting each other, provided they have different fit_uuid.
- new_data: A data.table containing the columns to add
- global_cols: If true, don't use the fit_uuid to make unique column names

next_in_chain(covariates=NULL, outcome=NULL, id=NULL, weights=NULL, offset=NULL, column_names=NULL, ne
Used by learner$chain methods to generate a task with the same underlying data, but redefined nodes. Most of the parameter values are passed to the sl3_Task constructor, documented above.

- covariates: An updated covariates character vector
- outcome: An updated outcome character vector
- id: An updated id character value
- weights: An updated weights character value
- offset: An updated offset character value
- column_names: An updated column_names character vector
- new_nodes: An updated list of node names
- ...: Other arguments passed to the sl3_Task constructor for the new task

subset_task(row_index) Returns a task with rows subsetted using the row_index index vector

- row_index: An index vector defining the subset

get_data(rows, columns) Returns a data.table containing a subset of task data.

- rows: An index vector defining the rows to return
- columns: A character vector of columns to return.

has_node(node_name) Returns true if the node is defined in the task

- node_name: The name of the node to look for

get_node(node_name, generator_fun=NULL) Returns a ddta.table with the requested node's data

- node_name: The name of the node to look for
- generator_fun: A function(node_name, n) that can generate the node if it was not specified in the task.

Fields

raw_data Internal representation of the data

data Formatted task data

nrow Number of observations

nodes A list of node variables

`X` a data.table containing the covariates

`X` a data.table containing the covariates and an intercept term

`Y` a vector containing the outcomes

`offsets` a vector containing the offset. Will return an error if the offset wasn't specified on construction

`weights` a vector containing the observation weights. If weights aren't specified on construction, weights will default to 1

`id` a vector containing the observation units. If the ids aren't specified on construction, id will return seq_len(nrow)

`folds` An origami fold object, as generated by `make_folds`, specifying a cross-validation scheme

`uuid` A unique identifier of this task

`column_names` The named list mapping variable names to internal column names

`outcome_type` A `variable_type` object specifying the type of the outcome

`Stack` *Learner Stacking*

Description

A Stack is a special Learner that combines multiple other learners, "stacking" their predictions in columns.

Format

`R6Class` object.

Value

`Lrnr_base` object with methods for training and prediction

Parameters

`...` Parameters should be individual `Learners`.

Common Parameters

Individual learners have their own sets of parameters. Below is a list of shared parameters, implemented by `Lrnr_base`, and shared by all learners.

`covariates` A character vector of covariates. The learner will use this to subset the covariates for any specified task

`outcome_type` A `variable_type` object used to control the outcome_type used by the learner. Overrides the task outcome_type if specified

`...` All other parameters should be handled by the invidual learner classes. See the documentation for the learner class you're instantiating

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, define_h2o_X(), undocumented_learner

---

subset_folds *Make folds work on subset of data*

Description

subset_folds takes a origami style folds list, and returns a list of folds applicable to a subset, by subsetting the training and validation index vectors

Usage

```
subset_folds(folds, subset)
```

Arguments

folds	an origami style folds list
subset	an index vector to be used to subset the data

---

train_task *Subset Tasks for CV THe functions use origami folds to subset tasks. These functions are used by Lrnr_cv (and therefore other learners that use Lrnr_cv). So that nested cv works properly, currently the subsetted task objects do not have fold structures of their own, and so generate them from defaults if nested cv is requested.*

Description

Subset Tasks for CV THe functions use origami folds to subset tasks. These functions are used by Lrnr_cv (and therefore other learners that use Lrnr_cv). So that nested cv works properly, currently the subsetted task objects do not have fold structures of their own, and so generate them from defaults if nested cv is requested.

Usage

```
train_task(task, fold)

validation_task(task, fold)
```

Arguments

task	a task to subset
fold	an origami fold object to use for subsetting

---

undocumented_learner *Undocumented Learner*

Description

We haven't documented this one yet. Feel free to contribute!

Format

R6Class object.

Value

Lrnr_base object with methods for training and prediction.

Fields

params A list of parameters needed to fully specify the learner. This includes things like model hyperparameters.

See Also

Other Learners: Custom_chain, Lrnr_HarmonicReg, Lrnr_arima, Lrnr_bartMachine, Lrnr_base, Lrnr_bayesglm, Lrnr_caret, Lrnr_cv, Lrnr_cv_selector, Lrnr_dbarts, Lrnr_define_interactions, Lrnr_density_discretize, Lrnr_density_hse, Lrnr_density_semiparametric, Lrnr_earth, Lrnr_expSmooth, Lrnr_ga, Lrnr_gam, Lrnr_gbm, Lrnr_glm, Lrnr_glm_fast, Lrnr_glm_semiparametric, Lrnr_glmnet, Lrnr_glmtree, Lrnr_grf, Lrnr_grfcate, Lrnr_gru_keras, Lrnr_h2o_grid, Lrnr_hal9001, Lrnr_haldensify, Lrnr_independent_binomial, Lrnr_lightgbm, Lrnr_lstm_keras, Lrnr_mean, Lrnr_multiple_ts, Lrnr_multivariate, Lrnr_nnet, Lrnr_nnls, Lrnr_optim, Lrnr_pca, Lrnr_pkg_SuperLearner, Lrnr_polspline, Lrnr_pooled_hazards, Lrnr_randomForest, Lrnr_ranger, Lrnr_revere_task, Lrnr_rpart, Lrnr_rugarch, Lrnr_screener_augment, Lrnr_screener_coefs, Lrnr_screener_correlation, Lrnr_screener_importance, Lrnr_sl, Lrnr_solnp, Lrnr_solnp_density, Lrnr_stratified, Lrnr_subset_covariates, Lrnr_svm, Lrnr_tsDyn, Lrnr_ts_weights, Lrnr_xgboost, Pipeline, Stack, define_h2o_X()

Variable_Type *Specify Variable Type*

Description

Specify Variable Type

Usage

```
variable_type(type = NULL, levels = NULL, bounds = NULL, x = NULL,
  pcontinuous = getOption("sl3.pcontinuous"))
```

Arguments

type	A type name. Valid choices include "binomial", "categorical", "continuous", and "multivariate". When not specified, this is inferred.
levels	Valid levels for discrete types.
bounds	Bounds for continuous variables.
x	Data to use for inferring type if not specified.
pcontinuous	If type above is inferred, the proportion of unique observations above which the variable is considered continuous.

write_learner_template *Generate a file containing a template* sl3 *Learner*

Description

Generates a template file that can be used to write new **sl3** Learners. For more information, see the Defining New Learners vignette.

Usage

```
write_learner_template(file)
```

Arguments

file	the path where the file should be written

Value

the return from file.copy. TRUE if writing the template was successful.

Index