Package 'listarrays'

April 23, 2018

Type Package

Title A Toolbox for Working with R Arrays in a Functional Programming Style

Version 0.1.0

Description A toolbox for R arrays. Flexibly split, bind, reshape, modify, subset and name arrays.

URL https://github.com/t-kalinowski/listarrays, https://t-kalinowski.github.io/listarrays/

BugReports https://github.com/t-kalinowski/listarrays/issues

License GPL-3

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 6.0.1

Suggests testthat, magrittr, zeallot, rlang, tibble, purrr

NeedsCompilation no

Author Tomasz Kalinowski [aut, cre]

Maintainer Tomasz Kalinowski <kalinowskit@gmail.com>

Repository CRAN

Date/Publication 2018-04-23 06:27:44 UTC

R topics documented:

`array2` *Make or reshape an array with C-style (row-major) semantics*

Description

These functions reshape or make an array using C-style, row-major semantics. The returned array is still R's native F-style, (meaning, the underlying vector has been reordered).

Usage

```
array2(data, dim = length(data), dimnames = NULL)

dim2(x) <- value

set_dim2(...)
```

Arguments

`data`	what to fill the array with
`dim`	numeric vector of dimensions
`dimnames`	a list of dimnames, must be the same length as `dims`
`x`	object to set dimensions on (array or atomic vector)
`value`	a numeric (integerish) vector of new dimensions
`...`	passed on to `set_dim()`

Details

Other than the C-style semantics, these functions behave identically to their counterparts (`array2()` behaves identically to `array()`, `` `dim2<-`() `` to `` `dim<-`() ``). `set_dim2()` is just a wrapper around `set_dim(..., order = "C")`.

See examples for a drop-in pure R replacement to `reticulate::array_reshape()`

Examples

```
array(1:4, c(2,2))
array2(1:4, c(2,2))

# for a drop-in replacement to reticulate::array_reshape
array_reshape <- listarrays:::array_reshape
array_reshape(1:4, c(2,2))
```

`bind_as_dim` *Bind arrays along a specified dimension*

Description

`bind_as_*` introduces a new dimension, such that each element in `list_of_arrays` corresponds to one index position along the new dimension in the returned array. `bind_on_*` binds all elements along an existing dimension, (meaning, the returned array has the same number of dimensions as each of the arrays in the list).

Usage

```
bind_as_dim(list_of_arrays, which_dim)

bind_as_rows(...)

bind_as_cols(...)

bind_on_dim(list_of_arrays, which_dim)

bind_on_rows(...)

bind_on_cols(...)
```

Arguments

`list_of_arrays`	a list of arrays. All arrays must be of the same dimension. NULL's in place of arrays are automatically dropped.
`which_dim`	Scalar integer specifying the index position of where to introduce the new dimension to introduce. Negative numbers count from the back. For example, given a 3 dimensional array, `-1`, is equivalent to `3`, `-2` to `2` and `-3` to `1`.
`...`	Arrays to be bound, specified individually or supplied as a single list

Details

`bind_*_rows()` is a wrapper for the common case of `bind_*_dim(X, 1)`. `bind_*_cols()` is a wrapper for the common case of `bind_*_dim(X, -1)`.

Value

An array, with one additional dimension.

Examples

```
list_of_arrays <- replicate(10, array(1:8, dim = c(2,3,4)), FALSE)

dim(list_of_arrays[[1]])

# bind on a new dimension
combined_as <- bind_as_rows(list_of_arrays)
dim(combined_as)
dim(combined_as)[1] == length(list_of_arrays)

# each element in `list_of_arrays` corresponds to one "row"
# (i.e., one entry in along the first dimension)
for(i in seq_along(list_of_arrays))
  stopifnot(identical(combined_as[i,,,], list_of_arrays[[i]]))

# bind on an existing dimension
combined_on <- bind_on_rows(list_of_arrays)
dim(combined_on)
dim(combined_on)[1] == sum(sapply(list_of_arrays, function(x) dim(x)[1]))
identical(list_of_arrays[[1]], combined_on[1:2,,])
for (i in seq_along(list_of_arrays))
  stopifnot(identical(
    list_of_arrays[[i]], combined_on[ (1:2) + (i-1)*2,,]
  ))

# bind on any dimension
combined <- bind_as_dim(list_of_arrays, 3)
dim(combined)
for(i in seq_along(list_of_arrays))
   stopifnot(identical(combined[,,i,], list_of_arrays[[i]]))
```

DIM *DIM*

Description

DIM() is to dim() as NROW() is to nrow(). That is, it is identical to dim() in most cases except if the input is a bare atomic vector with no dim attribute, in which case, the length of the vector is returned instead of NULL.

Usage

```
DIM(x)
```

Arguments

x	an array or atomic vector

---

drop_dimnames *Drop dimnames*

Description

A pipe-friendly wrapper for `dim(x) <- NULL` and `dimnames(x) <- NULL` or, if `which_dim` is not `NULL`, `dimnames(x)[which_dim] <- list(NULL)`

Usage

```
drop_dimnames(x, which_dim = NULL, keep_axis_names = FALSE)

drop_dim(x)

drop_dim2(x)
```

Arguments

x	an object, potentially with dimnames
which_dim	If `NULL` (the default) then all dimnames are dropped. If integer vector, then dimnames only at the specified dimensions are dropped.
keep_axis_names	TRUE or FALSE, whether to preserve the axis names when dropping the dimanmes

---

extract_dim *Extract with [on a specified dimension*

Description

Extract with `[` on a specified dimension

Usage

```
extract_dim(X, which_dim, idx, drop = NULL, depth = Inf)

extract_rows(X, idx, drop = NULL, depth = Inf)

extract_cols(X, idx, drop = NULL, depth = Inf)
```

Arguments

X	Typically, an array, but any object with a [method is accepted (e.g., dataframe, vectors)
which_dim	A scalar integer or character, specifying the dimension to extract from
idx	A numeric, boolean, or character vector to perform subsetting with.
drop	Passed on to [. If NULL (the default), then drop is omitted from the argument, and the default is used (defaults to TRUE for most objects, including arrays)
depth	Scalar number, how many levels to recurse down if X is a list of arrays. Set this if you want to explicitly treat a list as a vector (that is, a one-dimensional array). (You can alternatively set a dim attribute with dim<- on the list to prevent recursion)

Examples

```
# extract_rows is useful to keep the same code path for arrays of various sizes
X <- array(1:8, c(4, 3, 2))
y <- c("a", "b", "c", "d")
(Y <- onehot(y))

extract_rows(X, 2)
extract_rows(Y, 2)
extract_rows(y, 2)

library(zeallot)
c(X2, Y2, y2) %<-% extract_rows(list(X, Y, y), 2)
X2
Y2
y2
```

---

modify_along_dim *Modify an array by mapping over 1 or more dimensions*

Description

This function can be thought of as a version of base::apply() that is guaranteed to return a object of the same dimensions as it was input. It also generally preserves attributes, as it's built on top of [<-.

Usage

```
modify_along_dim(X, which_dim, .f, ...)

modify_along_rows(X, .f, ...)

modify_along_cols(X, .f, ...)
```

Arguments

`X`	An array, or a list of arrays
`which_dim`	integer vector of dimensions to modify at
`.f`	a function or formula defining a function(same semantics as `purrr::map()`). The function must return either an array the same shape as it was passed, a vector of the same length, or a scalar, although the type of the returned object does not need to be the same as was passed in.
`...`	passed on to `.f()`

Value

An array, or if `X` was a list, a list of arrays of the same shape as was passed in.

Examples

```
x <- array(1:6, 1:3)
modify_along_dim(x, 3, ~mean(.x))
modify_along_dim(x, 3, ~.x/mean(.x))
```

`ndim` *Length of* `DIM()`

Description

Returns the number of dimensions, or 1 for an atomic vector.

Usage

```
ndim(x)
```

Arguments

`x`	a matrix or atomic vector

onehot_with_decoder *Convert vector to a onehot representation (binary class matrix)*

Description

Convert vector to a onehot representation (binary class matrix)

Usage

```
onehot_with_decoder(y, order = NULL, named = TRUE)

onehot(y, order = NULL, named = TRUE)

decode_onehot(Y, classes = colnames(Y), n_classes = ncol(Y) %||%
  length(classes))

onehot_decoder(Y, classes = colnames(Y), n_classes = length(classes))
```

Arguments

y	character, factor, or numeric vector
order	NULL, FALSE, or a character vector. If NULL (the default), then levels are sorted with sort(). If FALSE, then levels are taken in order of their first appearance in y. If a character vector, then order must contain all levels found in y.
named	if the returned matrix should have column names
Y	a matrix, as returned by onehot() or similar.
classes	A character vector of class names in the order corresponding to Y's onehot encoding. Typically, colnames(Y). if NULL, then the decoder returns the column number.
n_classes	The total number of classes expected in Y. Used for input checking in the returned decoder, also, to reconstruct the correct dimensions if the passed in Y is missing dim() attributes.

Value

A binary class matrix

See Also

keras::to_categorical

Examples

```
if(require(zeallot)) {
  y <- letters[1:4]
  c(Y, decode) %<-% onehot_with_decoder(y)
  Y
  decode(Y)
  identical(y, decode(Y))
  decode(Y[2,,drop = TRUE])
  decode(Y[2,,drop = FALSE])
  decode(Y[2:3,])

  rm(Y, decode)
}

# more peicemeal functions
Y <- onehot(y)
decode_onehot(Y)

# if you need to decode a matrix that lost colnames,
# make your own decoder that remembers classes
my_decode <- onehot_decoder(Y)
colnames(Y) <- NULL
my_decode(Y)
decode_onehot(Y)

# factor and numeric vectors also accepted
onehot(factor(letters[1:4]))
onehot(4:8)
```

---

seq_along_dim *Sequence along a dimension*

Description

Sequence along a dimension

Usage

```
seq_along_dim(x, which_dim)

seq_along_rows(x)

seq_along_cols(x)
```

Arguments

x	a dataframe, array or vector. For `seq_along_rows`, and `seq_along_cols` sequence along the first and last dimensions, respectively. Atomic vectors are

	treated as 1 dimensional arrays (i.e., `seq_along_rows` is equivalent to `seq_along` when `x` is an atomic vector or list).
`which_dim`	a scalar integer or character string, specifying which dimension to generate a sequence for. Negative numbers count from the back.

Value

a vector of integers 1:nrow(x), safe for use in `for` loops and vectorized equivalents.

Examples

```
for (r in seq_along_rows(mtcars[1:4,]))
  print(mtcars[r,])

x <- 1:3
identical(seq_along_rows(x), seq_along(x))
```

---

set_as_rows *Reshape an array to send a dimension forward or back*

Description

Reshape an array to send a dimension forward or back

Usage

```
set_as_rows(X, which_dim)

set_as_cols(X, which_dim)
```

Arguments

`X`	an array
`which_dim`	scalar integer or string, which dim to bring forward. Negative numbers count from the back This is a powered by `base::aperm()`.

Value

a reshaped array

See Also

`base::aperm()` `set_dim()` `keras::array_reshape()`

Examples

```
x <- array(1:24, 2:4)
y <- set_as_rows(x, 3)

for (i in seq_along_dim(x, 3))
  stopifnot( identical(x[,,i], y[i,,]) )
```

`set_dim` *Reshape an array*

Description

Pipe friendly `dim<-()`, with option to pad to necessary length. Also allows for filling the array using C style row-major semantics.

Usage

```
set_dim(x, new_dim, pad = getOption("listarrays.autopad_arrays_with", NULL),
  order = c("F", "C"), verbose = getOption("verbose"))
```

Arguments

`x`	A vector or array to set dimensions on
`new_dim`	The desired dimensions (an integer(ish) vector)
`pad`	The value to pad the vector with. `NULL` (the default) performs no padding.
`order`	whether to use row-major (C) or column major (F) style semantics. The default, "F", corresponds to the default behavior of R's `dim<-()`, while "C" corresponds to the default behavior of `reticulate::array_reshape()`, numpy, reshaping semantics commonly encountered in the python world.
`verbose`	Whether to emit a message if padding. By default, `FALSE`.

Value

Object with dimensions set

See Also

`set_dim2()`, `` `dim<-`() ``, `reticulate::array_reshape()`

Examples

```
set_dim(1:10, c(2, 5))
try( set_dim(1:7, c(2, 5)) ) # error by default, just like `dim<-`()
     set_dim(1:7, c(2, 5), pad = 99)
     set_dim(1:7, c(2, 5), pad = 99, order = "C") # fills row-wise

y <- x <- 1:4
```

```
# base::dim<- fills the array column wise
dim(x) <- c(2, 2)
x

# dim2 will fill the array row-wise
dim2(y) <- c(2, 2)
y

identical(x, set_dim(1:4, c(2,2)))
identical(y, set_dim(1:4, c(2,2), order = "C"))

## Not run:
py_reshaped <- reticulate::array_reshape(1:4, c(2,2))
storage.mode(py_reshaped) <- "integer" # reticulate coerces to double
identical(y, py_reshaped)
# if needed, see listarrays:::array_reshape() for
# a drop-in pure R replacement for reticulate::array_reshape()

## End(Not run)
```

`set_dimnames` *Set dimnames*

Description

A more flexible and pipe-friendly version of `dimnames<-`.

Usage

```
set_dimnames(x, nm, which_dim = NULL)
```

Arguments

`x`	an array
`nm`	A list or character vector.
`which_dim`	a character vector or numeric vector or `NULL`

Details

This function is quite flexible. See examples for the complete picture.

Value

x, with modified dimnames and or axisnames

Note

The word "dimnames" is slightly overloaded. Most commonly it refers to the names of entries along a particular axis (e.g., date1, date2, date3, ...), but occasionally it is also used to refer to the names of the array axes themselves (e.g, dates, temperature, pressure, ...). To disambiguate, in the examples 'dimnames' always refers to the first case, while 'axis names' refers to the second. `set_dimnames()` can be used to set either or both axis names and dimnames.

Examples

```
x <- array(1:8, 2:4)

# to set axis names, leave which_dim=NULL and pass a character vector
dimnames(set_dimnames(x, c("a", "b", "c")))

# to set names along a single axis, specify which_dim
dimnames(set_dimnames(x, c("a", "b", "c"), 2))

# to set an axis name and names along the axis, pass a named list
dimnames(set_dimnames(x, list(axis2 = c("a", "b", "c")), 2))
dimnames(set_dimnames(x, list(axis2 = c("a", "b", "c"),
                              axis3 = 1:4), which_dim = 2:3))

# if the array already has axis names, those are used when possible
nx <- set_dimnames(x, paste0("axis", 1:3))
dimnames(nx)
dimnames(set_dimnames(nx, list(axis2 = c("x", "y", "z"))))
dimnames(set_dimnames(nx, c("x", "y", "z"), which_dim = "axis2"))


# pass NULL to drop all dimnames, or just names along a single dimension
nx2 <- set_dimnames(nx, c("x", "y", "z"), which_dim = "axis2")
nx2 <- set_dimnames(nx2, LETTERS[1:4], which_dim = "axis3")
dimnames(nx2)
dimnames(set_dimnames(nx2, NULL))
dimnames(set_dimnames(nx2, NULL, 2))
dimnames(set_dimnames(nx2, NULL, c(2, 3)))
# to preserve an axis name and only drop the dimnames, wrap the NULL in a list()
dimnames(set_dimnames(nx2, list(NULL)))
dimnames(set_dimnames(nx2, list(NULL), 2))
dimnames(set_dimnames(nx2, list(axis2 = NULL)))
dimnames(set_dimnames(nx2, list(axis2 = NULL, axis3 = NULL)))
dimnames(set_dimnames(nx2, list(NULL), 2:3))
```

---

shuffle_rows *Shuffle along the first dimension multiple arrays in sync*

Description

Shuffle along the first dimension multiple arrays in sync

Usage

```
shuffle_rows(..., in_sync = TRUE)
```

Arguments

...	arrays of various dimensions (vectors and data.frames OK too)
in_sync	if the objects should be shuffled in sync (i.e., row order is the same in all returned objects)

Value

A list of objects passed on to ...

Examples

```
x <- 1:3
y <- matrix(1:9, ncol = 3)
z <- array(1:27, c(3,3,3))

if(require(zeallot)) {
  c(xs, ys, zs) %<-% shuffle_rows(x, y, z)

  l <- lapply(seq_along_rows(y), function(r) {
    list(x = x[r], y = y[r,], z = z[r,,])
  })

  ls <- lapply(seq_along_rows(y), function(r) {
    list(x = xs[r], y = ys[r,], z = zs[r,,])
  })

  stopifnot(
    length(unique(c(l, ls))) == length(l))
}
```

---

split_on_dim *Split an array along a dimension*

Description

Split an array along a dimension

Usage

```
split_on_dim(X, which_dim, f = dimnames(X)[[which_dim]], drop = FALSE,
  depth = Inf)

split_on_rows(X, f = rownames(X), drop = FALSE, depth = Inf)
```

```
split_on_cols(X, f = rownames(X), drop = FALSE, depth = Inf)

split_along_dim(X, which_dim, drop = NULL, depth = Inf)

split_along_rows(X, drop = NULL, depth = Inf)

split_along_cols(X, drop = NULL, depth = Inf)
```

Arguments

X	an array, or list of arrays. An atomic vector without a dimension attribute is treated as a 1 dimensional array (Meaning, atomic vectors without a dim attribute are only accepted if `which_dim` is `1`. Names of the passed list are preserved. If a list of arrays, all the arrays must have the same length of the dimension being split.
which_dim	a scalar string or integer, specifying which dimension to split along. Negative integers count from the back. If a string, it must refer to a named dimension (e.g, one of `names(dimnames(X))`.
f	Specify how to split the dimension. **character, integer, factor** passed on to `base::split()`. Must be the same length as the dimension being split. **a list of vectors** Passed on to `base::interaction()` then `base::split()`. Each vector in the list must be the same length as the dimension being split. **a scalar integer** used to split into that many groups of equal size **a numeric vector where** `all(f<0)` specifies the relative size proportions of the groups being split. `sum(f)` must be `1`. For example `c(0.2, 0.2, 0.6)` will return approximately a 20%-20%-60% split.
drop	passed on to `[`.
depth	Scalar number, how many levels to recurse down. Set this if you want to explicitly treat a list as a vector (that is, a one-dimensional array). (You can alternatively set dim attributes with `dim<-` on the list to prevent recursion) `split_along_dim(X, which_dim)` is equivalent to `split_on_dim(X, which_dim, seq_along_dim(X,`

Value

A list of arrays, or if a list of arrays was passed in, then a list of lists of arrays.

Examples

```
X <- array(1:8, c(2,3,4))
X
split_along_dim(X, 2)

# specify f as a factor, akin to base::split()
split_on_dim(X, 2, c("a", "a", "b"), drop = FALSE)

d <- c(10, 3, 3)
X <- array(1:prod(d), d)
```

```
y <- letters[1:10]
Y <- onehot(y)

# specify `f`` as relative partition sizes
if(require(zeallot) && require(magrittr) && require(purrr)) {

c(train, validate, test) %<-% {
  list(X = X, Y = Y, y = y) %>%
    shuffle_rows() %>%
    split_on_rows(c(0.6, 0.2, 0.2)) %>%
    transpose()
}

str(test)
str(train)
str(validate)

}


# with with array data in a data frame by splitting row-wise
if(require(tibble))
  tibble(y, X = split_along_rows(X))
```

`t.array` *transpose an array*

Description

transpose an array

Usage

```
## S3 method for class 'array'
t(x)
```

Arguments

x	an array This reverses the dimensions of an array

Examples

```
x <- array(1:27, c(3,3,3))
tx <- t(x)
for (i in 1:3)
  for(j in 1:3)
    stopifnot(x[,j,i] == tx[i,j,])
```

Index