Package 'nestedmodels'

January 5, 2023

Title Tidy Modelling for Nested Data

Version 1.0.3

Description A modelling framework for nested data using the 'tidymodels' ecosystem. Specify how to nest data using the 'recipes' package, create testing and training splits using 'rsample', and fit models to this data using the 'parsnip' and 'workflows' packages. Allows any model to be fit to nested data.

License MIT + file LICENSE

URL https://github.com/ashbythorpe/nestedmodels, https://ashbythorpe.github.io/nestedmodels/

BugReports https://github.com/ashbythorpe/nestedmodels/issues

Depends R (>= 2.10)

Imports cli, dplyr, generics, glue, lifecycle, magrittr, parsnip, purrr, recipes, rlang (>= 0.4.11), rsample, stats, stringr, tibble, tidyr (> 0.8.99), vctrs

Suggests broom, covr, glmnet, hardhat, knitr, rmarkdown, roxygen2, testthat (>= 3.0.0), tidyselect, tune, utils, withr, workflows

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Ashby Thorpe [aut, cre] (<https://orcid.org/0000-0003-3106-099X>), Hadley Wickham [ctb] (<https://orcid.org/0000-0003-4757-117X>)

Maintainer Ashby Thorpe <ashbythorpe@gmail.com>

Repository CRAN

Date/Publication 2023-01-05 21:20:02 UTC

R topics documented:

`augment.nested_model_fit` *Augment data with predictions*

Description

`generics::augment()` method for nested models. `augment.nested_model_fit()` will add column(s) for predictions to the given data.

Usage

```
## S3 method for class 'nested_model_fit'
augment(x, new_data, ...)
```

Arguments

`x`	A `nested_model_fit` object produced by `fit.nested_model()`.
`new_data`	A data frame - can be nested or non-nested.
`...`	Passed onto `parsnip::augment.model_fit()`.

Value

A data frame with one or more added columns for predictions.

See Also

`parsnip::augment.model_fit()`

Examples

```
data <- dplyr::filter(example_nested_data, id %in% 1:5)

nested_data <- tidyr::nest(data, data = -c(id, id2))

model <- parsnip::linear_reg() %>%
  parsnip::set_engine("lm") %>%
  nested()

fitted <- fit(model, z ~ x + y + a + b, nested_data)

augment(fitted, example_nested_data)
```

---

example_nested_data *Example nested data*

Description

A dataset containing example data that can be nested. Mainly used for examples and testing.

Usage

```
example_nested_data
```

Format

A tibble with 1000 rows and 7 variables

id A column that can be nested, ranging from 1 to 20.

id2 Another column that can be nested, ranging from 1 to 10.

x A numeric column that depends on 'id'.

y A sequential numeric column (with some added randomness), independent of the other columns.

z A column dependent on id, id2, x and y.

a A randomly generated numeric column, ranging from 1 to 100.

b A randomly generated numeric column, centred around 50.

Examples

```
example_nested_data
```

`extract_inner_model` *Get the inner model of a nested model object*

Description

Extract the inner model of a `nested_model` object, or a workflow containing a nested model.

Usage

```
extract_inner_model(x, ...)

## Default S3 method:
extract_inner_model(x, ...)

## S3 method for class 'nested_model'
extract_inner_model(x, ...)

## S3 method for class 'workflow'
extract_inner_model(x, ...)

## S3 method for class 'model_spec'
extract_inner_model(x, ...)
```

Arguments

`x`	A model spec or workflow.
`...`	Not used.

Value

A `model_spec` object

Examples

```
model <- parsnip::linear_reg() %>%
  parsnip::set_engine("lm") %>%
  nested()

extract_inner_model(model)
```

`fit.nested_model` *Fit a nested model to a dataset*

Description

`fit.model_spec()` takes a nested model specification and fits the inner model specification to each nested data frame in the given dataset.

Usage

```
## S3 method for class 'nested_model'
fit(
  object,
  formula,
  data,
  case_weights = NULL,
  control = parsnip::control_parsnip(),
  ...
)
```

Arguments

`object`	An `nested_model` object (see `nested()`).
`formula`	An object of class `formula`. Passed into `parsnip::fit.model_spec()`. This should *not contain* the variable to nest by.
`data`	A data frame. If used with a 'nested_model' object, the data frame must already be nested.
`case_weights`	An optional vector of case weights. Passed into `parsnip::fit.model_spec()`.
`control`	A `parsnip::control_parsnip()` object. Passed into `parsnip::fit.model_spec()`.
`...`	Passed into `parsnip::fit.model_spec()`. Currently unused.

Value

A `nested_model_fit` object with several elements:

- `spec`: The model specification object (the inner model of the nested model object)
- `fit`: A tibble containing the model fits and the nests that they correspond to.
- `inner_names`: A character vector of names, used to help with nesting the data during predictions.

See Also

`parsnip::fit.model_spec()` parsnip::model_fit

Examples

```
model <- parsnip::linear_reg() %>%
  parsnip::set_engine("lm") %>%
  nested()

nested_data <- tidyr::nest(example_nested_data, data = -id)

fit(model, z ~ x + y + a + b, nested_data)
```

`fit_xy.nested_model` *Fit a nested model to a dataset using an xy interface.*

Description

`generics::fit_xy()` method for nested models. This should not be called directly and instead should be called by `workflows::fit.workflow()`.

Usage

```
## S3 method for class 'nested_model'
fit_xy(
  object,
  x,
  y,
  case_weights = NULL,
  control = parsnip::control_parsnip(),
  ...
)
```

Arguments

`object`	An `nested_model` object (see `nested()`).
`x`	A data frame of predictors.
`y`	A data frame of outcome data.
`case_weights`	An optional vector of case weights. Passed into `parsnip::fit.model_spec()`.
`control`	A `parsnip::control_parsnip()` object. Passed into `parsnip::fit.model_spec()`.
`...`	Passed into `parsnip::fit.model_spec()`. Currently unused.

Value

A `nested_model_fit` object with several elements:

- `spec`: The model specification object (the inner model of the nested model object)
- `fit`: A tibble containing the model fits and the nests that they correspond to.
- `inner_names`: A character vector of names, used to help with nesting the data during predictions.

See Also

parsnip::fit.model_spec() parsnip::model_fit

Examples

```
data <- dplyr::filter(example_nested_data, id %in% 11:20)

model <- parsnip::linear_reg() %>%
  parsnip::set_engine("lm") %>%
  nested()

recipe <- recipes::recipe(data, z ~ x + y + id) %>%
  step_nest(id)

wf <- workflows::workflow() %>%
  workflows::add_recipe(recipe) %>%
  workflows::add_model(model)

fit(wf, data)
```

---

multi_predict.nested_model_fit

Nested model predictions across many sub-models

Description

parsnip::multi_predict() method for nested models. Allows predictions to be made on sub-models in a model object.

Usage

```
## S3 method for class 'nested_model_fit'
multi_predict(object, new_data, ...)
```

Arguments

object	A nested_model_fit object produced by fit.nested_model().
new_data	A data frame - can be nested or non-nested.
...	Passed onto parsnip::multi_predict()

Value

A tibble with the same number of rows as new_data, after it has been unnested.

See Also

parsnip::multi_predict()

Examples

```
data <- dplyr::filter(example_nested_data, id %in% 16:20)

nested_data <- tidyr::nest(data, data = -id2)

model <- parsnip::linear_reg(penalty = 1) %>%
  parsnip::set_engine("glmnet") %>%
  nested()

fitted <- fit(model, z ~ x + y + a + b, nested_data)

parsnip::multi_predict(fitted, example_nested_data,
  penalty = c(0.1, 0.2, 0.3)
)
```

`nested` *Create a Nested Model*

Description

`nested()` turns a model or workflow into a nested model/workflow. `is_nested()` checks if a model or workflow is nested.

Usage

```
nested(x, ...)

is_nested(x, ...)

## Default S3 method:
nested(x, ...)

## S3 method for class 'model_spec'
nested(x, ...)

## S3 method for class 'nested_model'
nested(x, ...)

## S3 method for class 'workflow'
nested(x, ...)

## Default S3 method:
is_nested(x, ...)

## S3 method for class 'model_spec'
is_nested(x, ...)
```

```
## S3 method for class 'workflow'
is_nested(x, ...)
```

Arguments

x	A model specification or workflow.
...	Not currently used.

Value

A nested model object, or a workflow containing a nested model. For `is_nested()`, a logical vector of length 1.

Examples

```
model <- parsnip::linear_reg() %>%
  parsnip::set_engine("lm") %>%
  nested()

model

is_nested(model)

wf <- workflows::workflow() %>%
  workflows::add_model(model)

is_nested(wf)
```

---

nested_resamples *Create splits with nested data*

Description

Use any rsample split function on nested data, where nests act as strata. This almost guarantees that every split will contain data from every nested data frame.

Usage

```
nested_resamples(
  data,
  resamples,
  nesting_method = NULL,
 size_action = c("truncate", "recycle", "recycle-random", "combine", "combine-random",
    "combine-end", "error"),
  ...
)
```

Arguments

data	A data frame.
resamples	An expression, function, formula or string that can be evaluated to produce an rset or rsplit object.
nesting_method	A recipe, workflow or NULL, used to nest data if data is not already nested (see Details).
size_action	If different numbers of splits are produced in each nest, how should sizes be matched? (see Details)
...	Extra arguments to pass into resamples.

Details

This function breaks down a data frame into smaller, nested data frames. Resampling is then performed within these nests, and the results are combined together at the end. This ensures that each split contains data from every nest. However, this function does not perform any pooling (unlike rsample::make_strata()), so you may run into issues if a nest is too small.

Value

Either an rsplit object or an rset object, depending on resamples.

Nesting Data

data can be nested in several ways: If nesting_method is NULL and data is grouped (using dplyr::group_by()), the data will be nested (see tidyr::nest() for how this works). If data is not grouped, it is assumed to already be nested, and nested_resamples will try to find a column that contains nested data frames. If nesting_method is a workflow or recipe, and the recipe has a step created using step_nest(), data will be nested using the step in the recipe. This is convenient if you've already created a recipe or workflow, as it saves a line of code.

Resample Evaluation

The resamples argument can take many forms:

- A function call, such as vfold_cv(v = 5). This is similar to the format of rsample::nested_cv().
- A function, such as rsample::vfold_cv.
- A purrr-style anonymous function, which will be converted to a function using rlang::as_function().
- A string, which will be evaluated using rlang::exec().

Every method will be evaluated with data passed in as the first argument (with name 'data').

Size Matching

Before the set of resamples created in each nest can be combined, they must contain the same number of splits. For most resampling methods, this will not be an issue. rsample::vfold_cv(), for example, reliably creates the number of splits defined in its 'v' argument. However, other resampling methods, like rsample::rolling_origin(), depend on the size of their 'data' argument, and therefore may produce different numbers of resamples when presented with differently sized nests.

The `size_action` argument defines many ways of matching the sizes of resample sets with different numbers of splits. These methods will either try to reduce the number of splits in each set until each rset is the same length as the set with the lowest number of splits; or the opposite, where each rset will have the same number of splits as the largest set.

"truncate", the default, means that all splits beyond the required length will be removed.

"recycle" means that sets of splits will be extended by repeating elements until the required length has been reached, mimicking the process of vector recycling. The advantage of this method is that all created splits will be preserved.

"recycle-random" is a similar process to recycling, but splits will be copied at random to spaces in the output, which may be important if the order of resamples matters. This process is not completely random, and the program makes sure that every split is copied roughly the same number of times.

"combine" gets rid of excess splits by combining them with previous ones. This means the training and testing rows are merged into one split. Combining is done systematically: if a set of splits needs to be compacted down to a set of 5, the first split is combined with the sixth split, then the eleventh, then the sixteenth, etc. This approach is not recommended, since it is not clear what the benefit of a combined split is.

"combine-random" combines each split with a random set of other splits, instead of the systematic process described in the previous method. Once again, this process is not actually random, and each split will be combined with roughly the same number of other splits.

"combine-end" combines every excess split with the last non-excess split.

"error" throws an error if each nest does not produce the same number of splits.

See Also

`rsample::initial_split()` for an example of the strata argument.

Examples

```
nested_data <- example_nested_data %>%
  tidyr::nest(data = -id)

grouped_data <- example_nested_data %>%
  dplyr::group_by(id)

recipe <- recipes::recipe(example_nested_data, z ~ .) %>%
  step_nest(id)

wf <- workflows::workflow() %>%
  workflows::add_recipe(recipe)

nested_resamples(nested_data, rsample::vfold_cv())

nested_resamples(
  dplyr::group_by(example_nested_data, id),
  ~ rsample::initial_split(.)
)

nested_resamples(example_nested_data, ~ {
```

```
  rsample::validation_split(.)
}, nesting_method = recipe)

nested_resamples(example_nested_data, rsample::bootstraps,
  times = 25, nesting_method = wf
)

# nested nested resamples

nested_resamples(nested_data, rsample::nested_cv(
  rsample::vfold_cv(),
  rsample::bootstraps()
))
```

`predict.nested_model_fit` *Nested Model Predictions*

Description

Apply a fitted nested model to generate different types of predictions. `stats::predict()` / `parsnip::predict_raw()` methods for nested model fits.

Usage

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

## S3 method for class 'nested_model_fit'
predict_raw(object, new_data, opts = list(), ...)
```

Arguments

`object`	A `nested_model_fit` object produced by `fit.nested_model()`.
`new_data`	A data frame to make predictions on. Can be nested or non-nested.
`type`	A singular character vector or `NULL`. Passed on to `parsnip::predict.model_fit()`.
`opts`	A list of optional arguments. Passed on to `parsnip::predict.model_fit()`.
`...`	Arguments for the underlying model's predict function. Passed on to `parsnip::predict.model_fit()`.

Value

A data frame of model predictions. For `predict_raw()`, a matrix, data frame, vector or list.

See Also

`parsnip::predict.model_fit()`

Examples

```
data <- dplyr::filter(example_nested_data, id %in% 5:15)

nested_data <- tidyr::nest(data, data = -id)

model <- parsnip::linear_reg() %>%
  parsnip::set_engine("lm") %>%
  nested()

fitted <- fit(model, z ~ x + y + a + b, nested_data)

predict(fitted, example_nested_data)

parsnip::predict_raw(fitted, example_nested_data)
```

---

step_nest *Nest transformation*

Description

`step_nest()` creates a *specification* of a recipe step that will convert specified data into a single model term, specifying the 'nest' that each row of the dataset corresponds to.

Usage

```
step_nest(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  names = NULL,
  lookup_table = NULL,
  skip = FALSE,
  id = recipes::rand_id("nest")
)
```

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables. For `step_nest`, this indicates the variables which will *not* be nested. See `recipes::selections()` for more details.
`role`	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.

trained	A logical to indicate if the quantities for preprocessing have been estimated.
names	The names of the variables selected by ... are stored here once this preprocessing step has been trained by recipes::prep().
lookup_table	The table describing which values of your selected columns correspond to which 'nest_id' are stored here once this preprocessing step has been trained by recipes::prep().
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? While all operations are baked when recipes::prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

step_nest() will create a single nominal variable (named 'nest_id') from a set of variables (of any type). Every unique combination of the specified columns will receive a single nest id.

This recipe step is designed for use with nested models, since a model will be fitted on the data corresponding to each nest id. Using a recipe is often easier and more reliable than nesting the data manually.

The nest id corresponding to each unique combination of column values is decided when the recipe is prepped (if this recipe is contained in a workflow, this happens when the workflow is fitted). This means that when using a prepped recipe on new data (using recipes::prep() or workflows::predict.workflow()), all unique combinations of nesting columns must also exist in the training data. You will be warned if this is not the case. If you are using the 'rsample' package to create splits and this presents an issue, you may want to consider using nested_resamples().

step_nest() is designed so that nesting the transformed data by its 'nest_id' column is equivalent to the following action on the non-transformed data:

```
data %>%
  dplyr::group_by(...) %>% # '...' represents your specified terms
  tidyr::nest()
```

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

Tidying

When you tidy() this step, a tibble is returned showing how each unique value of the terms you have specified correspond to each nest id.

Case weights

The underlying operation does not allow for case weights.

Examples

```
recipe <- recipes::recipe(example_nested_data, z ~ x + id) %>%
  step_nest(id)

recipe %>%
  recipes::prep() %>%
  recipes::bake(NULL)

recipe2 <- recipes::recipe(example_nested_data, z ~ x + id) %>%
  step_nest(-c(x, z))

recipe2 %>%
  recipes::prep() %>%
  recipes::bake(NULL)
```

---

tidy.nested_model_fit *Turn a nested model into a tidy tibble*

Description

Use broom functions on fitted nested models.

`tidy.nested_model_fit()` summarises components of each model within a nested model fit, indicating which nested data frame each row corresponds to.

`glance.nested_model_fit()` summarises a nested model, returning a `tibble::tibble()` with 1 row.

`glance_nested()` summarises each model within a nested model fit, returning a `tibble::tibble()` with the same number of rows as the number of inner models.

Usage

```
## S3 method for class 'nested_model_fit'
tidy(x, ...)

## S3 method for class 'nested_model_fit'
glance(x, ...)

glance_nested(x, ...)
```

Arguments

`x`	A `nested_model_fit` object produced by `fit.nested_model()`.
`...`	Additional arguments passed into their respective functions. (e.g. for `tidy.nested_model_fit()`, `parsnip::tidy.model_fit()`).

Details

generics::glance() states that glance() methods should always return 1 row outputs for non-empty inputs. The 'nestedmodels' package is no exception: glance() methods will combine rows to produce a result with a single row. Specifically:

- If a column contains 1 unique value, that value is used.
- If a column is numeric, the mean will be calculated.
- Otherwise, the results will be combined into a list.

Value

A tibble::tibble(). With glance.nested_model_fit(), the tibble will have 1 row.

See Also

generics::tidy() generics::glance()

Examples

```
if (require("broom")) {
  data <- dplyr::filter(example_nested_data, id %in% 1:5)

  model <- parsnip::linear_reg() %>%
    parsnip::set_engine("lm") %>%
    nested()

  fit <- fit(
    model, z ~ x + y + a + b,
    dplyr::group_by(data, id)
  )

  tidy(fit)
  glance(fit)
  glance_nested(fit)
}
```

Index