Package ‘nadir’

February 20, 2026

Type Package

Title Super Learning with Flexible Formulas

Version 0.0.1

Description A functional programming based
implementation of the super learner algorithm with an emphasis on supporting
the use of formulas to specify learners. This approach offers several
improvements compared to past implementations including the ability to
easily use random-effects specified in formulas
(like y ~ (age | strata) + ...) and construction of new learners is
as simple as writing and passing a new function. The
super learner algorithm was originally described in van der Laan et al.
(2007) <`https://biostats.bepress.com/ucbbiostat/paper222/`>.

License MIT + file LICENSE

Depends R (>= 4.2.0)

Encoding UTF-8

RoxygenNote 7.3.2

Suggests MASS, ggplot2, knitr, palmerpenguins, rmarkdown, survival,
testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

URL `https://ctesta01.github.io/nadir/`,
`https://github.com/ctesta01/nadir/`

BugReports `https://github.com/ctesta01/nadir/issues`

Imports dplyr, earth, future, future.apply, gbm, glmnet, hal9001,
lifecycle, lme4, methods, mgcv, nnet, nnls, origami,
randomForest, ranger, tibble, tidyr, VGAM, xgboost

NeedsCompilation no

Author Christian Testa [aut, cre] (ORCID:
<`https://orcid.org/0000-0001-9103-5839`>),
Nima Hejazi [ths, aut] (ORCID: <`https://orcid.org/0000-0002-7127-2789`>)

Maintainer Christian Testa <ctesta@hsph.harvard.edu>

Repository CRAN

Date/Publication 2026-02-20 10:30:07 UTC

Contents

`add_screener` *Add a Screener to a Learner*

Description

Add a Screener to a Learner

Usage

```
add_screener(learner, screener, screener_extra_args = NULL)
```

Arguments

`learner`	A learner to be modified by wrapping a screening stage on top of it.
`screener`	A screener to be added on top of the learner
`screener_extra_args`	Extra arguments to be passed to the screener

Value

A modified learner that when called on data and a formula now runs a screening stage before fitting the learner and returning a prediction function.

Examples

```
# construct a learner where variables with less than .6 correlation are screened out
lnr_glm_with_cor_60_thresholding <-
  add_screener(
    learner = lnr_glm,
    screener = screener_cor,
    screener_extra_args = list(threshold = .6)
  )

# train that on the mtcars dataset - also checking that extra arguments are properly passed to glm
lnr_glm_with_cor_60_thresholding(mtcars, formula = mpg ~ ., family = "gaussian")(mtcars)

# if we've screened out variables with low correlation to mpg, one such variable is qsec,
# so changing qsec shouldn't modify the predictions from our learned algorithm
mtcars_but_qsec_is_changed <- mtcars
```

```
mtcars_but_qsec_is_changed$qsec <- rnorm(n = nrow(mtcars))

identical(
  lnr_glm_with_cor_60_thresholding(mtcars, formula = mpg ~ .)(mtcars),
 lnr_glm_with_cor_60_thresholding(mtcars, formula = mpg ~ .)(mtcars_but_qsec_is_changed)
 )

# earth version
lnr_earth_with_cor_60_thresholding <-
  add_screener(
    learner = lnr_earth,
    screener = screener_cor,
    screener_extra_args = list(threshold = .6)
  )
lnr_earth_with_cor_60_thresholding(mtcars, formula = mpg ~ .)(mtcars)

identical(
  lnr_earth_with_cor_60_thresholding(mtcars, formula = mpg ~ .)(mtcars),
  lnr_earth_with_cor_60_thresholding(mtcars, formula = mpg ~ .)(mtcars)
 )

# note that this 'test' does not pass for a learner like randomForest that has
# some randomness in its predictions.
```

---

binary_learners *Binary Learners in* {nadir}

Description

- lnr_nnet
- lnr_rf_binary
- lnr_logistic

Details

The important thing to know about binary learners is that they need to produce predictions that the outcome is == 1 or TRUE.

Also, for binary outcomes, we should make sure to use the determine_weights_for_binary_outcomes in our calls to super_learner() which calculates the estimated probability of the observed outcome (either 0 or 1) and then applies the negative log loss function afterwards. This can be done automatically by declaring outcome_type = 'binary' in calling super_learner()

See Also

density_learners learners

Examples

```
super_learner(
  data = mtcars,
  learners = list(logistic1 = lnr_logistic, logistic2 = lnr_logistic, lnr_rf_binary),
  formulas = list(
  .default = am ~ .,
  logistic2 = am ~ mpg * hp + .),
  outcome_type = 'binary'
  )
```

`compare_learners` *Compare Learners*

Description

Compare learners using the specified `loss_metric`

Usage

```
compare_learners(sl_output, y_variable = NULL, loss_metric)
```

Arguments

`sl_output`	Output from running `super_learner()` with `verbose_output` = `TRUE`.
`y_variable`	A character vector indicating the outcome variable. `y_variable` will be automatically inferred if it is missing and can be inferred from the `sl_output`.
`loss_metric`	A loss metric, like the mean-squared-error or negative-log-loss to be used in comparing the learners. A loss metric should take two (vector) arguments: predictions, and true outcomes, and produce a single statistic summarizing the performance of each learner.

Value

A data.frame with the loss-metric on the held-out data for each learner.

Examples

```
sl_model <- super_learner(
  data = mtcars,
  learners = list(lm = lnr_lm, rf = lnr_rf, mean = lnr_mean),
  formula = mpg ~ .)

compare_learners(sl_model)

sl_model <- super_learner(
  data = mtcars,
  learners = list(lnr_logistic, lnr_rf_binary, mean = lnr_mean),
  formula = am ~ mpg,
  outcome_type = 'binary')
compare_learners(sl_model)
```

---

cv_character_and_factors_schema

Cross Validation Training/Validation Splits with Characters/Factor Columns

---

Description

Designed to handle cross-validation on models like randomForest, ranger, glmnet, etc., where the model matrix of newdata must match eactly the model matrix of the training dataset, this function intends to answer the need "The training datasets need to have every level of every discrete-type column that appears in the data."

Usage

```
cv_character_and_factors_schema(
  data,
  n_folds = 5,
  cv_sl_mode = TRUE,
  check_validation_datasets_too = TRUE
)
```

Arguments

`data`	Data to use in training a `super_learner`.
`n_folds`	The number of cross-validation folds to use in constructing the `super_learner`.
`cv_sl_mode`	A binary (default: TRUE) indicator for if the output training/validation data lists will be used inside another `super_learner` call. If so, then the training data needs to have every level appear at least twice so that the data can be put into further training/validation splits.
`check_validation_datasets_too`	Enforce that the validation datasets produced also have every level of every character / factor type column present. This is particularly useful for learners like `glmnet` which require that the `newx` have the exact same shape/structure as the training data, down to binary indicators for every level that appears.

Details

The fundamental idea is to check if the unique levels of character and/or factor columns are represented in every training dataset.

Above and beyond this, this function is designed to support cv_super_learner, which inherently involves two layers of cross-validation. As a result, more stringent conditions are specified when the `cv_sl_mode` is enabled. For convenience this mode is enabled by default

Value

A list of two lists (`$training_data` and `$validation_data`) which are each lists of length `n_folds`. In each of those entries is a data.frame that contains the nth training or validation fold of the data.

a named list of two lists, each being a list of `n_folds` data frames.

Examples

```
if (requireNamespace("palmerpenguins", quietly = TRUE)) {
training_validation_splits <- cv_character_and_factors_schema(
  palmerpenguins::penguins)

# we can see the population breakdown across all the training
# splits:
sapply(training_validation_splits$training_data, function(df) {
  table(df$species)
  })
# notably, none of them are empty! this is crucial for certain
# types of learning algorithms that must see all levels appear in the
# training data, like random forests.

# certain models like glmnet require that the prediction dataset
# newx have the _exact_ same shape as the training data, so it
# can be important that every level appears in the validation data
# as well.  check that by looking into these types of tables:
sapply(training_validation_splits$validation_data, function(df) {
  table(df$species)
  })

# if you don't need this level of stringency, but you just want
# to make cv_splits where every level appears in the training_data,
# you can do so using the check_validation_datasets_too = FALSE
# argument.
penguins_small <- palmerpenguins::penguins[c(1:3, 154:156, 277:279), ]
penguins_small <- penguins_small[complete.cases(penguins_small),]

training_validation_splits <- cv_character_and_factors_schema(
  penguins_small,
  cv_sl_mode = FALSE,
  n_folds = 5,
  check_validation_datasets_too = FALSE)

sapply(training_validation_splits$training_data, function(df) {
  table(df$species)
  })

# now you can see plenty of non-appearing levels in the validation data:
sapply(training_validation_splits$validation_data, function(df) {
  table(df$species)
  })
}
```

cv_origami_schema *Cross-Validation with Origami*

Description

Cross-Validation with Origami

Usage

```
cv_origami_schema(
  data = data,
  n_folds = 5,
  fold_fun = origami::folds_vfold,
  cluster_ids = NULL,
  strata_ids = NULL,
  ...
)
```

Arguments

`data`	a data.frame (or similar) to split into training and validation datasets.
`n_folds`	The number of `training_data` and `validation_data` data frames to make.
`fold_fun`	An `origami::folds_*` function
`cluster_ids`	A vector of cluster ids. Clusters are treated as a unit – that is, all observations within a cluster are placed in either the training or validation set. See `?origami::make_folds`.
`strata_ids`	A vector of strata ids. Strata are balanced: insofar as possible the distribution in the sample should be the same as the distribution in the training and validation sets. See `?origami::make_folds`.
`...`	Extra arguments to be passed to `origami::make_folds()`

Value

A list of `n_folds` `training_data` and `validation_data` data.frames

Examples

```
# to use origami::folds_vfold behind the scenes, just tell nadir::super_learner
# you want to use cv_origami_schema.

sl_model <- super_learner(
  data = mtcars,
  formula = mpg ~ cyl + hp,
  learners = list(rf = lnr_rf, lm = lnr_lm, mean = lnr_mean),
  cv_schema = cv_origami_schema
 )
```

```
# if you want to use a different origami::folds_* function, pass it into cv_origami_schema
sl_model <- super_learner(
  data = mtcars,
  formula = mpg ~ cyl + hp,
  learners = list(rf = lnr_rf, lm = lnr_lm, mean = lnr_mean),
  cv_schema = \(data, n_folds) {
    cv_origami_schema(data, n_folds, fold_fun = origami::folds_loo)
    }
 )
```

---

cv_random_schema — *Assign Data to One of n_folds Randomly and Produce Training/Validation Data Lists*

Description

Each row in the data are assigned to one of `1:n_folds` at random. Then for each of `i` in `1:n_folds`, the `training_data[[i]]` is comprised of the data with `sl_fold != i`, i.e., capturing roughly `(n-folds-1)/n_folds` proportion of the data. The validation data is a list of dataframes, each comprising of roughly `1/n_folds` proportion of the data.

Usage

```
cv_random_schema(data, n_folds = 5)
```

Arguments

`data`	a data.frame (or similar) to split into training and validation datasets.
`n_folds`	The number of `training_data` and `validation_data` data frames to make.

Details

Since the assignment to folds is random, the proportions are not exact or guaranteed and there is some variability in the size of each `training_data` data frame, and likewise for the `validation_data` data frames.

Value

A list of two lists (`$training_data` and `$validation_data`) which are each lists of length `n_folds`. In each of those entries is a data.frame that contains the nth training or validation fold of the data.

Examples

```
data(Boston, package = 'MASS')
training_validation_data <- cv_random_schema(Boston, n_folds = 3)
# take a look at what's in the output:
str(training_validation_data, max.level = 2)
```

cv_super_learner *Cross-Validating a* super_learner

Description

Produce cv-rmse for a super_learner specified by a closure that accepts data and returns a super_learner prediction function.

Usage

```
cv_super_learner(
  data,
  learners,
  formulas,
  y_variable = NULL,
  n_folds = 5,
  determine_super_learner_weights = determine_super_learner_weights_nnls,
  ensemble_or_discrete = "ensemble",
  cv_schema = cv_random_schema,
  outcome_type = "continuous",
  extra_learner_args = NULL,
  cluster_ids = NULL,
  strata_ids = NULL,
  weights = NULL,
  loss_metric,
  use_complete_cases = FALSE
)
```

Arguments

data	Data to use in training a super_learner.
learners	A list of predictor/closure-returning-functions. See Details.
formulas	Either a single regression formula or a vector of regression formulas.
y_variable	Typically y_variable can be inferred automatically from the formulas, but if needed, the y_variable can be specified explicitly.
n_folds	The number of cross-validation folds to use in constructing the super_learner.
determine_super_learner_weights	A function/method to determine the weights for each of the candidate learners. The default is to use determine_super_learner_weights_nnls.
ensemble_or_discrete	Defaults to 'ensemble', but can be set to 'discrete'. Discrete super_learner() chooses only one of the candidate learners to have weight 1 in the resulting prediction algorithm, while ensemble super_learner() combines predictions from 1 or more candidate learners, with respective weights adding up to 1.

`cv_schema`	A function that takes `data`, `n_folds` and returns a list containing `training_data` and `validation_data`, each of which are lists of `n_folds` data frames.
`outcome_type`	One of 'continuous', 'binary', 'multiclass', or 'density'. `outcome_type` is used to infer the correct `determine_super_learner_weights` function if it is not explicitly passed.
`extra_learner_args`	A list of equal length to the `learners` with additional arguments to pass to each of the specified learners.
`cluster_ids`	(default: null) If specified, clusters will either be entirely assigned to training or validation (not both) in each cross-validation split.
`strata_ids`	(default: null) If specified, strata are balanced across training and validation splits so that strata appear in both the training and validation splits.
`weights`	If specified, (per observation) weights are used to indicate that risk minimization across models (i.e., the meta-learning step) should be targeted to higher weight observations.
`loss_metric`	A loss metric function, like the mean-squared-error or negative-log-loss to be used in evaluating the learners on held-out data and minimized through convex optimization. A loss metric should take two (vector) arguments: predictions, and true outcomes, and produce a single statistic summarizing the performance of each learner. Defaults to the mean-squared-error `nadir:::mse()`.
`use_complete_cases`	(default: FALSE) If the `data` passed have any NA or NaN missing data, restrict the `data` to `data[complete.cases(data),]`.

Details

The idea is that `cv_super_learner` splits the data into training/validation splits, trains `super_learner` on each training split, and then evaluates their predictions on the held-out validation data, calculating a root-mean-squared-error on those held-out data.

This function does print a message if the `loss_function` argument is not set explicitly, letting the user know that the mean-squared-error will be used by default. Pass in `loss_function = nadir:::mse` to `super_learner()` if you'd like to suppress this message, or use a similar approach for the appropriate loss function depending on context.

Value

A list containing `$trained_learners` and `$cv_loss` which respectively include 1) the trained super learner models on each fold of the data, their holdout predictions and, 2) the cross-validated estimate of the risk (expected loss) on held-out data.

Examples

```
cv_super_learner(
  data = mtcars,
  formula = mpg ~ cyl + hp,
  learners = list(lnr_mean, lnr_lm))
```

`density_learners` *Conditional Density Estimation in the* `{nadir}` *Package*

Description

The following learners are available for conditional density estimation:

- `lnr_lm_density`
- `lnr_glm_density`
- `lnr_homoscedastic_density`

Details

There are a few important things to know about conditional density estimation in the `nadir` package.

Firstly, conditional density learners must produce prediction functions that predict *densities* at the new outcome values given the new covariates.

Secondly, the implemented density estimators come in two flavors: those with a strong assumption (that of conditional normality), and those with much weaker assumptions. The strong assumption is encoded into learners like `lnr_lm_density` and `lnr_glm_density` and says "after we model the predicted mean given covariates, we expect the remaining errors to be normally distributed." The more flexible learners produced by `lnr_homoskedastic_density` are similar in spirit, except they fit a `stats::density` kernel bandwidth smoother to the error distribution (after predicting the conditional expected mean).

A subpoint to the above point that's worth calling attention to is that `lnr_homoskedastic_density` is a learner factory. That is to say, given a `mean_lnr`, `lnr_homoskedastic_density` produces a conditional density learner that uses that `mean_lnr`.

Work is ongoing on implementing a `lnr_heteroskedastic_density` learner that allows for predicting higher or lower variance in the conditional density given covariates.

Conditional density learners should be combined with the negative log loss function when using `super_learner()` or using `compare_learners`. Refer to the 2003 Dudoit and van der Laan paper for a starting place on the appropriate loss functions to use for different types of outcomes. `https://biostats.bepress.com/ucbbiostat/paper130/`

See Also

learners binary_learners multiclass_learners

determine_super_learner_weights_nnls

Determine SuperLearner Weights with Nonnegative Least Squares

Description

This function accepts a dataframe that is structured to have one column `Y` and other columns with unique names corresponding to different model predictions for `Y`, and it will use nonnegative least squares to determine the weights to use for a SuperLearner.

Usage

```
determine_super_learner_weights_nnls(data, y_variable, obs_weights = NULL)
```

Arguments

`data`	A data frame consisting of an outcome (y_variable) and other columns corresponding to predictions from candidate learners.
`y_variable`	The string name of the outcome column in `data`.
`obs_weights`	A vector of weights for each observation that dictate how prediction should be more targeted to higher weighted observations.

Value

A vector of weights to be used for each of the learners.

Examples

```
# suppose that we have a data.frame of predictions from different candidate
# learners:
prediction_data <- data.frame(
  lm = lnr_lm(mtcars, mpg ~ hp)(mtcars),
  rf = lnr_rf(mtcars, mpg ~ hp)(mtcars),
  rf2 = lnr_rf(mtcars, mpg ~ hp, ntree = 20)(mtcars),
  earth = lnr_earth(mtcars, mpg ~ hp)(mtcars))
# make sure it includes the outcome y_variable
prediction_data$mpg <- mtcars$mpg

# we can use determine_super_learner_weights() fn to apply the non-negative least
# squares algorithm to produce weights for averaging the learners
determine_super_learner_weights_nnls(
  data = prediction_data,
  y_variable = 'mpg')
```

determine_weights_for_binary_outcomes

Determine Weights Appropriately for Super Learner given Binary Outcomes

Description

Determine Weights Appropriately for Super Learner given Binary Outcomes

Usage

```
determine_weights_for_binary_outcomes(data, y_variable, obs_weights = NULL)
```

Arguments

`data`	A data.frame with columns corresponding to predicted probabilities of 1 from each learner and the true y_variable from held-out data
`y_variable`	A character indicating the outcome variable in the data.frame.
`obs_weights`	A vector of weights for each observation that dictate how prediction should be more targeted to higher weighted observations.

Value

A vector of weights to be used for each of the learners.

Examples

```
predicted_probabilities <- data.frame(
  logistic = lnr_logistic(mtcars, am ~ hp)(mtcars),
  nnet = lnr_nnet(mtcars, am ~ hp)(mtcars),
  am = mtcars$am)
determine_weights_for_binary_outcomes(predicted_probabilities, y_variable = 'am')
```

determine_weights_using_neg_log_loss

Determine Weights for Density Estimators for SuperLearner

Description

Determine Weights for Density Estimators for SuperLearner

Usage

```
determine_weights_using_neg_log_loss(data, y_variable, obs_weights = NULL)
```

Arguments

`data`	A data.frame with columns corresponding to predicted densities from each learner and the true y_variable from held-out data
`y_variable`	A character indicating the outcome variable in the data.frame.
`obs_weights`	A vector of weights for each observation that dictate how prediction should be more targeted to higher weighted observations.

Value

A vector of weights to be used for each of the learners.

Examples

```
predicted_densities <- data.frame(
  lm = lnr_lm_density(mtcars, mpg ~ hp)(mtcars),
  earth = lnr_homoskedastic_density(mtcars, mpg ~ hp, mean_lnr = lnr_earth)(mtcars),
  rf = lnr_homoskedastic_density(mtcars, mpg ~ hp, mean_lnr = lnr_rf)(mtcars),
  rf2 = lnr_homoskedastic_density(mtcars, mpg ~ hp, mean_lnr = lnr_rf,
    mean_lnr_args = list(ntree = 20))(mtcars),
  mpg = mtcars$mpg)
determine_weights_using_neg_log_loss(predicted_densities, y_variable = 'mpg')
```

`df_to_survival_stacked` *Repeat Observations for Survival Stacking*

Description

Per the approach in *A review of survival stacking: a method to cast survival regression analysis as a classification problem* https://www.degruyterbrill.com/document/doi/10.1515/ijb-2022-0055/html https://arxiv.org/abs/2107.13480, we provide `df_to_survival_stacked` as a helper function for converting traditional survival data (one observation = one row) into the survival stacked data structure, a repeated observations data structure where multiple rows exist for each individual for each timepoint at which they were still in the risk set up to and including their event time.

Usage

```
df_to_survival_stacked(
  data,
  id_col = NULL,
  time_col,
  status_col,
  covariate_cols,
  period_duration = 1,
  custom_times = NULL
)
```

Arguments

`data`	A data frame with survival -type outcomes including an event indicator and a time-to-event-or-censoring column
`id_col`	(string) name of the id column that is unique to each observation in `data`. If one is not specified, one will be created (called `.id`) assuming that each row is a unique observation.
`time_col`	(string) name of the time-to-event column
`status_col`	(string) name of the 0/1 event indicator column
`covariate_cols`	(string vector) names of your predictors
`period_duration`	(numeric) length of each time-period (e.g. 1)
`custom_times`	(numeric vector) (optional) A vector of the time-period breakpoints. If events could have occurred at any time after zero, this should begin with 0.

Value

a data.frame of repeated observations, one row for each time-step, that indicates which observations remain in the risk set and whether or not an event occurs.

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  df_to_survival_stacked(
    data = survival::kidney,
    id_col = 'id',
    time_col = 'time',
    status_col = 'status',
    covariate_cols = c('age', 'sex', 'disease', 'frail'))
}
```

`learners` *Learners in the* `{nadir}` *Package*

Description

The following learners are available for continuous outcomes:

Details

- `lnr_mean`
- `lnr_earth`
- `lnr_gam`
- `lnr_glm`
- `lnr_glmer`

- `lnr_glmnet`
- `lnr_hal`
- `lnr_lm`
- `lnr_lmer`
- `lnr_ranger`
- `lnr_rf`
- `lnr_xgboost`

See `?density_learners` to learn more about using conditional density estimation in `nadir`.

`lnr_mean` is generally provided only for benchmarking purposes to compare other learners against to ensure correct specification of learners, since any prediction algorithm should (in theory) outperform just using the mean of the outcome for all predictions.

If you'd like to build a new learner, we recommend reading the source code of several of the learners provided with `{nadir}` to get a sense of how they should be specified.

A learner, as `{nadir}` understands them, is a function which takes in `data`, a `formula`, possibly `...`, and returns a function that predicts on its input `newdata`.

A simple example is reproduced here for ease of reference:

Examples

```
lnr_glm <- function(data, formula, weights = NULL, ...) {
 model <- stats::glm(formula = formula, data = data, weights = weights, ...)

 return(function(newdata) {
   predict(model, newdata = newdata, type = 'response')
 })
}
```

`list_known_learners` *List Known Learners*

Description

List Known Learners

Usage

```
list_known_learners(type = "any")
```

Arguments

`type`	One of 'any' or a supported outcome type in nadir including at least 'continuous', 'binary', 'multiclass', 'density'. See `?super_learner()`.

Value

A character vector of functions that were automatically recognized as nadir learners with the prediction/outcome type given.

Examples

```
list_known_learners()
list_known_learners('continuous')
list_known_learners('binary')
list_known_learners('density')
list_known_learners('multiclass')
```

`lnr_earth` *Earth Learner*

Description

A wrapper for `earth::earth()` for use in `nadir::super_learner()`.

Usage

```
lnr_earth(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_earth(mtcars, mpg ~ hp + disp + am + wt)(mtcars)
```

`lnr_gam` *Generalized Additive Model Learner*

Description

A wrapper for `mgcv::gam()` for use in `nadir::super_learner()`.

Usage

```
lnr_gam(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_gam(mtcars, mpg ~ s(hp) + disp + am + wt)(mtcars)
lnr_gam(mtcars, mpg ~ s(hp) + disp + am + wt, family = Gamma)(mtcars)
```

`lnr_gbm` *Gradient Boosting Machines Learner*

Description

A wrapper for `gbm::gbm()` for use in `nadir::super_learner()`.

Usage

```
lnr_gbm(data, formula, verbose = FALSE, n.minobsinnode = 0, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`verbose`	(default: FALSE) if set to TRUE, information about the automatic outcome type inferred by `gbm` will be messaged to the console, as well as the number of trees used.
`n.minobsinnode`	(default: 0) An integer specifying the minimum number of observations in the terminal nodes of the trees. See the gbm documentation for more. Set here to 0 to account for the potential of very small splits in cross-fitting.
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_gbm(mtcars, mpg ~ hp)(mtcars)
```

`lnr_glm` *GLM Learner*

Description

A wrapper for `stats::glm()` for use in `nadir::super_learner()`.

Usage

```
lnr_glm(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_glm(mtcars, mpg ~ hp + disp + am + wt)(mtcars)
lnr_glm(mtcars, mpg ~ hp + disp + am + wt, family = Gamma)(mtcars)
```

`lnr_glmer` *Generalized Linear Mixed-Effects (*`lme4::glmer`*) Learner*

Description

A wrapper for `lme4::glmer()` for use in `nadir::super_learner()`.

Usage

```
lnr_glmer(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
# random intercepts for each level of cyl column:
suppressMessages({
# singular fit, but that's ok if all you need is prediction:
lnr_glmer(mtcars, mpg ~ (1|cyl) + disp + wt, family = Gamma)(mtcars)
})
```

`lnr_glmnet` *glmnet Learner*

Description

A wrapper for `glmnet::glmnet()` for use in `nadir::super_learner()`.

Usage

```
lnr_glmnet(data, formula, weights = NULL, lambda = 0.2, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`lambda`	The multiplier parameter for the penalty; see `?glmnet::glmnet`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Details

glmnet predictions will by default, if lambda is unspecified, return a matrix of predictions for varied lambda values, hence the need to explicitly handle the lambda argument in building glmnet learners.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_glmnet(mtcars, mpg ~ hp + disp + am + wt, lambda = .5)(mtcars)
```

`lnr_glm_density` *Conditional Normal Density Estimation Given Mean Predictors — with GLMs*

Description

This is a step up from the `lnr_lm_density` in that it uses a `glm` for the conditional mean model. Note that this allows for specification of `glm` features like `family = ...` in the `,..` arguments, and that's the main advantage over the `lnr_lm_density`. Also note that this still differs from using `lnr_homoskedastic_density` with `mean_lnr = lnr_glm` because `lnr_homoscedastic_density` uses `stats::density` to do kernel bandwidth smoothing on the error distribution of the mean predictions..

Usage

```
lnr_glm_density(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

a closure (function) that produces density estimates at the `newdata` given according to the fit model.

Examples

```
# for example, we could use a Poisson assumption with identity link:

lnr_glm_density(mtcars, hp ~ mpg, family = poisson(link = 'identity'))(mtcars)
hp_seq <- seq(min(mtcars$hp), max(mtcars$hp), length.out = 1000)
plot(
  x = hp_seq,
  y = lnr_glm_density(mtcars, hp ~ mpg, family = poisson(link = 'identity'))(
    data.frame(hp = hp_seq, mpg = rep(mean(mtcars$mpg), 1000))),
  xlab = 'hp',
  ylab = 'density',
  main = 'normal density model of horsepower given mean(mpg)')
```

`lnr_hal` *Highly Adaptive Lasso*

Description

Highly Adaptive Lasso

Usage

```
lnr_hal(data, formula, weights = NULL, lambda = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`lambda`	The multiplier parameter for the penalty; see `?glmnet::glmnet`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
suppressWarnings({
# hal prints a lot of messages about some threads not reaching convergence
lnr_hal(mtcars, mpg ~ hp)(mtcars)
})
```

lnr_heteroskedastic_density *Conditional Density Estimation with Heteroskedasticity*

Description

Conditional Density Estimation with Heteroskedasticity

Usage

```
lnr_heteroskedastic_density(
  data,
  formula,
  mean_lnr,
  var_lnr,
  mean_lnr_args = NULL,
  var_lnr_args = NULL,
  density_args = NULL
)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`mean_lnr`	A learner (function) passed in to be trained on the data with the given formula and then used to predict conditional means for provided `newdata`.
`var_lnr`	A learner (function) passed in to be trained on the squared error from the `mean_lnr` on the given data and then used to predict the expected variance for the density distribution of the outcome centered around the predicted conditional mean in the output.
`mean_lnr_args`	Extra arguments to be passed to the `mean_lnr`
`var_lnr_args`	Extra arguments to be passed to the `var_lnr`
`density_args`	Extra arguments to be passed to the kernel density smoother `stats::density`, especially things like bw for specifying the smoothing bandwidth. See `?stats::density`.

Value

a closure (function) that produces density estimates at the `newdata` given according to the fit model.

Examples

```
# fit a conditional density model with mean model as a randomForest
fit_density_hetero <- lnr_heteroskedastic_density(
  data = mtcars,
  formula = mpg ~ hp,
  mean_lnr = lnr_rf,
  var_lnr = lnr_lm)

# and what we should get back should be predicted densities at the
# observed mpg given the covariates hp
fit_density_hetero(mtcars)

if (requireNamespace("ggplot2", quietly = TRUE)) {
hp_grid <- with(mtcars, seq(min(hp), max(hp), length.out=100))
mpg_grid <- with(mtcars, seq(min(mpg), max(mpg), length.out=100))
mt_grid <- expand.grid(mpg = mpg_grid, hp = hp_grid)
plt_df <- cbind(mt_grid, pred_dens = fit_density_hetero(mt_grid))
require(ggplot2)
ggplot(plt_df, aes(x = hp, y = mpg, fill = pred_dens)) +
geom_tile() +
scale_fill_viridis_c() +
ggtitle("Density Model of MPG given HP")
}
```

lnr_homoskedastic_density *Conditional Density Estimation with Homoskedasticity Assumption*

Description

This function accepting an `mean_lnr`, which it then trains on the data and formula given. Then `stats::density` is fit to the error (difference between observed outcome and the `mean_lnr` predictions).

Usage

```
lnr_homoskedastic_density(
  data,
  formula,
  mean_lnr,
  mean_lnr_args = NULL,
  density_args = NULL,
  weights = NULL
)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`mean_lnr`	A learner (function) passed in to be trained on the data with the given formula and then used to predict conditional means for provided `newdata`.
`mean_lnr_args`	Extra arguments to be passed to the `mean_lnr`
`density_args`	Extra arguments to be passed to the kernel density smoother `stats::density`, especially things like bw for specifying the smoothing bandwidth. See `?stats::density`.
`weights`	Observation weights; see `?lm`

Details

This returns a function that takes in `newdata` and produces density estimates according to the estimated `stats::density` fit the error from the `newdata` observed outcome and the prediction from the `mean_lnr`.

That is to say, this follows the following procedure (assuming Y as the outcome and X as a matrix of predictors):

$$\texttt{obtain } \hat{\mathbb{E}}(Y|X) \quad \texttt{using} \quad \texttt{mean\_learner}$$
$$\texttt{fit } \hat{f} \leftarrow \texttt{density}(Y - \hat{\mathbb{E}}(Y|X))$$
$$\texttt{return} \quad \texttt{function}(\texttt{newdata})\{\hat{f}(\texttt{newdata\$Y} - \hat{\mathbb{E}}[Y|\texttt{newdata\$X}])\}$$

Value

A predictor function that takes in `newdata` and produces density estimates

Examples

```
# fit a conditional density model with mean model as a randomForest
fit_density_lnr <- lnr_homoskedastic_density(
  data = mtcars,
  formula = mpg ~ hp,
  mean_lnr = lnr_rf)

# and what we should get back should be predicted densities at the
# observed mpg given the covariates hp
fit_density_lnr(mtcars)
```

`lnr_lm` *Linear Model Learner*

Description

A wrapper for `lm()` for use in `nadir::super_learner()`.

Usage

```
lnr_lm(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_lm(mtcars, mpg ~ hp + disp + am + wt)(mtcars)
```

`lnr_lmer` *Random/Mixed-Effects (`lme4::lmer`) Learner*

Description

A wrapper for `lme4::lmer` for use in `nadir::super_learner()`.

Usage

```
lnr_lmer(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
# random intercepts for each level of cyl column:
lnr_lmer(mtcars, mpg ~ (1|cyl) + disp + am + wt)(mtcars)
```

lnr_lm_density *Conditional Normal Density Estimation Given Mean Predictors*

Description

This is the simplest possible density estimator that is entertainable. It fits a `lm` model to the data, and uses the variance of the residuals to parameterize a model of the data as $\mathcal{N}(y|\beta x, \sigma^2)$.

Usage

```
lnr_lm_density(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

a closure (function) that produces density estimates at the `newdata` given according to the fit model.

Examples

```
lnr_lm_density(mtcars, hp ~ mpg)(mtcars)
hp_seq <- seq(min(mtcars$hp), max(mtcars$hp), length.out = 1000)
plot(
  x = hp_seq,
  y = lnr_lm_density(mtcars, hp ~ mpg)(
    data.frame(hp = hp_seq, mpg = rep(mean(mtcars$mpg), 1000))),
  xlab = 'hp',
  ylab = 'density',
  main = 'normal density model of horsepower given mean(mpg)')
```

lnr_logistic *Standard Logistic Regression for Binary Classification*

Description

A wrapper provided for convenience around `lnr_glm` that sets `family = binomial(link = 'logit')`.

Usage

```
lnr_logistic(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions for the probability of the outcome being 1/TRUE (a numeric vector of values, one for each row of `newdata`).

Examples

```
lnr_logistic(mtcars, am ~ hp)(mtcars)
```

`lnr_mean` *Mean Learner*

Description

This is a very naive/simple learner that simply predicts the mean of the outcome for every row of input `newdata`. This is primarily useful for benchmarking and confirming that other learners are performing better than `lnr_mean`. Additionally, it may be the case that some learners are over-fitting the data, and giving some weight to `lnr_mean` helps to reduce over-fitting in `super_learner()`.

Usage

```
lnr_mean(data, formula, weights = NULL)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_mean(mtcars, mpg ~ hp)(mtcars)
```

`lnr_multinomial_nnet` `nnet::multinom` *Multinomial Learner*

Description

`nnet::multinom` Multinomial Learner

Usage

```
lnr_multinomial_nnet(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of density prediction values at the outcome value observed in the `newdata` conditioning on the predictor variables in `newdata`).

Examples

```
df <- mtcars
df$cyl <- as.factor(df$cyl)
lnr_multinomial_nnet(df, cyl ~ hp + mpg)(df)
lnr_multinomial_nnet(iris, Species ~ .)(iris)
```

`lnr_multinomial_vglm` `VGAM::vglm` *Multinomial Learner*

Description

`VGAM::vglm` Multinomial Learner

Usage

```
lnr_multinomial_vglm(data, formula, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of density prediction values at the outcome value observed in the `newdata` conditioning on the predictor variables in `newdata`).

Examples

```
df <- mtcars
df$cyl <- as.factor(df$cyl)
lnr_multinomial_vglm(df, cyl ~ hp + mpg)(df)
lnr_multinomial_vglm(iris, Species ~ .)(iris)
```

`lnr_nnet` *Use nnet for Binary Classification*

Description

Use nnet for Binary Classification

Usage

```
lnr_nnet(data, formula, trace = FALSE, size, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`trace`	Whether nnet should print out its optimization success
`size`	Size for neural network hidden layer
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

Examples

```
lnr_nnet(mtcars, am ~ ., size = 50)(mtcars)
lnr_nnet(iris, I(Species=='setosa') ~ ., size = 50)(iris)
```

`lnr_ranger` *ranger Learner*

Description

A wrapper for `ranger::ranger()` for use in `nadir::super_learner()`.

Usage

```
lnr_ranger(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_ranger(mtcars, mpg ~ hp)(mtcars)
```

---

lnr_ranger_binary *ranger Learner for Binary Outcomes*

Description

A wrapper for `ranger::ranger()` for use in `nadir::super_learner()`.

Usage

```
lnr_ranger_binary(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_ranger_binary(mtcars, am ~ hp)(mtcars)
```

`lnr_rf` *randomForest Learner*

Description

A wrapper for `randomForest::randomForest()` for use in `nadir::super_learner()`.

Usage

```
lnr_rf(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_rf(mtcars, mpg ~ hp + disp + am + wt, ntree = 20)(mtcars)
```

`lnr_rf_binary` *Use Random Forest for Binary Classification*

Description

Use Random Forest for Binary Classification

Usage

```
lnr_rf_binary(data, formula, weights = NULL, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions for the probability of the outcome being 1/TRUE (a numeric vector of values, one for each row of `newdata`).

Examples

```
lnr_rf_binary(data = mtcars, am ~ mpg)(mtcars)
lnr_rf_binary(mtcars, am ~ hp)(mtcars)
```

`lnr_xgboost` *XGBoost Learner*

Description

A wrapper for `xgboost::xgboost()` for use in `nadir::super_learner()`.

Usage

```
lnr_xgboost(data, formula, weights = NULL, nrounds = 1000, ...)
```

Arguments

`data`	A dataframe to train a learner / learners on.
`formula`	A regression formula to use inside this learner.
`weights`	Observation weights; see `?lm`
`nrounds`	The max number of boosting iterations
`...`	Any extra arguments that should be passed to the internal model for model fitting purposes.

Value

A prediction function that accepts `newdata`, which returns predictions (a numeric vector of values, one for each row of `newdata`).

See Also

learners

Examples

```
lnr_xgboost(mtcars, mpg ~ hp)(mtcars)
```

`make_learner_names_unique`
Make Unique Learner Names

Description

Make Unique Learner Names

Usage

```
make_learner_names_unique(learners)
```

Arguments

`learners`	A list of learners. See `?learners`

Value

A list of learners with (possibly) improved names.

Examples

```
learners <-
  list(
    mean = lnr_mean,
    rf = lnr_rf,
    rf = lnr_rf,
    lnr_glm,
    lnr_xgboost,
    function(data, formula) {},
    function(data, formula) {})
learners <- nadir::make_learner_names_unique(learners)
names(learners)

learners <-
  list(
    lnr_mean,
    lnr_rf,
    lnr_rf,
    lnr_glm,
    lnr_xgboost,
    function(data, formula) {},
    function(data, formula) {})
learners <- nadir::make_learner_names_unique(learners)
names(learners)
```

`multiclass_learners` *Multiclass Learners in* `{nadir}`

Description

- `lnr_multinomial_nnet`
- `lnr_multinomial_vglm`

Details

Suppose one of these is trained on some data and the fit learner is stored. Suppose we are going to call it on `newdata` and `newdata$class` is the outcome variable being predicting.

The important thing to know about multiclass learners is that they produce predictions that the outcome class is equal to `newdata$class` given the covariates specified in `newdata`.

Similar to density estimation, we want to use `determine_weights_using_neg_log_loss` in our calls to `super_learner()`. This can be done automatically by declaring `outcome_type = 'multiclass'` in calling `super_learner()`

See Also

density_learners binary_learners learners

Examples

```
super_learner(
  data = iris,
  learners = list(lnr_multinomial_vglm, lnr_multinomial_vglm, lnr_multinomial_nnet),
  formulas = list(
  .default = Species ~ .,
  multinomial_vglm2 = Species ~ Petal.Length*Petal.Width + .),
  outcome_type = 'multiclass'
  )
```

`nadir_supported_types` *Outcome types supported by* `{nadir}`

Description

The following outcome types are supported in the `{nadir}` package:

Usage

```
nadir_supported_types
```

Format

An object of class `character` of length 4.

Details

- continuous
- binary
- multiclass
- density

See Also

super_learner

`negative_log_loss` *Negative Log Loss*

Description

Negative Log Loss

Usage

```
negative_log_loss(predicted_densities, ...)
```

Arguments

`predicted_densities`	The predicted densities from a learner predicted at `newdata`.
`...`	Because `nadir::compare_learners()` passes `estimates, truth` to the `loss_metric` passed to it, `negative_log_loss` accepts ... but doesn't do anything with it.

Details

`negative_log_loss` encodes the logic: if $\hat{p}_n$ is a good model of the conditional densities, then it should minimize:

$$-\sum(\log(\hat{p}_n(X_i))$$

Value

A sum of the negative log loss given a vector of predicted probabilities/densities for some observed outcome.

Examples

```
# suppose we have some prediction probabilities _at the true values_:
predicted_probabilities <- lnr_logistic(mtcars, am ~ hp)(mtcars)
# we can calculate the -log(loss) for binary predicted probabilities like so:
negative_log_loss(predicted_probabilities)
```

negative_log_loss_for_binary

Negative Log Loss for Binary

Description

Negative Log Loss for Binary

Usage

```
negative_log_loss_for_binary(predicted_probabilities, true_outcomes)
```

Arguments

`predicted_probabilities`	The predicted probabilities from a learner predicted at `newdata`.
`true_outcomes`	A vector of true outcomes to use in calculating the negative log loss of the relevant predicted probabilities.

Value

A sum of the negative log loss given a vector of predicted probabilities for `outcome == 1` or equivalently a ’success’.

predict.nadir_sl_model

Predict from a `nadir::super_learner()` *model*

Description

Predict from a `nadir::super_learner()` model

Usage

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

Arguments

`object`	An object of class inheriting from `nadir_sl_model`.
`newdata`	A tabular data structure (data.frame or matrix) of predictor variables.
`...`	Ellipses, solely provided so that the `predict.nadir_sl_model` method is compatible with the generic `predict`, which takes ellipses as an argument.

Value

a numeric vector of predicted values

Examples

```
sl_fit <- super_learner(mtcars, mpg ~ hp,
  learners = list(lnr_lm, lnr_rf, lnr_earth))
predict(sl_fit, newdata = mtcars)
```

---

screeners *Wrapping Learners with a Screener*

Description

Screeners work off of the principle that they should take the same arguments that a learner does and return a modified dataset and formula in which variables that have failed to meet some threshold have been screened out.

Details

A screener can be added to a learner by using the `add_screener(learner, screener)` function provided. This returns a modified learner that implements screening based on the data and formula passed.

So far, the screeners implemented rely on being able to call `model.matrix` and therefore only support standard (generalized) linear model syntax like those mentioned in `?formula`.

See Also

screener_cor, screener_cor_top_n, screener_t_test, add_screener

Examples

```
# examples for setting up a screened regression problem:
#
# users can just run a screener to see what data and formula terms pass the
# given screener conditions:

screened_regression_problem <- screener_cor(data = mtcars,
  formula = mpg ~ ., threshold = 0.5)
screened_regression_problem

screened_regression_problem2 <- screener_cor(data = mtcars,
  formula = mpg ~ ., threshold = 0.5, cor... = list(method = 'spearman'))
screened_regression_problem2

screened_regression_problem3 <- screener_t_test(data = mtcars,
  formula = mpg ~ ., t_statistic_threshold = 10)
screened_regression_problem3
```

```
# build a new learner with screening builtin:
 lnr_rf_screener_top_5_cor_terms <- add_screener(
   learner = lnr_rf,
   screener = screener_cor_top_n,
   screener_extra_args = list(cor... = list(method = 'spearman'),
                              keep_n_terms = 5)
 )

# train learner
trained_learner <- lnr_rf_screener_top_5_cor_terms(data = mtcars, formula = mpg ~ .)
mtcars_modified <- mtcars
mtcars_modified['gear'] <- 1 # gear is one of the least correlated variables with mpg
identical(trained_learner(mtcars), trained_learner(mtcars_modified))
```

screener_cor *Correlation Threshold Based Screening*

Description

Correlation Threshold Based Screening

Usage

```
screener_cor(data, formula, threshold = 0.2, cor... = NULL)
```

Arguments

`data`	A dataframe intended to be used with `super_learner()`
`formula`	The formula specifying the regression to be done
`threshold`	The correlation coefficient cutoff, below which variables are screened out from the dataset and regression formula.
`cor...`	An optional list of extra arguments to pass to `cor`. Use `method = 'spearman'` for the Spearman rank based correlation coefficient.

Details

If a variable used has little correlation with the outcome being predicted, we might want to screen that variable out from the predictors.

In large datasets, this is quite important, as having a huge number of columns could be computationally intractable or frustratingly time-consuming to run `super_learner()` with.

Value

A list of `$data` with columns screened out, `$formula` with variables screened out, and `$failed_to_correlate_names` the names of variables that failed to correlate with the outcome at least at the threshold level.

Examples

```
screener_cor(
  data = mtcars,
  formula = mpg ~ .,
  threshold = .5)

# We're also showing how to specify that you want the Spearman rank-based
# correlation coefficient, to get away from the assumption of linearity.

screener_cor(
  data = mtcars,
  formula = mpg ~ .,
  threshold = .5,
  cor... = list(method = 'spearman')
  )
```

`screener_cor_top_n` *Correlation Threshold Based Screening*

Description

Correlation Threshold Based Screening

Usage

```
screener_cor_top_n(data, formula, keep_n_terms, cor... = NULL)
```

Arguments

`data`	A dataframe intended to be used with `super_learner()`
`formula`	The formula specifying the regression to be done
`keep_n_terms`	Set to an integer value >=1, this indicates that the top n terms in the model frame with greatest absolute correlation with the outcome will be kept.
`cor...`	An optional list of extra arguments to pass to `cor`. Use `method = 'spearman'` for the Spearman rank based correlation coefficient.

Details

If a variable used has little correlation with the outcome being predicted, we might want to screen that variable out from the predictors.

In large datasets, this is quite important, as having a huge number of columns could be computationally intractable or frustratingly time-consuming to run `super_learner()` with.

Value

A list of `$data` with columns screened out, `$formula` with variables screened out, and `$failed_to_correlate_names` the names of variables that failed to correlate with the outcome at least at the threshold level.

Examples

```
screener_cor_top_n(
  data = mtcars,
  formula = mpg ~ .,
  keep_n_terms = 5)

# We're also showing how to specify that you want the Spearman rank-based
# correlation coefficient, to get away from the assumption of linearity.

screener_cor_top_n(
  data = mtcars,
  formula = mpg ~ .,
  keep_n_terms = 5,
  cor... = list(method = 'spearman')
  )
```

`screener_t_test` *t-test Based Screening: Thresholds on p.values and/or t statistics*

Description

Screens out variables from the formula and dataset based on a p.value and/or the absolute value of the t statistic from a univariate linear regression (with intercept and one term) comparing each predictor to the outcome (dependent) variable.

Usage

```
screener_t_test(
  data,
  formula,
  p_value_threshold = NULL,
  t_statistic_threshold = NULL
)
```

Arguments

`data`	a dataset with variables mentioned in the `formula`
`formula`	a `formula` with terms from `data`, intended to be used with a learner from `nadir`.
`p_value_threshold`	A numeric scalar where terms pass if the t test for the linear model coefficient has p value lower than or equal to the `p_value_threshold` given.
`t_statistic_threshold`	A numeric scalar where terms pass if they have a t test statistic greater than or equal to the `t_statistic_threshold` given.

Details

The intended use of `screener_t_test` and other screeners is for pragmatic purposes: when there are a very large number of candidate predictors, such that `super_learner` is very slow to run, predictor variables that fail to have a detectable association with the dependent variable of a formula should be dropped from the learner.

Value

A list of `$data` with columns screened out, `$formula` with variables screened out, and `$failed_to_pass_threshold` the names of variables that failed to associate with the outcome at least at the threshold level.

See Also

screeners, add_screener, screener_cor_top_n

`super_learner` *Super Learner: Cross-Validation Based Ensemble Learning*

Description

Super learning with functional programming!

Usage

```
super_learner(
  data,
  learners,
  formulas,
  y_variable = NULL,
  n_folds = 5,
  determine_super_learner_weights,
  ensemble_or_discrete = "ensemble",
  cv_schema,
  outcome_type = "continuous",
  extra_learner_args = NULL,
  cluster_ids = NULL,
  strata_ids = NULL,
  weights = NULL,
  use_complete_cases = FALSE
)
```

Arguments

`data`	Data to use in training a `super_learner`.
`learners`	A list of predictor/closure-returning-functions. See Details.
`formulas`	Either a single regression formula or a vector of regression formulas.

`y_variable`	Typically `y_variable` can be inferred automatically from the `formulas`, but if needed, the y_variable can be specified explicitly.
`n_folds`	The number of cross-validation folds to use in constructing the `super_learner`.
`determine_super_learner_weights`	A function/method to determine the weights for each of the candidate `learners`. The default is to use `determine_super_learner_weights_nnls`.
`ensemble_or_discrete`	Defaults to `'ensemble'`, but can be set to `'discrete'`. Discrete `super_learner()` chooses only one of the candidate learners to have weight 1 in the resulting prediction algorithm, while `ensemble` `super_learner()` combines predictions from 1 or more candidate learners, with respective weights adding up to 1.
`cv_schema`	A function that takes `data`, `n_folds` and returns a list containing `training_data` and `validation_data`, each of which are lists of `n_folds` data frames.
`outcome_type`	One of 'continuous', 'binary', 'multiclass', or 'density'. `outcome_type` is used to infer the correct `determine_super_learner_weights` function if it is not explicitly passed.
`extra_learner_args`	A list of equal length to the `learners` with additional arguments to pass to each of the specified learners.
`cluster_ids`	(default: null) If specified, clusters will either be entirely assigned to training or validation (not both) in each cross-validation split.
`strata_ids`	(default: null) If specified, strata are balanced across training and validation splits so that strata appear in both the training and validation splits.
`weights`	If specified, (per observation) weights are used to indicate that risk minimization across models (i.e., the meta-learning step) should be targeted to higher weight observations.
`use_complete_cases`	(default: FALSE) If the `data` passed have any NA or NaN missing data, restrict the `data` to `data[complete.cases(data),]`.

Details

The goal of any super learner is to use cross-validation and a set of candidate learners to 1) evaluate how the learners perform on held out data and 2) to use that evaluation to produce a weighted average (for continuous super learner) or to pick a best learner (for discrete super learner) of the specified candidate learners.

Super learner and its statistically desirable properties have been written about at length, including at least the following references:

- https://biostats.bepress.com/ucbbiostat/paper222/
- https://www.stat.berkeley.edu/users/laan/Class/Class_subpages/BASS_sec1_3.1.pdf

`nadir::super_learner` adopts several user-interface design-perspectives that will be useful to know in understanding what it does and how it works:

- The specification of learners should be *very flexible*, really only constrained by the fact that candidate learners should be designed for the same prediction problem but their details can wildly vary from learner to learner.
- It should be easy to specify a customized or new learner.

`nadir::super_learner` at its core accepts `data`, a `formula` (a single one passed to `formulas` is fine), and a list of `learners`.

`learners` are taken to be lists of functions of the following specification:

- a learner must accept a `data` and `formula` argument,
- a learner may accept more arguments, and
- a learner must return a prediction function that accepts `newdata` and produces a vector of prediction values given `newdata`.

In essence, a learner is specified to be a function taking (`data`, `formula`, ...) and returning a *closure* (see `http://adv-r.had.co.nz/Functional-programming.html#closures` for an introduction to closures) which is a function accepting `newdata` returning predictions.

Since many candidate learners will have hyperparameters that should be tuned, like depth of trees in random forests, or the `lambda` parameter for `glmnet`, extra arguments can be passed to each learner via the `extra_learner_args` argument. `extra_learner_args` should be a list of lists, one list of extra arguments for each learner. If no additional arguments are needed for some learners, but some learners you're using do require additional arguments, you can just put a `NULL` value into the `extra_learner_args`. See the examples.

In order to seamlessly support using features implemented by extensions to the formula syntax (like random effects formatted like random intercepts or slopes that use the `(age | strata)` syntax in `lme4` or splines like `s(age | strata)` in `mgcv`), we allow for the `formulas` argument to either be one fixed formula that `super_learner` will use for all the models, or a vector of formulas, one for each learner specified.

Note that in the examples a mean-squared-error (mse) is calculated on the same training/test set, and this is only useful as a crude diagnostic to see that super_learner is working. A more rigorous performance metric to evaluate `super_learner` on is the cv-rmse produced by cv_super_learner.

Value

An object of class inheriting from `nadir_sl_model`. This is an S3 object, with elements including a `$predict(newdata)` method, and some information about the fit model including `y_variable`, `outcome_type`, `learner_weights`, `holdout_predictions` and optionally information about any errors thrown by the learner fitting process.

See Also

predict.nadir_sl_model compare_learners

cv_super_learner

Examples

```
learners <- list(
     glm = lnr_glm,
     rf = lnr_rf,
     glmnet = lnr_glmnet,
     lmer = lnr_lmer
  )

# mtcars example ---
formulas <- c(
  .default = mpg ~ cyl + hp, # first three models use same formula
  lmer = mpg ~ (1 | cyl) + hp # lme4 uses different language features
  )

# fit a super_learner
sl_model <- super_learner(
  data = mtcars,
  formula = formulas,
  learners = learners)

# We recommend taking a look at this object to see what's contained inside it:
sl_model

compare_learners(sl_model)

# iris example ---
sl_model <- super_learner(
  data = iris,
  formula = list(
    .default = Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width,
    lmer = Sepal.Length ~ (Sepal.Width | Species) + Petal.Length),
  learners = learners)

# produce super_learner predictions and compare against the individual learners
compare_learners(sl_model)
```

Index