Package ‘stacks’

May 13, 2022

Title Tidy Model Stacking

Version 0.2.3

Description Model stacking is an ensemble technique that involves training a model to combine the outputs of many diverse statistical models, and has been shown to improve predictive performance in a variety of settings. 'stacks' implements a grammar for 'tidymodels'-aligned model stacking.

License MIT + file LICENSE

URL https://stacks.tidymodels.org/, https://github.com/tidymodels/stacks

BugReports https://github.com/tidymodels/stacks/issues

Depends R (>= 2.10)

Imports tune (>= 0.1.3), dplyr (>= 1.0.0), rlang (>= 0.4.0), tibble (>= 2.1.3), purrr (>= 0.3.2), parsnip (>= 0.0.4), workflows (>= 0.2.3), recipes (>= 0.1.15), rsample (>= 0.1.1), workflowsets (>= 0.1.0), butcher (>= 0.1.3), yardstick, tidyr, glue, ggplot2, glmnet, cli, stats, foreach

Suggests testthat (>= 3.0.0), covr, kknn, ranger, knitr, modeldata, rmarkdown, nnet, kernlab, mockr, h2o, SuperLearner

Config/Needs/website tidyverse/tidytemplate

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Simon Couch [aut, cre], Max Kuhn [aut], RStudio [cph, fnd]

Maintainer Simon Couch <simonpatrickcouch@gmail.com>

Repository CRAN

Date/Publication 2022-05-12 23:10:02 UTC

R topics documented:

`add_candidates` *Add model definitions to a data stack*

Description

`add_candidates()` collates the assessment set predictions and additional attributes from the supplied model definition (i.e. set of "candidates") to a data stack.

Behind the scenes, data stack objects are just tibble::tbl_dfs, where the first column gives the true response values, and the remaining columns give the assessment set predictions for each candidate. In the regression setting, there's only one column per ensemble member. In classification settings, there are as many columns per candidate ensemble member as there are levels of the outcome variable.

To initialize a data stack, use the `stacks()` function. Model definitions are appended to a data stack iteratively using several calls to `add_candidates()`. Data stacks are evaluated using the `blend_predictions()` function.

Usage

```
add_candidates(
  data_stack,
  candidates,
  name = deparse(substitute(candidates)),
  ...
)
```

Arguments

data_stack	A data_stack object.
candidates	A model definition: either a tune_results or resample_results object outputted from tune::tune_grid(), tune::tune_bayes(), or tune::fit_resamples(). These results must have been fitted with the control settings save_pred = TRUE, save_workflow = TRU the control_stack_grid(), control_stack_bayes(), and control_stack_resamples() documentation for helper functions.
name	The label for the model definition—defaults to the name of the candidates object.
...	Additional arguments. Currently ignored.

Value

A data_stack object–see stacks() for more details!

Example Data

This package provides some resampling objects and datasets for use in examples and vignettes derived from a study on 1212 red-eyed tree frog embryos!

Red-eyed tree frog (RETF) embryos can hatch earlier than their normal 7ish days if they detect potential predator threat. Researchers wanted to determine how, and when, these tree frog embryos were able to detect stimulus from their environment. To do so, they subjected the embryos at varying developmental stages to "predator stimulus" by jiggling the embryos with a blunt probe. Beforehand, though some of the embryos were treated with gentamicin, a compound that knocks out their lateral line (a sensory organ.) Researcher Julie Jung and her crew found that these factors inform whether an embryo hatches prematurely or not!

Note that the data included with the stacks package is not necessarily a representative or unbiased subset of the complete dataset, and is only for demonstrative purposes.

reg_folds and class_folds are rset cross-fold validation objects from rsample, splitting the training data into for the regression and classification model objects, respectively. tree_frogs_reg_test and tree_frogs_class_test are the analogous testing sets.

reg_res_lr, reg_res_svm, and reg_res_sp contain regression tuning results for a linear regression, support vector machine, and spline model, respectively, fitting latency (i.e. how long the embryos took to hatch in response to the jiggle) in the tree_frogs data, using most all of the other variables as predictors. Note that the data underlying these models is filtered to include data only from embryos that hatched in response to the stimulus.

class_res_rf and class_res_nn contain multiclass classification tuning results for a random forest and neural network classification model, respectively, fitting reflex (a measure of ear function) in the data using most all of the other variables as predictors.

log_res_rf and log_res_nn, contain binary classification tuning results for a random forest and neural network classification model, respectively, fitting hatched (whether or not the embryos hatched in response to the stimulus) using most all of the other variables as predictors.

See ?example_data to learn more about these objects, as well as browse the source code that generated them.

See Also

Other core verbs: blend_predictions(), fit_members(), stacks()

Examples

```
# see the "Example Data" section above for
# clarification on the objects used in these examples!

# put together a data stack using
# tuning results for regression models
reg_st <-
  stacks() %>%
  add_candidates(reg_res_lr) %>%
  add_candidates(reg_res_svm) %>%
  add_candidates(reg_res_sp)

reg_st

# do the same with multinomial classification models
class_st <-
  stacks() %>%
  add_candidates(class_res_nn) %>%
  add_candidates(class_res_rf)

class_st

# ...or binomial classification models
log_st <-
  stacks() %>%
  add_candidates(log_res_nn) %>%
  add_candidates(log_res_rf)

log_st

# use custom names for each model:
log_st2 <-
  stacks() %>%
  add_candidates(log_res_nn, name = "neural_network") %>%
  add_candidates(log_res_rf, name = "random_forest")

log_st2

# these objects would likely then be
# passed to blend_predictions():
log_st2 %>% blend_predictions()
```

autoplot.linear_stack *Plot results of a stacked ensemble model.*

Description

Plot results of a stacked ensemble model.

Usage

```
## S3 method for class 'linear_stack'
autoplot(object, type = "performance", n = Inf, ...)
```

Arguments

object	A `linear_stack` object outputted from `blend_predictions()` or `fit_members()`.
type	A single character string for plot type with values "performance", "members", or "weights".
n	An integer for how many members weights to plot when `type = "weights"`. With multi-class data, this is the total number of weights across classes; otherwise this is equal to the number of members.
...	Not currently used.

Details

A "performance" plot shows the relationship between the lasso penalty and the resampled performance metrics. The latter includes the average number of ensemble members. This plot can be helpful for understanding what penalty values are reasonable.

A "members" plot shows the relationship between the average number of ensemble members and the performance metrics. Each point is for a different penalty value.

Neither of the "performance" or "members" plots are helpful when a single penalty is used.

A "weights" plot shows the blending weights for the top ensemble members. The results are for the final penalty value used to fit the ensemble.

Value

A `ggplot` object.

axe_model_stack *Axing a model_stack.*

Description

Axing a model_stack.

Remove the call.

Remove controls used for training.

Remove the training data.

Remove environments.

Remove fitted values.

Usage

```
## S3 method for class 'model_stack'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'model_stack'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'model_stack'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'model_stack'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'model_stack'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Additional arguments. Currently ignored.

Value

Axed model_stack object.

Examples

```
# build a regression model stack
st <-
  stacks() %>%
  add_candidates(reg_res_lr) %>%
  add_candidates(reg_res_sp) %>%
  blend_predictions() %>%
  fit_members()

# remove any of the "butcherable"
# elements individually
axe_call(st)
axe_ctrl(st)
axe_data(st)
axe_fitted(st)
axe_env(st)

# or do it all at once!
butchered_st <- butcher(st, verbose = TRUE)

format(object.size(st))
format(object.size(butchered_st))
```

`blend_predictions` *Determine stacking coefficients from a data stack*

Description

Evaluates a data stack by fitting a regularized model on the assessment predictions from each candidate member to predict the true outcome.

This process determines the "stacking coefficients" of the model stack. The stacking coefficients are used to weight the predictions from each candidate (represented by a unique column in the data stack), and are given by the betas of a LASSO model fitting the true outcome with the predictions given in the remaining columns of the data stack.

Candidates with non-zero stacking coefficients are model stack members, and need to be trained on the full training set (rather than just the assessment set) with `fit_members()`. This function is typically used after a number of calls to `add_candidates()`.

Usage

```
blend_predictions(
  data_stack,
  penalty = 10^(-6:-1),
  mixture = 1,
  non_negative = TRUE,
```

```
  metric = NULL,
  control = tune::control_grid(),
  times = 25,
  ...
)
```

Arguments

data_stack	A data_stack object
penalty	A numeric vector of proposed values for total amount of regularization used in member weighting. Higher penalties will generally result in fewer members being included in the resulting model stack, and vice versa. The package will tune over a grid formed from the cross product of the penalty and mixture arguments.
mixture	A number between zero and one (inclusive) giving the proportion of L1 regularization (i.e. lasso) in the model. mixture = 1 indicates a pure lasso model, mixture = 0 indicates ridge regression, and values in (0, 1) indicate an elastic net. The package will tune over a grid formed from the cross product of the penalty and mixture arguments.
non_negative	A logical giving whether to restrict stacking coefficients to non-negative values. If TRUE (default), 0 is passed as the lower.limits argument to glmnet::glmnet() in fitting the model on the data stack. Otherwise, -Inf.
metric	A call to yardstick::metric_set(). The metric(s) to use in tuning the lasso penalty on the stacking coefficients. Default values are determined by tune::tune_grid() from the outcome class.
control	An object inheriting from control_grid to be passed to the model determining stacking coefficients. See tune::control_grid() documentation for details on possible values. Note that any extract entry will be overwritten internally.
times	Number of bootstrap samples tuned over by the model that determines stacking coefficients. See rsample::bootstraps() to learn more.
...	Additional arguments. Currently ignored.

Details

Note that a regularized linear model is one of many possible learning algorithms that could be used to fit a stacked ensemble model. For implementations of additional ensemble learning algorithms, see h2o::h2o.stackedEnsemble() and SuperLearner::SuperLearner().

Value

A model_stack object—while model_stacks largely contain the same elements as data_stacks, the primary data objects shift from the assessment set predictions to the member models.

Example Data

This package provides some resampling objects and datasets for use in examples and vignettes derived from a study on 1212 red-eyed tree frog embryos!

Red-eyed tree frog (RETF) embryos can hatch earlier than their normal 7ish days if they detect potential predator threat. Researchers wanted to determine how, and when, these tree frog embryos were able to detect stimulus from their environment. To do so, they subjected the embryos at varying developmental stages to "predator stimulus" by jiggling the embryos with a blunt probe. Beforehand, though some of the embryos were treated with gentamicin, a compound that knocks out their lateral line (a sensory organ.) Researcher Julie Jung and her crew found that these factors inform whether an embryo hatches prematurely or not!

Note that the data included with the stacks package is not necessarily a representative or unbiased subset of the complete dataset, and is only for demonstrative purposes.

`reg_folds` and `class_folds` are `rset` cross-fold validation objects from `rsample`, splitting the training data into for the regression and classification model objects, respectively. `tree_frogs_reg_test` and `tree_frogs_class_test` are the analogous testing sets.

`reg_res_lr`, `reg_res_svm`, and `reg_res_sp` contain regression tuning results for a linear regression, support vector machine, and spline model, respectively, fitting `latency` (i.e. how long the embryos took to hatch in response to the jiggle) in the `tree_frogs` data, using most all of the other variables as predictors. Note that the data underlying these models is filtered to include data only from embryos that hatched in response to the stimulus.

`class_res_rf` and `class_res_nn` contain multiclass classification tuning results for a random forest and neural network classification model, respectively, fitting `reflex` (a measure of ear function) in the data using most all of the other variables as predictors.

`log_res_rf` and `log_res_nn`, contain binary classification tuning results for a random forest and neural network classification model, respectively, fitting `hatched` (whether or not the embryos hatched in response to the stimulus) using most all of the other variables as predictors.

See `?example_data` to learn more about these objects, as well as browse the source code that generated them.

See Also

Other core verbs: `add_candidates()`, `fit_members()`, `stacks()`

Examples

```
# see the "Example Data" section above for
# clarification on the objects used in these examples!

# put together a data stack
reg_st <-
  stacks() %>%
  add_candidates(reg_res_lr) %>%
  add_candidates(reg_res_svm) %>%
  add_candidates(reg_res_sp)

reg_st

# evaluate the data stack
reg_st %>%
```

```
  blend_predictions()

# include fewer models by proposing higher penalties
reg_st %>%
  blend_predictions(penalty = c(.5, 1))

# allow for negative stacking coefficients
# with the non_negative argument
reg_st %>%
  blend_predictions(non_negative = FALSE)

# use a custom metric in tuning the lasso penalty
library(yardstick)
reg_st %>%
  blend_predictions(metric = metric_set(rmse))

# pass control options for stack blending
reg_st %>%
  blend_predictions(
    control = tune::control_grid(allow_par = TRUE)
  )

# to speed up the stacking process for preliminary
# results, bump down the `times` argument:
reg_st %>%
  blend_predictions(times = 5)

# the process looks the same with
# multinomial classification models
class_st <-
  stacks() %>%
  add_candidates(class_res_nn) %>%
  add_candidates(class_res_rf) %>%
  blend_predictions()

class_st

# ...or binomial classification models
log_st <-
  stacks() %>%
  add_candidates(log_res_nn) %>%
  add_candidates(log_res_rf) %>%
  blend_predictions()

log_st
```

`collect_parameters` *Collect candidate parameters and stacking coefficients*

Description

A function to help situate candidates within a stack. Takes in a data stack or model stack and candidate name and returns a tibble mapping the candidate/member names to their hyperparameters (and, if a model stack, to their stacking coefficients as well).

Usage

```
collect_parameters(stack, candidates, ...)

## Default S3 method:
collect_parameters(stack, candidates, ...)

## S3 method for class 'data_stack'
collect_parameters(stack, candidates, ...)

## S3 method for class 'model_stack'
collect_parameters(stack, candidates, ...)
```

Arguments

`stack`	A `data_stack` or `model_stack` object.
`candidates`	The name of the candidates to collect parameters on. This will either be the `name` argument supplied to `add_candidates()` or, if not supplied, the name of the object supplied to the `candidates` argument in `add_candidates()`.
`...`	Additional arguments. Currently ignored.

Value

A tibble::tbl_df with information on member names and hyperparameters.

Example Data

This package provides some resampling objects and datasets for use in examples and vignettes derived from a study on 1212 red-eyed tree frog embryos!

Red-eyed tree frog (RETF) embryos can hatch earlier than their normal 7ish days if they detect potential predator threat. Researchers wanted to determine how, and when, these tree frog embryos were able to detect stimulus from their environment. To do so, they subjected the embryos at varying developmental stages to "predator stimulus" by jiggling the embryos with a blunt probe. Beforehand, though some of the embryos were treated with gentamicin, a compound that knocks out their lateral line (a sensory organ.) Researcher Julie Jung and her crew found that these factors inform whether an embryo hatches prematurely or not!

Note that the data included with the stacks package is not necessarily a representative or unbiased subset of the complete dataset, and is only for demonstrative purposes.

`reg_folds` and `class_folds` are `rset` cross-fold validation objects from `rsample`, splitting the training data into for the regression and classification model objects, respectively. `tree_frogs_reg_test` and `tree_frogs_class_test` are the analogous testing sets.

reg_res_lr, reg_res_svm, and reg_res_sp contain regression tuning results for a linear regression, support vector machine, and spline model, respectively, fitting latency (i.e. how long the embryos took to hatch in response to the jiggle) in the tree_frogs data, using most all of the other variables as predictors. Note that the data underlying these models is filtered to include data only from embryos that hatched in response to the stimulus.

class_res_rf and class_res_nn contain multiclass classification tuning results for a random forest and neural network classification model, respectively, fitting reflex (a measure of ear function) in the data using most all of the other variables as predictors.

log_res_rf and log_res_nn, contain binary classification tuning results for a random forest and neural network classification model, respectively, fitting hatched (whether or not the embryos hatched in response to the stimulus) using most all of the other variables as predictors.

See ?example_data to learn more about these objects, as well as browse the source code that generated them.

Examples

```
# see the "Example Data" section above for
# clarification on the objects used in these examples!

# put together a data stack using
# tuning results for regression models
reg_st <-
  stacks() %>%
  add_candidates(reg_res_lr) %>%
  add_candidates(reg_res_svm) %>%
  add_candidates(reg_res_sp, "spline")

reg_st

# check out the hyperparameters for some of the candidates
collect_parameters(reg_st, "reg_res_svm")

collect_parameters(reg_st, "spline")

# blend the data stack to view the hyperparameters
# along with the stacking coefficients!
collect_parameters(
  reg_st %>% blend_predictions(),
  "spline"
)
```

---

control_stack_grid *Control wrappers*

Description

Supply these light wrappers as the `control` argument in a `tune::tune_grid()`, `tune::tune_bayes()`, or `tune::fit_resamples()` call to return the needed elements for use in a data stack. These functions will return the appropriate control grid to ensure that assessment set predictions and information on model specifications and preprocessors, is supplied in the resampling results object!

To integrate stack settings with your existing control settings, note that these functions just call the appropriate `tune::control_*` function with the arguments `save_pred = TRUE, save_workflow = TRUE`.

Usage

```
control_stack_grid()

control_stack_resamples()

control_stack_bayes()
```

Value

A tune::control_grid, tune::control_bayes, or tune::control_resamples object.

See Also

See example_data for examples of these functions used in context.

---

example_data *Example Objects*

Description

stacks provides some resampling objects and datasets for use in examples and vignettes derived from a study on 1212 red-eyed tree frog embryos!

Usage

```
reg_res_svm

reg_res_sp

reg_res_lr

reg_folds

class_res_nn

class_res_rf

class_folds
```

log_res_nn

log_res_rf

Format

An object of class tune_results (inherits from tbl_df, tbl, data.frame) with 5 rows and 5 columns.

An object of class tune_results (inherits from tbl_df, tbl, data.frame) with 5 rows and 5 columns.

An object of class resample_results (inherits from tune_results, tbl_df, tbl, data.frame) with 5 rows and 5 columns.

An object of class vfold_cv (inherits from rset, tbl_df, tbl, data.frame) with 5 rows and 2 columns.

An object of class resample_results (inherits from tune_results, tbl_df, tbl, data.frame) with 5 rows and 5 columns.

An object of class tune_results (inherits from tbl_df, tbl, data.frame) with 5 rows and 5 columns.

An object of class vfold_cv (inherits from rset, tbl_df, tbl, data.frame) with 5 rows and 2 columns.

An object of class resample_results (inherits from tune_results, tbl_df, tbl, data.frame) with 5 rows and 5 columns.

An object of class tune_results (inherits from tbl_df, tbl, data.frame) with 5 rows and 5 columns.

Details

Red-eyed tree frog (RETF) embryos can hatch earlier than their normal 7ish days if they detect potential predator threat. Researchers wanted to determine how, and when, these tree frog embryos were able to detect stimulus from their environment. To do so, they subjected the embryos at varying developmental stages to "predator stimulus" by jiggling the embryos with a blunt probe. Beforehand, though some of the embryos were treated with gentamicin, a compound that knocks out their lateral line (a sensory organ.) Researcher Julie Jung and her crew found that these factors inform whether an embryo hatches prematurely or not!

Note that the data included with the stacks package is not necessarily a representative or unbiased subset of the complete dataset, and is only for demonstrative purposes.

reg_folds and class_folds are rset cross-fold validation objects from rsample, splitting the training data into for the regression and classification model objects, respectively. tree_frogs_reg_test and tree_frogs_class_test are the analogous testing sets.

reg_res_lr, reg_res_svm, and reg_res_sp contain regression tuning results for a linear regression, support vector machine, and spline model, respectively, fitting latency (i.e. how long the embryos took to hatch in response to the jiggle) in the tree_frogs data, using most all of the other variables as predictors. Note that the data underlying these models is filtered to include data only from embryos that hatched in response to the stimulus.

class_res_rf and class_res_nn contain multiclass classification tuning results for a random forest and neural network classification model, respectively, fitting reflex (a measure of ear function) in the data using most all of the other variables as predictors.

log_res_rf and log_res_nn, contain binary classification tuning results for a random forest and neural network classification model, respectively, fitting hatched (whether or not the embryos hatched in response to the stimulus) using most all of the other variables as predictors.

The source code for generating these objects is given below.

```
# setup: packages, data, resample, basic recipe ------------------------
library(stacks)
library(tune)
library(rsample)
library(parsnip)
library(workflows)
library(recipes)
library(yardstick)
library(workflowsets)

set.seed(1)

ctrl_grid <-
  tune::control_grid(
    save_pred = TRUE,
    save_workflow = TRUE
  )

ctrl_res <-
  tune::control_resamples(
    save_pred = TRUE,
    save_workflow = TRUE
  )

# for regression, predict latency to hatch (excluding NAs)
tree_frogs_reg <-
  tree_frogs %>%
  filter(!is.na(latency)) %>%
  select(-clutch, -hatched)

set.seed(1)
tree_frogs_reg_split <- rsample::initial_split(tree_frogs_reg)

set.seed(1)
tree_frogs_reg_train <- rsample::training(tree_frogs_reg_split)

set.seed(1)
tree_frogs_reg_test  <- rsample::testing(tree_frogs_reg_split)

set.seed(1)
```

```
reg_folds <- rsample::vfold_cv(tree_frogs_reg_train, v = 5)

tree_frogs_reg_rec <-
  recipes::recipe(latency ~ ., data = tree_frogs_reg_train) %>%
  recipes::step_dummy(recipes::all_nominal()) %>%
  recipes::step_zv(recipes::all_predictors())

metric <- yardstick::metric_set(yardstick::rmse)

# linear regression ---------------------------------------
lin_reg_spec <-
  parsnip::linear_reg() %>%
  parsnip::set_engine("lm")

reg_wf_lr <-
  workflows::workflow() %>%
  workflows::add_model(lin_reg_spec) %>%
  workflows::add_recipe(tree_frogs_reg_rec)

set.seed(1)
reg_res_lr <-
  tune::fit_resamples(
    object = reg_wf_lr,
    resamples = reg_folds,
    metrics = metric,
    control = ctrl_res
  )

# SVM regression ----------------------------------
svm_spec <-
  parsnip::svm_rbf(
    cost = tune::tune(),
    rbf_sigma = tune::tune()
  ) %>%
  parsnip::set_engine("kernlab") %>%
  parsnip::set_mode("regression")

reg_wf_svm <-
  workflows::workflow() %>%
  workflows::add_model(svm_spec) %>%
  workflows::add_recipe(tree_frogs_reg_rec)

set.seed(1)
reg_res_svm <-
  tune::tune_grid(
    object = reg_wf_svm,
    resamples = reg_folds,
    grid = 5,
```

```
    control = ctrl_grid
  )

# spline regression ---------------------------------------
spline_rec <-
  tree_frogs_reg_rec %>%
  recipes::step_ns(age, deg_free = tune::tune("age"))

reg_wf_sp <-
  workflows::workflow() %>%
  workflows::add_model(lin_reg_spec) %>%
  workflows::add_recipe(spline_rec)

set.seed(1)
reg_res_sp <-
  tune::tune_grid(
    object = reg_wf_sp,
    resamples = reg_folds,
    metrics = metric,
    control = ctrl_grid
  )

# classification - preliminaries -----------------------------------
tree_frogs_class <-
  tree_frogs %>%
  dplyr::select(-c(clutch, latency))

set.seed(1)
tree_frogs_class_split <- rsample::initial_split(tree_frogs_class)

set.seed(1)
tree_frogs_class_train <- rsample::training(tree_frogs_class_split)

set.seed(1)
tree_frogs_class_test  <- rsample::testing(tree_frogs_class_split)

set.seed(1)
class_folds <- rsample::vfold_cv(tree_frogs_class_train, v = 5)

tree_frogs_class_rec <-
  recipes::recipe(reflex ~ ., data = tree_frogs_class_train) %>%
  recipes::step_dummy(recipes::all_nominal(), -reflex) %>%
  recipes::step_zv(recipes::all_predictors()) %>%
  recipes::step_normalize(recipes::all_numeric())

# random forest classification --------------------------------------
rand_forest_spec <-
  parsnip::rand_forest(
```

```
    mtry = tune::tune(),
    trees = 500,
    min_n = tune::tune()
  ) %>%
  parsnip::set_mode("classification") %>%
  parsnip::set_engine("ranger")

class_wf_rf <-
  workflows::workflow() %>%
  workflows::add_recipe(tree_frogs_class_rec) %>%
  workflows::add_model(rand_forest_spec)

set.seed(1)
class_res_rf <-
  tune::tune_grid(
    object = class_wf_rf,
    resamples = class_folds,
    grid = 10,
    control = ctrl_grid
  )

# neural network classification ------------------------------------
nnet_spec <-
  mlp(hidden_units = 5, penalty = 0.01, epochs = 100) %>%
  set_mode("classification") %>%
  set_engine("nnet")

class_wf_nn <-
  workflows::workflow() %>%
  workflows::add_recipe(tree_frogs_class_rec) %>%
  workflows::add_model(nnet_spec)

set.seed(1)
class_res_nn <-
  tune::fit_resamples(
    object = class_wf_nn,
    resamples = class_folds,
    control = ctrl_res
  )

# binary classification -------------------------------
tree_frogs_2_class_rec <-
  recipes::recipe(hatched ~ ., data = tree_frogs_class_train) %>%
  recipes::step_dummy(recipes::all_nominal(), -hatched) %>%
  recipes::step_zv(recipes::all_predictors()) %>%
  recipes::step_normalize(recipes::all_numeric())

set.seed(1)
```

```
rand_forest_spec_2 <-
  parsnip::rand_forest(
    mtry = tune(),
    trees = 500,
    min_n = tune()
  ) %>%
  parsnip::set_mode("classification") %>%
  parsnip::set_engine("ranger")

log_wf_rf <-
  workflows::workflow() %>%
  workflows::add_recipe(tree_frogs_2_class_rec) %>%
  workflows::add_model(rand_forest_spec_2)

set.seed(1)
log_res_rf <-
  tune::tune_grid(
    object = log_wf_rf,
    resamples = class_folds,
    grid = 10,
    control = ctrl_grid
  )

nnet_spec_2 <-
  parsnip::mlp(epochs = 100, hidden_units = 5, penalty = 0.1) %>%
  parsnip::set_mode("classification") %>%
  parsnip::set_engine("nnet", verbose = 0)

log_wf_nn <-
  workflows::workflow() %>%
  workflows::add_recipe(tree_frogs_2_class_rec) %>%
  workflows::add_model(nnet_spec_2)

set.seed(1)
log_res_nn <-
  tune::fit_resamples(
    object = log_wf_nn,
    resamples = class_folds,
    control = ctrl_res
  )
```

Source

Julie Jung et al. (2020) Multimodal mechanosensing enables treefrog embryos to escape egg-predators. doi:10.1242/jeb.236141 `fit_members` *Fit model stack members with non-zero stacking coefficients*

Description

After evaluating a data stack with `blend_predictions()`, some number of candidates will have nonzero stacking coefficients. Such candidates are referred to as "members." Since members' predictions will ultimately inform the model stack's predictions, members should be trained on the full training set using `fit_members()`.

Usage

```
fit_members(model_stack, ...)
```

Arguments

`model_stack`	A `model_stack` object outputted by `blend_predictions()`.
`...`	Additional arguments. Currently ignored.

Details

To fit members in parallel, please register a parallel backend function. See the documentation of `foreach::foreach()` for examples.

Value

A `model_stack` object with a subclass `linear_stack`—this fitted model contains the necessary components to predict on new data.

Example Data

This package provides some resampling objects and datasets for use in examples and vignettes derived from a study on 1212 red-eyed tree frog embryos!

Red-eyed tree frog (RETF) embryos can hatch earlier than their normal 7ish days if they detect potential predator threat. Researchers wanted to determine how, and when, these tree frog embryos were able to detect stimulus from their environment. To do so, they subjected the embryos at varying developmental stages to "predator stimulus" by jiggling the embryos with a blunt probe. Beforehand, though some of the embryos were treated with gentamicin, a compound that knocks out their lateral line (a sensory organ.) Researcher Julie Jung and her crew found that these factors inform whether an embryo hatches prematurely or not!

Note that the data included with the stacks package is not necessarily a representative or unbiased subset of the complete dataset, and is only for demonstrative purposes.

`reg_folds` and `class_folds` are `rset` cross-fold validation objects from `rsample`, splitting the training data into for the regression and classification model objects, respectively. `tree_frogs_reg_test` and `tree_frogs_class_test` are the analogous testing sets.

reg_res_lr, reg_res_svm, and reg_res_sp contain regression tuning results for a linear regression, support vector machine, and spline model, respectively, fitting latency (i.e. how long the embryos took to hatch in response to the jiggle) in the tree_frogs data, using most all of the other variables as predictors. Note that the data underlying these models is filtered to include data only from embryos that hatched in response to the stimulus.

class_res_rf and class_res_nn contain multiclass classification tuning results for a random forest and neural network classification model, respectively, fitting reflex (a measure of ear function) in the data using most all of the other variables as predictors.

log_res_rf and log_res_nn, contain binary classification tuning results for a random forest and neural network classification model, respectively, fitting hatched (whether or not the embryos hatched in response to the stimulus) using most all of the other variables as predictors.

See ?example_data to learn more about these objects, as well as browse the source code that generated them.

See Also

Other core verbs: add_candidates(), blend_predictions(), stacks()

Examples

```
# see the "Example Data" section above for
# clarification on the objects used in these examples!

# put together a data stack
reg_st <-
  stacks() %>%
  add_candidates(reg_res_lr) %>%
  add_candidates(reg_res_svm) %>%
  add_candidates(reg_res_sp)

reg_st

# evaluate the data stack and fit the member models
reg_st %>%
  blend_predictions() %>%
  fit_members()

reg_st

# do the same with multinomial classification models
class_st <-
  stacks() %>%
  add_candidates(class_res_nn) %>%
  add_candidates(class_res_rf) %>%
  blend_predictions() %>%
  fit_members()
```

```
class_st

# ...or binomial classification models
log_st <-
  stacks() %>%
  add_candidates(log_res_nn) %>%
  add_candidates(log_res_rf) %>%
  blend_predictions() %>%
  fit_members()

log_st
```

---

get_expressions *Obtain prediction equations for all possible values of type*

Description

Obtain prediction equations for all possible values of type

Usage

```
get_expressions(x, ...)

## S3 method for class '`_multnet`'
get_expressions(x, ...)

## S3 method for class '`_lognet`'
get_expressions(x, ...)

## S3 method for class '`_elnet`'
get_expressions(x, ...)
```

Arguments

x	A `parsnip` model with the `glmnet` engine.
...	Not used

Value

A named list with prediction equations for each possibel type.

predict.data_stack *Predicting with a model stack*

Description

The data stack must be evaluated with `blend_predictions()` and its member models fitted with `fit_members()` to predict on new data.

Usage

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

Arguments

`object`	A data stack.
`...`	Additional arguments. Currently ignored.

predict.model_stack *Predicting with a model stack*

Description

Apply a model stack to create different types of predictions.

Usage

```
## S3 method for class 'model_stack'
predict(object, new_data, type = NULL, members = FALSE, opts = list(), ...)
```

Arguments

`object`	A model stack with fitted members outputted from `fit_members()`.
`new_data`	A rectangular data object, such as a data frame.
`type`	Format of returned predicted values—one of "numeric", "class", or "prob". When NULL, `predict()` will choose an appropriate value based on the model's mode.
`members`	Logical. Whether or not to additionally return the predictions for each of the ensemble members.
`opts`	A list of optional arguments to the underlying predict function passed on to parsnip::predict.model_fit for each member.
`...`	Additional arguments. Currently ignored.

Example Data

This package provides some resampling objects and datasets for use in examples and vignettes derived from a study on 1212 red-eyed tree frog embryos!

Red-eyed tree frog (RETF) embryos can hatch earlier than their normal 7ish days if they detect potential predator threat. Researchers wanted to determine how, and when, these tree frog embryos were able to detect stimulus from their environment. To do so, they subjected the embryos at varying developmental stages to "predator stimulus" by jiggling the embryos with a blunt probe. Beforehand, though some of the embryos were treated with gentamicin, a compound that knocks out their lateral line (a sensory organ.) Researcher Julie Jung and her crew found that these factors inform whether an embryo hatches prematurely or not!

Note that the data included with the stacks package is not necessarily a representative or unbiased subset of the complete dataset, and is only for demonstrative purposes.

`reg_folds` and `class_folds` are `rset` cross-fold validation objects from `rsample`, splitting the training data into for the regression and classification model objects, respectively. `tree_frogs_reg_test` and `tree_frogs_class_test` are the analogous testing sets.

`reg_res_lr`, `reg_res_svm`, and `reg_res_sp` contain regression tuning results for a linear regression, support vector machine, and spline model, respectively, fitting `latency` (i.e. how long the embryos took to hatch in response to the jiggle) in the `tree_frogs` data, using most all of the other variables as predictors. Note that the data underlying these models is filtered to include data only from embryos that hatched in response to the stimulus.

`class_res_rf` and `class_res_nn` contain multiclass classification tuning results for a random forest and neural network classification model, respectively, fitting `reflex` (a measure of ear function) in the data using most all of the other variables as predictors.

`log_res_rf` and `log_res_nn`, contain binary classification tuning results for a random forest and neural network classification model, respectively, fitting `hatched` (whether or not the embryos hatched in response to the stimulus) using most all of the other variables as predictors.

See `?example_data` to learn more about these objects, as well as browse the source code that generated them.

Examples

```
# see the "Example Data" section above for
# clarification on the data and tuning results
# objects used in these examples!

data(tree_frogs_reg_test)
data(tree_frogs_class_test)

# build and fit a regression model stack
reg_st <-
  stacks() %>%
  add_candidates(reg_res_lr) %>%
  add_candidates(reg_res_sp) %>%
  blend_predictions() %>%
```

```
  fit_members()

reg_st

# predict on the tree frogs testing data
predict(reg_st, tree_frogs_reg_test)

# include the predictions from the members
predict(reg_st, tree_frogs_reg_test, members = TRUE)

# build and fit a classification model stack
class_st <-
  stacks() %>%
  add_candidates(class_res_nn) %>%
  add_candidates(class_res_rf) %>%
  blend_predictions() %>%
  fit_members()

class_st

# predict reflex, first as a class, then as
# class probabilities
predict(class_st, tree_frogs_class_test)
predict(class_st, tree_frogs_class_test, type = "prob")

# returning the member predictions as well
predict(
  class_st,
  tree_frogs_class_test,
  type = "prob",
  members = TRUE
)
```

`stacks` *Initialize a Stack*

Description

The `stacks()` function initializes a `data_stack` object. Principally, `data_stacks` are tibbles, where the first column gives the true outcome in the assessment set, and the remaining columns give the predictions from each candidate ensemble member. (When the outcome is numeric, there's only one column per candidate member. For classification, there are as many columns per candidate member as there are levels in the outcome variable minus 1.) They also bring along a few extra attributes to keep track of model definitions, resamples, and training data.

See `?stacks_description` for more discussion of the package, generally, and the `basics` vignette for a detailed walk-through of functionality.

Usage

```
stacks(...)
```

Arguments

`...`	Additional arguments. Currently ignored.

Value

A `data_stack` object.

See Also

Other core verbs: `add_candidates()`, `blend_predictions()`, `fit_members()`

`stacks_description` *stacks: Tidy Model Stacking*

Description

Model stacking is an ensemble technique that involves training a model to combine the outputs of many diverse statistical models, and has been shown to improve predictive performance in a variety of settings. 'stacks' implements a grammar for 'tidymodels'-aligned model stacking.

Author(s)

Maintainer: Simon Couch <`simonpatrickcouch@gmail.com`>

Authors:

- Max Kuhn <`max@rstudio.com`>

Other contributors:

- RStudio [copyright holder, funder]

See Also

Useful links:

- `https://stacks.tidymodels.org/`
- `https://github.com/tidymodels/stacks`
- Report bugs at `https://github.com/tidymodels/stacks/issues`

`tree_frogs` *Tree frog embryo hatching data*

Description

A dataset containing experimental results on hatching behavior of red-eyed tree frog embryos.

Red-eyed tree frog (RETF) embryos can hatch earlier than their normal 7ish days if they detect potential predator threat. Researchers wanted to determine how, and when, these tree frog embryos were able to detect stimulus from their environment. To do so, they subjected the embryos at varying developmental stages to "predator stimulus" by jiggling the embryos with a blunt probe. Beforehand, though some of the embryos were treated with gentamicin, a compound that knocks out their lateral line (a sensory organ.) Researcher Julie Jung and her crew found that these factors inform whether an embryo hatches prematurely or not!

Usage

```
tree_frogs
```

Format

A data frame with 1212 rows and 6 variables:

clutch RETFs lay their eggs in gelatinous "clutches" of 30-40 eggs. Eggs with the same clutch ID are siblings of each other! This variable is useful in mixed effects models. (Unordered factor.)

treatment The treatment group for the embryo. Either "gentamicin", a compound that knocks out the embryos' lateral line, or "control" for the negative control group (i.e. sensory organs intact). (Character.)

reflex A measure of ear function called the vestibulo-ocular reflex, categorized into bins. Ear function increases from factor levels "low", to "mid", to "full". (Ordered factor.)

age Age of the embryo, in seconds, at the time that the embryo was jiggled. (Numeric, in seconds.)

t_o_d The time of day that the stimulus (i.e. jiggle) was applied. "morning" is 5 a.m. to noon, "afternoon" is noon to 8 p.m., and "night" is 8 p.m. to 5 a.m. (Character.)

hatched Whether or not the embryo hatched in response to the jiggling! Either "yes" or "no". (Character.)

latency Time elapsed between the stimulus (i.e. jiggling) and hatching in response to the stimulus, in seconds. Missing values indicate that the embryo didn't hatch in response to the stimulus. (Numeric, in seconds.)

Details

Note that the data included with the `stacks` package is not necessarily a representative or unbiased subset of the complete dataset, and is only for demonstrative purposes.

Source

Julie Jung et al. (2020) Multimodal mechanosensing enables treefrog embryos to escape egg-predators. doi:10.1242/jeb.236141

Index