

Package ‘rsample’

September 23, 2020

Title General Resampling Infrastructure

Version 0.0.8

Maintainer Max Kuhn <max@rstudio.com>

Description Classes and functions to create and summarize different types of resampling objects (e.g. bootstrap, cross-validation).

Imports dplyr (>= 0.8.5), purrr, tibble, rlang (>= 0.4.0), methods, generics, utils, tidyselect, furrr, tidyr, vctrs (>= 0.3.0), modeldata, slider (>= 0.1.5), ellipsis

Depends R (>= 3.2)

Suggests ggplot2, testthat, rmarkdown, knitr, AmesHousing, recipes (>= 0.1.4), broom, xml2, covr

URL <https://rsample.tidymodels.org>,
<https://github.com/tidymodels/rsample>

BugReports <https://github.com/tidymodels/rsample/issues>

License GPL-2

Encoding UTF-8

VignetteBuilder knitr

LazyData true

RoxygenNote 7.1.1.9000

NeedsCompilation no

Author Max Kuhn [aut, cre],
Fanny Chow [aut],
Hadley Wickham [aut],
RStudio [cph]

Repository CRAN

Date/Publication 2020-09-23 17:20:03 UTC

R topics documented:

add_resample_id	2
apparent	3
as.data.frame.rsplit	4
attrition	5
bootstraps	5
complement	6
drinks	7
form_pred	8
gather.rset	9
group_vfold_cv	10
initial_split	11
int_pctl	12
labels.rset	14
labels.rsplit	15
loo_cv	15
make_strata	16
manual_rset	17
mc_cv	18
nested_cv	19
populate	21
rolling_origin	21
rsample	23
rsample-dplyr	24
rsample2caret	26
slide-resampling	26
tidy.rsplit	30
two_class_dat	32
validation_split	32
vfold_cv	33
Index	36

add_resample_id	<i>Augment a data set with resampling identifiers</i>
-----------------	---

Description

For a data set, `add_resample_id()` will add at least one new column that identifies which resample that the data came from. In most cases, a single column is added but for some resampling methods, two or more are added.

Usage

```
add_resample_id(.data, split, dots = FALSE)
```

Arguments

<code>.data</code>	A data frame
<code>split</code>	A single rset object.
<code>dots</code>	A single logical: should the id columns be prefixed with a "." to avoid name conflicts with <code>.data</code> ?

Value

An updated data frame.

See Also

`labels.rsplit`

Examples

```
library(dplyr)

set.seed(363)
car_folds <- vfold_cv(mtcars, repeats = 3)

analysis(car_folds$splits[[1]]) %>%
  add_resample_id(car_folds$splits[[1]]) %>%
  head()

car_bt <- bootstraps(mtcars)

analysis(car_bt$splits[[1]]) %>%
  add_resample_id(car_bt$splits[[1]]) %>%
  head()
```

apparent

Sampling for the Apparent Error Rate

Description

When building a model on a data set and re-predicting the same data, the performance estimate from those predictions is often called the "apparent" performance of the model. This estimate can be wildly optimistic. "Apparent sampling" here means that the analysis and assessment samples are the same. These resamples are sometimes used in the analysis of bootstrap samples and should otherwise be avoided like old sushi.

Usage

```
apparent(data, ...)
```

Arguments

data	A data frame.
...	Not currently used.

Value

A tibble with a single row and classes apparent, rset, tbl_df, tbl, and data.frame. The results include a column for the data split objects and one column called id that has a character string with the resample identifier.

Examples

```
apparent(mtcars)
```

```
as.data.frame.rsplit Convert an rsplit object to a data frame
```

Description

The analysis or assessment code can be returned as a data frame (as dictated by the data argument) using as.data.frame.rsplit. analysis and assessment are shortcuts.

Usage

```
## S3 method for class 'rsplit'
as.data.frame(x, row.names = NULL, optional = FALSE, data = "analysis", ...)

analysis(x, ...)

assessment(x, ...)
```

Arguments

x	An rsplit object.
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	A logical: should the column names of the data be checked for legality?
data	Either "analysis" or "assessment" to specify which data are returned.
...	Additional arguments to be passed to or from methods. Not currently used.

Examples

```
library(dplyr)
set.seed(104)
folds <- vfold_cv(mtcars)

model_data_1 <- folds$splits[[1]] %>% analysis()
holdout_data_1 <- folds$splits[[1]] %>% assessment()
```

attrition	<i>Job Attrition</i>
-----------	----------------------

Description

Job Attrition

Details

These data are from the IBM Watson Analytics Lab. The website describes the data with “Uncover the factors that lead to employee attrition and explore important questions such as ‘show me a breakdown of distance from home by job role and attrition’ or ‘compare average monthly income by education and attrition’. This is a fictional data set created by IBM data scientists.”. There are 1470 rows.

These data are now in the `modeldata` package.

bootstraps	<i>Bootstrap Sampling</i>
------------	---------------------------

Description

A bootstrap sample is a sample that is the same size as the original data set that is made using replacement. This results in analysis samples that have multiple replicates of some of the original rows of the data. The assessment set is defined as the rows of the original data that were not included in the bootstrap sample. This is often referred to as the "out-of-bag" (OOB) sample.

Usage

```
bootstraps(data, times = 25, strata = NULL, breaks = 4, apparent = FALSE, ...)
```

Arguments

<code>data</code>	A data frame.
<code>times</code>	The number of bootstrap samples.
<code>strata</code>	A variable that is used to conduct stratified sampling. When not <code>NULL</code> , each bootstrap sample is created within the stratification variable. This could be a single character value or a variable name that corresponds to a variable that exists in the data frame.
<code>breaks</code>	A single number giving the number of bins desired to stratify a numeric stratification variable.
<code>apparent</code>	A logical. Should an extra resample be added where the analysis and holdout subset are the entire data set. This is required for some estimators used by the summary function that require the apparent error rate.
<code>...</code>	Not currently used.

Details

The argument `apparent` enables the option of an additional "resample" where the analysis and assessment data sets are the same as the original data set. This can be required for some types of analysis of the bootstrap results. The `strata` argument is based on a similar argument in the random forest package where the bootstrap samples are conducted *within the stratification variable*. This can help ensure that the number of data points in the bootstrap sample is equivalent to the proportions in the original data set. (Strata below 10% of the total are pooled together.)

Value

An tibble with classes `bootstraps`, `rset`, `tbl_df`, `tbl`, and `data.frame`. The results include a column for the data split objects and a column called `id` that has a character string with the resample identifier.

Examples

```
bootstraps(mtcars, times = 2)
bootstraps(mtcars, times = 2, apparent = TRUE)

library(purrr)
data(wa_churn, package = "modeldata")

set.seed(13)
resample1 <- bootstraps(wa_churn, times = 3)
map_dbl(resample1$splits,
  function(x) {
    dat <- as.data.frame(x)$churn
    mean(dat == "Yes")
  })

set.seed(13)
resample2 <- bootstraps(wa_churn, strata = "churn", times = 3)
map_dbl(resample2$splits,
  function(x) {
    dat <- as.data.frame(x)$churn
    mean(dat == "Yes")
  })

set.seed(13)
resample3 <- bootstraps(wa_churn, strata = "tenure", breaks = 6, times = 3)
map_dbl(resample3$splits,
  function(x) {
    dat <- as.data.frame(x)$churn
    mean(dat == "Yes")
  })
```

Description

Given an `rsplit` object, `complement` will determine which of the data rows are contained in the assessment set. To save space, many of the `rset` objects will not contain indices for the assessment split.

Usage

```
complement(x, ...)
```

Arguments

<code>x</code>	An <code>rsplit</code> object
<code>...</code>	Not currently used

Value

A integer vector.

See Also

[populate\(\)](#)

Examples

```
set.seed(28432)
fold_rs <- vfold_cv(mtcars)
head(fold_rs$splits[[1]]$in_id)
fold_rs$splits[[1]]$out_id
complement(fold_rs$splits[[1]])
```

drinks

Sample Time Series Data

Description

Sample Time Series Data

Details

Drink sales. The exact name of the series from FRED is: "Merchant Wholesalers, Except Manufacturers' Sales Branches and Offices Sales: Nondurable Goods: Beer, Wine, and Distilled Alcoholic Beverages Sales"

These data are now in the `modeldata` package.

`form_pred`*Extract Predictor Names from Formula or Terms*

Description

`all.vars` returns all variables used in a formula. This function only returns the variables explicitly used on the right-hand side (i.e., it will not resolve dots unless the object is terms with a data set specified).

Usage

```
form_pred(object, ...)
```

Arguments

<code>object</code>	A model formula or <code>stats::terms()</code> object.
<code>...</code>	Arguments to pass to <code>all.vars()</code>

Value

A character vector of names

Examples

```
form_pred(y ~ x + z)
form_pred(terms(y ~ x + z))

form_pred(y ~ x + log(z))
form_pred(log(y) ~ x + z)

form_pred(y1 + y2 ~ x + z)
form_pred(log(y1) + y2 ~ x + z)

# will fail:
# form_pred(y ~ .)

form_pred(terms(mpg ~ (.)^2, data = mtcars))
form_pred(terms( ~ (.)^2, data = mtcars))
```

`gather.rset`*Gather an rset Object*

Description

This method uses `gather` on an `rset` object to stack all of the non-ID or split columns in the data and is useful for stacking model evaluation statistics. The resulting data frame has a column based on the column names of data and another for the values.

Usage

```
## S3 method for class 'rset'
gather(
  data,
  key = NULL,
  value = NULL,
  ...,
  na.rm = TRUE,
  convert = FALSE,
  factor_key = TRUE
)
```

Arguments

<code>data</code>	An <code>rset</code> object.
<code>key</code> , <code>value</code> , ...	Not specified in this method and will be ignored. Note that this means that selectors are ignored if they are passed to the function.
<code>na.rm</code>	If <code>TRUE</code> , will remove rows from output where the value column is NA.
<code>convert</code>	If <code>TRUE</code> will automatically run <code>type.convert()</code> on the key column. This is useful if the column names are actually numeric, integer, or logical.
<code>factor_key</code>	If <code>FALSE</code> , the default, the key values will be stored as a character vector. If <code>TRUE</code> , will be stored as a factor, which preserves the original ordering of the columns.

Value

A data frame with the ID columns, a column called `model` (with the previous column names), and a column called `statistic` (with the values).

Examples

```
library(rsample)
cv_obj <- vfold_cv(mtcars, v = 10)
cv_obj$lm_rmse <- rnorm(10, mean = 2)
cv_obj$nnet_rmse <- rnorm(10, mean = 1)
gather(cv_obj)
```

group_vfold_cv	<i>Group V-Fold Cross-Validation</i>
----------------	--------------------------------------

Description

Group V-fold cross-validation creates splits of the data based on some grouping variable (which may have more than a single row associated with it). The function can create as many splits as there are unique values of the grouping variable or it can create a smaller set of splits where more than one value is left out at a time.

Usage

```
group_vfold_cv(data, group = NULL, v = NULL, ...)
```

Arguments

data	A data frame.
group	This could be a single character value or a variable name that corresponds to a variable that exists in the data frame.
v	The number of partitions of the data set. If left NULL, v will be set to the number of unique values in the group.
...	Not currently used.

Value

A tibble with classes `group_vfold_cv`, `rset`, `tbl_df`, `tbl`, and `data.frame`. The results include a column for the data split objects and an identification variable.

Examples

```
set.seed(3527)
test_data <- data.frame(id = sort(sample(1:20, size = 80, replace = TRUE)))
test_data$dat <- runif(nrow(test_data))

set.seed(5144)
split_by_id <- group_vfold_cv(test_data, group = "id")

get_id_left_out <- function(x)
  unique(assessment(x)$id)

library(purrr)
table(map_int(split_by_id$splits, get_id_left_out))

set.seed(5144)
split_by_some_id <- group_vfold_cv(test_data, group = "id", v = 7)
held_out <- map(split_by_some_id$splits, get_id_left_out)
table(unlist(held_out))
# number held out per resample:
map_int(held_out, length)
```

initial_split	<i>Simple Training/Test Set Splitting</i>
---------------	---

Description

`initial_split` creates a single binary split of the data into a training set and testing set. `initial_time_split` does the same, but takes the *first* prop samples for training, instead of a random selection. `training` and `testing` are used to extract the resulting data.

Usage

```
initial_split(data, prop = 3/4, strata = NULL, breaks = 4, ...)
```

```
initial_time_split(data, prop = 3/4, lag = 0, ...)
```

```
training(x)
```

```
testing(x)
```

Arguments

<code>data</code>	A data frame.
<code>prop</code>	The proportion of data to be retained for modeling/analysis.
<code>strata</code>	A variable that is used to conduct stratified sampling to create the resamples. This could be a single character value or a variable name that corresponds to a variable that exists in the data frame.
<code>breaks</code>	A single number giving the number of bins desired to stratify a numeric stratification variable.
<code>...</code>	Not currently used.
<code>lag</code>	A value to include a lag between the assessment and analysis set. This is useful if lagged predictors will be used during training and testing.
<code>x</code>	An <code>rsplit</code> object produced by <code>initial_split</code>

Details

The `strata` argument causes the random sampling to be conducted *within the stratification variable*. This can help ensure that the number of data points in the training data is equivalent to the proportions in the original data set. (Strata below 10% of the total are pooled together.)

Value

An `rsplit` object that can be used with the `training` and `testing` functions to extract the data in each split.

Examples

```

set.seed(1353)
car_split <- initial_split(mtcars)
train_data <- training(car_split)
test_data <- testing(car_split)

data(drinks, package = "modeldata")
drinks_split <- initial_time_split(drinks)
train_data <- training(drinks_split)
test_data <- testing(drinks_split)
c(max(train_data$date), min(test_data$date)) # no lag

# With 12 period lag
drinks_lag_split <- initial_time_split(drinks, lag = 12)
train_data <- training(drinks_lag_split)
test_data <- testing(drinks_lag_split)
c(max(train_data$date), min(test_data$date)) # 12 period lag

```

int_pctl

*Bootstrap confidence intervals***Description**

Calculate bootstrap confidence intervals using various methods.

Usage

```

int_pctl(.data, statistics, alpha = 0.05)

int_t(.data, statistics, alpha = 0.05)

int_bca(.data, statistics, alpha = 0.05, .fn, ...)

```

Arguments

.data	A data frame containing the bootstrap resamples created using <code>bootstraps()</code> . For t- and BCa-intervals, the <code>apparent</code> argument should be set to <code>TRUE</code> . Even if the <code>apparent</code> argument is set to <code>TRUE</code> for the percentile method, the <code>apparent</code> data is never used in calculating the percentile confidence interval.
statistics	An unquoted column name or <code>dplyr</code> selector that identifies a single column in the data set that contains the individual bootstrap estimates. This can be a list column of tidy tibbles (that contains columns <code>term</code> and <code>estimate</code>) or a simple numeric column. For t-intervals, a standard tidy column (usually called <code>std.err</code>) is required. See the examples below.
alpha	Level of significance
.fn	A function to calculate statistic of interest. The function should take an <code>rsplit</code> as the first argument and the <code>...</code> are required.
...	Arguments to pass to <code>.fn</code> .

Details

Percentile intervals are the standard method of obtaining confidence intervals but require thousands of resamples to be accurate. T-intervals may need fewer resamples but require a corresponding variance estimate. Bias-corrected and accelerated intervals require the original function that was used to create the statistics of interest and are computationally taxing.

Value

Each function returns a tibble with columns `.lower`, `.estimate`, `.upper`, `.alpha`, `.method`, and `term`. `.method` is the type of interval (eg. "percentile", "student-t", or "BCa"). `term` is the name of the estimate. Note the `.estimate` returned from `int_pctl()` is the mean of the estimates from the bootstrap resamples and not the estimate from the apparent model.

References

Davison, A., & Hinkley, D. (1997). *Bootstrap Methods and their Application*. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511802843

<https://rsample.tidymodels.org/articles/Applications/Intervals.html>

Examples

```
library(broom)
library(dplyr)
library(purrr)
library(tibble)

lm_est <- function(split, ...) {
  lm(mpg ~ disp + hp, data = analysis(split)) %>%
    tidy()
}

set.seed(52156)
car_rs <-
  bootstraps(mtcars, 500, apparent = TRUE) %>%
  mutate(results = map(splits, lm_est))

int_pctl(car_rs, results)
int_t(car_rs, results)
int_bca(car_rs, results, .fn = lm_est)

# putting results into a tidy format
rank_corr <- function(split) {
  dat <- analysis(split)
  tibble(
    term = "corr",
    estimate = cor(dat$sqft, dat$price, method = "spearman"),
    # don't know the analytical std.err so no t-intervals
    std.err = NA_real_
  )
}
```

```
set.seed(69325)
data(Sacramento, package = "modeldata")
bootstraps(Sacramento, 1000, apparent = TRUE) %>%
  mutate(correlations = map(splits, rank_corr)) %>%
  int_pctl(correlations)
```

labels.rset

Find Labels from rset Object

Description

Produce a vector of resampling labels (e.g. "Fold1") from an rset object. Currently, nested_cv is not supported.

Usage

```
## S3 method for class 'rset'
labels(object, make_factor = FALSE, ...)

## S3 method for class 'vfold_cv'
labels(object, make_factor = FALSE, ...)
```

Arguments

object	An rset object
make_factor	A logical for whether the results should be a character or a factor.
...	Not currently used.

Value

A single character or factor vector.

Examples

```
labels(vfold_cv(mtcars))
```

labels.rsplitt	<i>Find Labels from rsplitt Object</i>
----------------	--

Description

Produce a tibble of identification variables so that single splits can be linked to a particular resample.

Usage

```
## S3 method for class 'rsplitt'  
labels(object, ...)
```

Arguments

object	An rsplitt object
...	Not currently used.

Value

A tibble.

See Also

`add_resample_id`

Examples

```
cv_splits <- vfold_cv(mtcars)  
labels(cv_splits$splits[[1]])
```

loo_cv	<i>Leave-One-Out Cross-Validation</i>
--------	---------------------------------------

Description

Leave-one-out (LOO) cross-validation uses one data point in the original set as the assessment data and all other data points as the analysis set. A LOO resampling set has as many resamples as rows in the original data set.

Usage

```
loo_cv(data, ...)
```

Arguments

data	A data frame.
...	Not currently used.

Value

An tibble with classes `loo_cv`, `rset`, `tbl_df`, `tbl`, and `data.frame`. The results include a column for the data split objects and one column called `id` that has a character string with the resample identifier.

Examples

```
loo_cv(mtcars)
```

make_strata

Create or Modify Stratification Variables

Description

For stratified resampling, this function can create strata from numeric data and also make non-numeric data more conducive to be used for stratification.

Usage

```
make_strata(x, breaks = 4, nunique = 5, pool = 0.1, depth = 20)
```

Arguments

<code>x</code>	An input vector.
<code>breaks</code>	A single number giving the number of bins desired to stratify a numeric stratification variable.
<code>nunique</code>	An integer for the number of unique value threshold in the algorithm.
<code>pool</code>	A proportion of data used to determine if a particular group is too small and should be pooled into another group.
<code>depth</code>	An integer that is used to determine the best number of percentiles that should be used. The number of bins are based on $\min(5, \text{floor}(n / \text{depth}))$ where $n = \text{length}(x)$. If <code>x</code> is numeric, there must be at least 40 rows in the data set (when <code>depth = 20</code>) to conduct stratified sampling.

Details

For numeric data, if the number of unique levels is less than `nunique`, the data are treated as categorical data.

For categorical inputs, the function will find levels of `x` than occur in the data with percentage less than `pool`. The values from these groups will be randomly assigned to the remaining strata (as will data points that have missing values in `x`).

For numeric data with more unique values than `nunique`, the data will be converted to being categorical based on percentiles of the data. The percentile groups will have no more than 20 percent of the data in each group. Again, missing values in `x` are randomly assigned to groups.

Value

A factor vector.

Examples

```
set.seed(61)
x1 <- rpois(100, lambda = 5)
table(x1)
table(make_strata(x1))

set.seed(554)
x2 <- rpois(100, lambda = 1)
table(x2)
table(make_strata(x2))

# small groups are randomly assigned
x3 <- factor(x2)
table(x3)
table(make_strata(x3))

# `oilType` data from `caret`
x4 <- rep(LETTERS[1:7], c(37, 26, 3, 7, 11, 10, 2))
table(x4)
table(make_strata(x4))
table(make_strata(x4, pool = 0.1))
table(make_strata(x4, pool = 0.0))

# not enough data to stratify
x5 <- rnorm(20)
table(make_strata(x5))

set.seed(483)
x6 <- rnorm(200)
quantile(x6, probs = (0:10)/10)
table(make_strata(x6, breaks = 10))
```

manual_rset

Manual resampling

Description

`manual_rset()` is used for constructing the most minimal rset possible. It can be useful when you have custom `rsplit` objects built from `make_splits()`, or when you want to create a new rset from splits contained within an existing rset.

Usage

```
manual_rset(splits, ids)
```

Arguments

splits	A list of "rsplit" objects. It is easiest to create these using <code>make_splits()</code> .
ids	A character vector of ids. The length of ids must be the same as the length of splits.

Examples

```
df <- data.frame(x = c(1, 2, 3, 4, 5, 6))

# Create an rset from custom indices
indices <- list(
  list(analysis = c(1L, 2L), assessment = 3L),
  list(analysis = c(4L, 5L), assessment = 6L)
)

splits <- lapply(indices, make_splits, data = df)

manual_rset(splits, c("Split 1", "Split 2"))

# You can also use this to create an rset from a subset of an
# existing rset
resamples <- vfold_cv(mtcars)
best_split <- resamples[5,]
manual_rset(best_split$splits, best_split$id)
```

mc_cv

*Monte Carlo Cross-Validation***Description**

One resample of Monte Carlo cross-validation takes a random sample (without replacement) of the original data set to be used for analysis. All other data points are added to the assessment set.

Usage

```
mc_cv(data, prop = 3/4, times = 25, strata = NULL, breaks = 4, ...)
```

Arguments

data	A data frame.
prop	The proportion of data to be retained for modeling/analysis.
times	The number of times to repeat the sampling.
strata	A variable that is used to conduct stratified sampling to create the resamples. This could be a single character value or a variable name that corresponds to a variable that exists in the data frame.
breaks	A single number giving the number of bins desired to stratify a numeric stratification variable.
...	Not currently used.

Details

The `strata` argument causes the random sampling to be conducted *within the stratification variable*. This can help ensure that the number of data points in the analysis data is equivalent to the proportions in the original data set. (Strata below 10% of the total are pooled together.)

Value

An tibble with classes `mc_cv`, `rset`, `tbl_df`, `tbl`, and `data.frame`. The results include a column for the data split objects and a column called `id` that has a character string with the resample identifier.

Examples

```
mc_cv(mtcars, times = 2)
mc_cv(mtcars, prop = .5, times = 2)

library(purrr)
data(wa_churn, package = "modeldata")

set.seed(13)
resample1 <- mc_cv(wa_churn, times = 3, prop = .5)
map_dbl(resample1$splits,
  function(x) {
    dat <- as.data.frame(x)$churn
    mean(dat == "Yes")
  })

set.seed(13)
resample2 <- mc_cv(wa_churn, strata = "churn", times = 3, prop = .5)
map_dbl(resample2$splits,
  function(x) {
    dat <- as.data.frame(x)$churn
    mean(dat == "Yes")
  })

set.seed(13)
resample3 <- mc_cv(wa_churn, strata = "tenure", breaks = 6, times = 3, prop = .5)
map_dbl(resample3$splits,
  function(x) {
    dat <- as.data.frame(x)$churn
    mean(dat == "Yes")
  })
```

nested_cv

Nested or Double Resampling

Description

`nested_cv` can be used to take the results of one resampling procedure and conduct further resamples within each split. Any type of resampling used in `rsample` can be used.

Usage

```
nested_cv(data, outside, inside)
```

Arguments

data	A data frame.
outside	The initial resampling specification. This can be an already created object or an expression of a new object (see the examples below). If the latter is used, the data argument does not need to be specified and, if it is given, will be ignored.
inside	An expression for the type of resampling to be conducted within the initial procedure.

Details

It is a bad idea to use bootstrapping as the outer resampling procedure (see the example below)

Value

An tibble with nested_cv class and any other classes that outer resampling process normally contains. The results include a column for the outer data split objects, one or more id columns, and a column of nested tibbles called inner_resamples with the additional resamples.

Examples

```
## Using expressions for the resampling procedures:
nested_cv(mtcars, outside = vfold_cv(v = 3), inside = bootstraps(times = 5))

## Using an existing object:
folds <- vfold_cv(mtcars)
nested_cv(mtcars, folds, inside = bootstraps(times = 5))

## The dangers of outer bootstraps:
set.seed(2222)
bad_idea <- nested_cv(mtcars,
                      outside = bootstraps(times = 5),
                      inside = vfold_cv(v = 3))

first_outer_split <- bad_idea$splits[[1]]
outer_analysis <- as.data.frame(first_outer_split)
sum(grepl("Volvo 142E", rownames(outer_analysis)))

## For the 3-fold CV used inside of each bootstrap, how are the replicated
## `Volvo 142E` data partitioned?
first_inner_split <- bad_idea$inner_resamples[[1]]$splits[[1]]
inner_analysis <- as.data.frame(first_inner_split)
inner_assess <- as.data.frame(first_inner_split, data = "assessment")

sum(grepl("Volvo 142E", rownames(inner_analysis)))
sum(grepl("Volvo 142E", rownames(inner_assess)))
```

populate	<i>Add Assessment Indices</i>
----------	-------------------------------

Description

Many `rsplit` and `rset` objects do not contain indicators for the assessment samples. `populate()` can be used to fill the slot for the appropriate indices.

Usage

```
populate(x, ...)
```

Arguments

<code>x</code>	A <code>rsplit</code> and <code>rset</code> object.
<code>...</code>	Not currently used

Value

An object of the same kind with the integer indices.

Examples

```
set.seed(28432)
fold_rs <- vfold_cv(mtcars)

fold_rs$splits[[1]]$out_id
complement(fold_rs$splits[[1]])

populate(fold_rs$splits[[1]])$out_id

fold_rs_all <- populate(fold_rs)
fold_rs_all$splits[[1]]$out_id
```

rolling_origin	<i>Rolling Origin Forecast Resampling</i>
----------------	---

Description

This resampling method is useful when the data set has a strong time component. The resamples are not random and contain data points that are consecutive values. The function assumes that the original data set are sorted in time order.

Usage

```
rolling_origin(
  data,
  initial = 5,
  assess = 1,
  cumulative = TRUE,
  skip = 0,
  lag = 0,
  ...
)
```

Arguments

data	A data frame.
initial	The number of samples used for analysis/modeling in the initial resample.
assess	The number of samples used for each assessment resample.
cumulative	A logical. Should the analysis resample grow beyond the size specified by <code>initial</code> at each resample?.
skip	A integer indicating how many (if any) <i>additional</i> resamples to skip to thin the total amount of data points in the analysis resample. See the example below.
lag	A value to include a lag between the assessment and analysis set. This is useful if lagged predictors will be used during training and testing.
...	Not currently used.

Details

The main options, `initial` and `assess`, control the number of data points from the original data that are in the analysis and assessment set, respectively. When `cumulative = TRUE`, the analysis set will grow as resampling continues while the assessment set size will always remain static. `skip` enables the function to not use every data point in the resamples. When `skip = 0`, the resampling data sets will increment by one position. Suppose that the rows of a data set are consecutive days. Using `skip = 6` will make the analysis data set to operate on *weeks* instead of days. The assessment set size is not affected by this option.

Value

An tibble with classes `rolling_origin`, `rset`, `tbl_df`, `tbl`, and `data.frame`. The results include a column for the data split objects and a column called `id` that has a character string with the resample identifier.

See Also

[sliding_window\(\)](#), [sliding_index\(\)](#), and [sliding_period\(\)](#) for additional time based resampling functions.

Examples

```

set.seed(1131)
ex_data <- data.frame(row = 1:20, some_var = rnorm(20))
dim(rolling_origin(ex_data))
dim(rolling_origin(ex_data, skip = 2))
dim(rolling_origin(ex_data, skip = 2, cumulative = FALSE))

# You can also roll over calendar periods by first nesting by that period,
# which is especially useful for irregular series where a fixed window
# is not useful. This example slides over 5 years at a time.
library(dplyr)
library(tidyr)
data(drinks, package = "modeldata")

drinks_annual <- drinks %>%
  mutate(year = as.POSIXlt(date)$year + 1900) %>%
  nest(-year)

multi_year_roll <- rolling_origin(drinks_annual, cumulative = FALSE)

analysis(multi_year_roll$splits[[1]])
assessment(multi_year_roll$splits[[1]])

```

rsample

rsample: General Resampling Infrastructure for R

Description

rsample has functions to create variations of a data set that can be used to evaluate models or to estimate the sampling distribution of some statistic.

Terminology

- A **resample** is the result of a two-way split of a data set. For example, when bootstrapping, one part of the resample is a sample with replacement of the original data. The other part of the split contains the instances that were not contained in the bootstrap sample. The data structure `rsplit` is used to store a single resample.
- When the data are split in two, the portion that is used to estimate the model or calculate the statistic is called the **analysis** set here. In machine learning this is sometimes called the "training set" but this would be poorly named since it might conflict with any initial split of the original data.
- Conversely, the other data in the split are called the **assessment** data. In bootstrapping, these data are often called the "out-of-bag" samples.
- A collection of resamples is contained in an `rset` object.

Basic Functions

The main resampling functions are: `vfold_cv()`, `bootstraps()`, `mc_cv()`, `rolling_origin()`, and `nested_cv()`.

rsample-dplyr

Compatibility with dplyr

Description

rsample should be fully compatible with dplyr 1.0.0.

With older versions of dplyr, there is partial support for the following verbs: `mutate()`, `arrange()`, `filter()`, `rename()`, `select()`, and `slice()`. We strongly recommend updating to dplyr 1.0.0 if possible to get more complete integration with dplyr.

Version Specific Behavior

rsample performs somewhat differently depending on whether you have dplyr \geq 1.0.0 (new) or dplyr $<$ 1.0.0 (old). Additionally, version 0.0.7 of rsample (new) introduced some changes to how rsample objects work with dplyr, even on old dplyr. Most of these changes influence the return value of a dplyr verb and determine whether it will be a tibble or an rsample rset subclass.

The table below attempts to capture most of these changes. These examples are not exhaustive and may not capture some edge-cases.

Joins:

The following affect all of the dplyr joins, such as `left_join()`, `right_join()`, `full_join()`, and `inner_join()`.

Joins that alter the rows of the original rset object:

operation	old rsample + old dplyr	new rsample + old dplyr	new rsample + new dplyr
<code>join(rset, tbl)</code>	error	error	tibble

The idea here is that, if there are less rows in the result, the result should not be an rset object. For example, you can't have a 10-fold CV object without 10 rows.

Joins that keep the rows of the original rset object:

operation	old rsample + old dplyr	new rsample + old dplyr	new rsample + new dplyr
<code>join(rset, tbl)</code>	error	error	rset

As with the logic above, if the original rset object (defined by the split column and the id column(s)) is left intact, the results should be an rset.

Row Subsetting:

As mentioned above, this should result in a tibble if any rows are removed or added. Simply reordering rows still results in a valid rset with new rsample.

Cases where rows are removed or added:

operation	old rsample + old dplyr	new rsample + old dplyr	new rsample + new dplyr
rset[ind,]	tibble	tibble	tibble
slice(rset)	rset	tibble	tibble
filter(rset)	rset	tibble	tibble

Cases where all rows are kept, but are possibly reordered:

operation	old rsample + old dplyr	new rsample + old dplyr	new rsample + new dplyr
rset[ind,]	tibble	rset	rset
slice(rset)	rset	rset	rset
filter(rset)	rset	rset	rset
arrange(rset)	rset	rset	rset

Column Subsetting:

When the `splits` column or any id columns are dropped or renamed, the result should no longer be considered a valid rset.

Cases when the required columns are removed or renamed:

operation	old rsample + old dplyr	new rsample + old dplyr	new rsample + new dplyr
rset[, ind]	tibble	tibble	tibble
select(rset)	rset	tibble	tibble
rename(rset)	tibble	tibble	tibble

Cases when no required columns are affected:

operation	old rsample + old dplyr	new rsample + old dplyr	new rsample + new dplyr
rset[, ind]	tibble	rset	rset
select(rset)	rset	rset	rset
rename(rset)	rset	rset	rset

Other Column Operations:

Cases when the required columns are altered:

operation	old rsample + old dplyr	new rsample + old dplyr	new rsample + new dplyr
mutate(rset)	rset	tibble	tibble

Cases when no required columns are affected:

operation	old rsample + old dplyr	new rsample + old dplyr	new rsample + new dplyr
mutate(rset)	rset	rset	rset

 rsample2caret

Convert Resampling Objects to Other Formats

Description

These functions can convert resampling objects between **rsample** and **caret**.

Usage

```
rsample2caret(object, data = c("analysis", "assessment"))
```

```
caret2rsample(ctrl, data = NULL)
```

Arguments

object	An rset object. Currently, nested_cv is not supported.
data	The data that was originally used to produce the ctrl object.
ctrl	An object produced by trainControl that has had the index and indexOut elements populated by integers. One method of getting this is to extract the control objects from an object produced by train.

Value

rsample2caret returns a list that mimics the index and indexOut elements of a trainControl object. caret2rsample returns an rset object of the appropriate class.

 slide-resampling

Time-based Resampling

Description

These resampling functions are focused on various forms of *time series* resampling.

- `sliding_window()` uses the row number when computing the resampling indices. It is independent of any time index, but is useful with completely regular series.
- `sliding_index()` computes resampling indices relative to the index column. This is often a Date or POSIXct column, but doesn't have to be. This is useful when resampling irregular series, or for using irregular lookback periods such as `lookback = lubridate::years(1)` with daily data (where the number of days in a year may vary).
- `sliding_period()` first breaks up the index into more granular groups based on period, and then uses that to construct the resampling indices. This is extremely useful for constructing rolling monthly or yearly windows from daily data.

Usage

```
sliding_window(  
  data,  
  ...,  
  lookback = 0L,  
  assess_start = 1L,  
  assess_stop = 1L,  
  complete = TRUE,  
  step = 1L,  
  skip = 0L  
)
```

```
sliding_index(  
  data,  
  index,  
  ...,  
  lookback = 0L,  
  assess_start = 1L,  
  assess_stop = 1L,  
  complete = TRUE,  
  step = 1L,  
  skip = 0L  
)
```

```
sliding_period(  
  data,  
  index,  
  period,  
  ...,  
  lookback = 0L,  
  assess_start = 1L,  
  assess_stop = 1L,  
  complete = TRUE,  
  step = 1L,  
  skip = 0L,  
  every = 1L,  
  origin = NULL  
)
```

Arguments

data	A data frame.
...	These dots are for future extensions and must be empty.
lookback	The number of elements to look back from the current element when computing the resampling indices of the analysis set. The current row is always included in the analysis set. <ul style="list-style-type: none">• For <code>sliding_window()</code>, a single integer defining the number of rows to look back from the current row.

- For `sliding_index()`, a single object that will be subtracted from the index as `index -lookback` to define the boundary of where to start searching for rows to include in the current resample. This is often an integer value corresponding to the number of days to look back, or a lubridate Period object.
- For `sliding_period()`, a single integer defining the number of groups to look back from the current group, where the groups were defined from breaking up the index according to the period.

In all cases, `Inf` is also allowed to force an expanding window.

`assess_start`, `assess_stop`

This combination of arguments determines how far into the future to look when constructing the assessment set. Together they construct a range of `[index + assess_start, index + assess_stop]` to search for rows to include in the assessment set.

Generally, `assess_start` will always be 1 to indicate that the first value to potentially include in the assessment set should start one element after the current row, but it can be increased to a larger value to create "gaps" between the analysis and assessment set if you are worried about high levels of correlation in short term forecasting.

- For `sliding_window()`, these are both single integers defining the number of rows to look forward from the current row.
- For `sliding_index()`, these are single objects that will be added to the index to compute the range to search for rows to include in the assessment set. This is often an integer value corresponding to the number of days to look forward, or a lubridate Period object.
- For `sliding_period()`, these are both single integers defining the number of groups to look forward from the current group, where the groups were defined from breaking up the index according to the period.

`complete`

A single logical. When using `lookback` to compute the analysis sets, should only complete windows be considered? If set to `FALSE`, partial windows will be used until it is possible to create a complete window (based on `lookback`). This is a way to use an expanding window up to a certain point, and then switch to a sliding window.

`step`

A single positive integer. After computing the resampling indices, `step` is used to thin out the results by selecting every `step`-th result by subsetting the indices with `seq(1L, n_indices, by = step)`. `step` is applied after `skip`. Note that `step` is independent of any time index used.

`skip`

A single positive integer, or zero. After computing the resampling indices, the first `skip` results will be dropped by subsetting the indices with `seq(skip + 1L, n_indices)`. This can be especially useful when combined with `lookback = Inf`, which creates an expanding window starting from the first row. By skipping forward, you can drop the first few windows that have very few data points. `skip` is applied before `step`. Note that `skip` is independent of any time index used.

`index`

The index to compute resampling indices relative to, specified as a bare column name. This must be an existing column in data.

- For `sliding_index()`, this is commonly a date vector, but is not required.

- For `sliding_period()`, it is required that this is a Date or POSIXct vector. The index must be an *increasing* vector, but duplicate values are allowed. Additionally, the index cannot contain any missing values.

period	The period to group the index by. This is specified as a single string, such as "year" or "month". See the <code>.period</code> argument of <code>slider::slide_index()</code> for the full list of options and further explanation.
every	A single positive integer. The number of periods to group together. For example, if the period was set to "year" with an every value of 2, then the years 1970 and 1971 would be placed in the same group.
origin	The reference date time value. The default when left as NULL is the epoch time of 1970-01-01 00:00:00, <i>in the time zone of the index</i> . This is generally used to define the anchor time to count from, which is relevant when the every value is > 1.

See Also

`rolling_origin()`

`slider::slide()`, `slider::slide_index()`, and `slider::slide_period()`, which power these resamplers.

Examples

```
library(vctrs)
library(tibble)
library(modeldata)
data("Chicago")

index <- new_date(c(1, 3, 4, 7, 8, 9, 13, 15, 16, 17))
df <- tibble(x = 1:10, index = index)
df

# Look back two rows beyond the current row, for a total of three rows
# in each analysis set. Each assessment set is composed of the two rows after
# the current row.
sliding_window(df, lookback = 2, assess_stop = 2)

# Same as before, but step forward by 3 rows between each resampling slice,
# rather than just by 1.
rset <- sliding_window(df, lookback = 2, assess_stop = 2, step = 3)
rset

analysis(rset$plits[[1]])
analysis(rset$plits[[2]])

# Now slide relative to the `index` column in `df`. This time we look back
# 2 days from the current row's `index` value, and 2 days forward from
# it to construct the assessment set. Note that this series is irregular,
# so it produces different results than `sliding_window()`. Additionally,
# note that it is entirely possible for the assessment set to contain no
```

```
# data if you have a highly irregular series and "look forward" into a
# date range where no data points actually exist!
sliding_index(df, index, lookback = 2, assess_stop = 2)

# With `sliding_period()`, we can break up our date index into more granular
# chunks, and slide over them instead of the index directly. Here we'll use
# the Chicago data, which contains daily data spanning 16 years, and we'll
# break it up into rolling yearly chunks. Three years worth of data will
# be used for the analysis set, and one years worth of data will be held out
# for performance assessment.
sliding_period(
  Chicago,
  date,
  "year",
  lookback = 2,
  assess_stop = 1
)

# Because `lookback = 2`, three years are required to form a "complete"
# window of data. To allow partial windows, set `complete = FALSE`.
# Here that first constructs two expanding windows until a complete three
# year window can be formed, at which point we switch to a sliding window.
sliding_period(
  Chicago,
  date,
  "year",
  lookback = 2,
  assess_stop = 1,
  complete = FALSE
)

# Alternatively, you could break the resamples up by month. Here we'll
# use an expanding monthly window by setting `lookback = Inf`, and each
# assessment set will contain two months of data. To ensure that we have
# enough data to fit our models, we'll `skip` the first 4 expanding windows.
# Finally, to thin out the results, we'll `step` forward by 2 between
# each resample.
sliding_period(
  Chicago,
  date,
  "month",
  lookback = Inf,
  assess_stop = 2,
  skip = 4,
  step = 2
)
```

Description

The tidy function from the **broom** package can be used on rset and rsplit objects to generate tibbles with which rows are in the analysis and assessment sets.

Usage

```
## S3 method for class 'rsplit'  
tidy(x, unique_ind = TRUE, ...)  
  
## S3 method for class 'rset'  
tidy(x, ...)  
  
## S3 method for class 'vfold_cv'  
tidy(x, ...)  
  
## S3 method for class 'nested_cv'  
tidy(x, ...)
```

Arguments

x	A rset or rsplit object
unique_ind	Should unique row identifiers be returned? For example, if FALSE then bootstrapping results will include multiple rows in the sample for the same row in the original data.
...	Not currently used.

Details

Note that for nested resampling, the rows of the inner resample, named inner_Row, are *relative* row indices and do not correspond to the rows in the original data set.

Value

A tibble with columns Row and Data. The latter has possible values "Analysis" or "Assessment". For rset inputs, identification columns are also returned but their names and values depend on the type of resampling. vfold_cv contains a column "Fold" and, if repeats are used, another called "Repeats". bootstraps and mc_cv use the column "Resample".

Examples

```
library(ggplot2)  
theme_set(theme_bw())  
  
set.seed(4121)  
cv <- tidy(vfold_cv(mtcars, v = 5))  
ggplot(cv, aes(x = Fold, y = Row, fill = Data)) +  
  geom_tile() + scale_fill_brewer()  
  
set.seed(4121)
```

```

rcv <- tidy(vfold_cv(mtcars, v = 5, repeats = 2))
ggplot(rcv, aes(x = Fold, y = Row, fill = Data)) +
  geom_tile() + facet_wrap(~Repeat) + scale_fill_brewer()

set.seed(4121)
mccv <- tidy(mc_cv(mtcars, times = 5))
ggplot(mccv, aes(x = Resample, y = Row, fill = Data)) +
  geom_tile() + scale_fill_brewer()

set.seed(4121)
bt <- tidy(bootstraps(mtcars, time = 5))
ggplot(bt, aes(x = Resample, y = Row, fill = Data)) +
  geom_tile() + scale_fill_brewer()

dat <- data.frame(day = 1:30)
# Resample by week instead of day
ts_cv <- rolling_origin(dat, initial = 7, assess = 7,
  skip = 6, cumulative = FALSE)
ts_cv <- tidy(ts_cv)
ggplot(ts_cv, aes(x = Resample, y = factor(Row), fill = Data)) +
  geom_tile() + scale_fill_brewer()

```

two_class_dat

Two Class Data

Description

Two Class Data

Details

There are artificial data with two predictors (A and B) and a factor outcome variable (Class). These data are now in the modeldata package.

validation_split

Create a Validation Set

Description

validation_split() takes a single random sample (without replacement) of the original data set to be used for analysis. All other data points are added to the assessment set (to be used as the validation set).

Usage

```
validation_split(data, prop = 3/4, strata = NULL, breaks = 4, ...)
```


Arguments

data	A data frame.
prop	The proportion of data to be retained for modeling/analysis.
strata	A variable that is used to conduct stratified sampling to create the resamples. This could be a single character value or a variable name that corresponds to a variable that exists in the data frame.
breaks	A single number giving the number of bins desired to stratify a numeric stratification variable.
...	Not currently used.

Details

The `strata` argument causes the random sampling to be conducted *within the stratification variable*. This can help ensure that the number of data points in the analysis data is equivalent to the proportions in the original data set. (Strata below 10% of the total are pooled together.)

Value

An tibble with classes `validation_split`, `rset`, `tbl_df`, `tbl`, and `data.frame`. The results include a column for the data split objects and a column called `id` that has a character string with the resample identifier.

Examples

```
validation_split(mtcars, prop = .9)
```

vfold_cv	<i>V-Fold Cross-Validation</i>
----------	--------------------------------

Description

V-fold cross-validation randomly splits the data into V groups of roughly equal size (called "folds"). A resample of the analysis data consisted of V-1 of the folds while the assessment set contains the final fold. In basic V-fold cross-validation (i.e. no repeats), the number of resamples is equal to V.

Usage

```
vfold_cv(data, v = 10, repeats = 1, strata = NULL, breaks = 4, ...)
```

Arguments

data	A data frame.
v	The number of partitions of the data set.
repeats	The number of times to repeat the V-fold partitioning.

strata	A variable that is used to conduct stratified sampling to create the folds. This could be a single character value or a variable name that corresponds to a variable that exists in the data frame.
breaks	A single number giving the number of bins desired to stratify a numeric stratification variable.
...	Not currently used.

Details

The `strata` argument causes the random sampling to be conducted *within the stratification variable*. This can help ensure that the number of data points in the analysis data is equivalent to the proportions in the original data set. (Strata below 10% of the total are pooled together.) When more than one repeat is requested, the basic V-fold cross-validation is conducted each time. For example, if three repeats are used with $v = 10$, there are a total of 30 splits which as three groups of 10 that are generated separately.

Value

A tibble with classes `vfold_cv`, `rset`, `tbl_df`, `tbl`, and `data.frame`. The results include a column for the data split objects and one or more identification variables. For a single repeat, there will be one column called `id` that has a character string with the fold identifier. For repeats, `id` is the repeat number and an additional column called `id2` that contains the fold information (within repeat).

Examples

```
vfold_cv(mtcars, v = 10)
vfold_cv(mtcars, v = 10, repeats = 2)

library(purrr)
data(wa_churn, package = "modeldata")

set.seed(13)
folds1 <- vfold_cv(wa_churn, v = 5)
map_dbl(folds1$splits,
  function(x) {
    dat <- as.data.frame(x)$churn
    mean(dat == "Yes")
  })

set.seed(13)
folds2 <- vfold_cv(wa_churn, strata = "churn", v = 5)
map_dbl(folds2$splits,
  function(x) {
    dat <- as.data.frame(x)$churn
    mean(dat == "Yes")
  })

set.seed(13)
folds3 <- vfold_cv(wa_churn, strata = "tenure", breaks = 6, v = 5)
map_dbl(folds3$splits,
  function(x) {
```

```
dat <- as.data.frame(x)$churn  
mean(dat == "Yes")  
})
```

Index

add_resample_id, 2
all.vars(), 8
analysis (as.data.frame.rsplitted), 4
apparent, 3
as.data.frame.rsplitted, 4
assessment (as.data.frame.rsplitted), 4
attrition, 5

bootstraps, 5
bootstraps(), 24

caret2rsample (rsample2caret), 26
complement, 6

drinks, 7

form_pred, 8

gather.rset, 9
group_vfold_cv, 10

initial_split, 11
initial_time_split (initial_split), 11
int_bca (int_pctl), 12
int_pctl, 12
int_t (int_pctl), 12

labels.rset, 14
labels.rsplitted, 15
labels.vfold_cv (labels.rset), 14
loo_cv, 15

make_splits(), 17, 18
make_strata, 16
manual_rset, 17
mc_cv, 18
mc_cv(), 24

nested_cv, 19
nested_cv(), 24

populate, 21

populate(), 7

rolling_origin, 21
rolling_origin(), 24, 29
rsample, 23
rsample-dplyr, 24
rsample2caret, 26

slide-resampling, 26
slider::slide(), 29
slider::slide_index(), 29
slider::slide_period(), 29
sliding_index (slide-resampling), 26
sliding_index(), 22
sliding_period (slide-resampling), 26
sliding_period(), 22
sliding_window (slide-resampling), 26
sliding_window(), 22
stats::terms(), 8

testing (initial_split), 11
tidy.nested_cv (tidy.rsplitted), 30
tidy.rset (tidy.rsplitted), 30
tidy.rsplitted, 30
tidy.vfold_cv (tidy.rsplitted), 30
training (initial_split), 11
two_class_dat, 32

validation_split, 32
vfold_cv, 33
vfold_cv(), 24