

Package ‘listarrays’

March 24, 2019

Type Package

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

Version 0.2.0

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 6.1.1

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

NeedsCompilation no

Author Tomasz Kalinowski [aut, cre]

Maintainer Tomasz Kalinowski <kalinowskit@gmail.com>

Repository CRAN

Date/Publication 2019-03-24 21:40:02 UTC

R topics documented:

array2	2
bind_as_dim	3
DIM	4
drop_dimnames	5
expand_dims	5
extract_dim	6

map_along_dim	7
modify_along_dim	8
ndim	9
onehot_with_decoder	9
seq_along_dim	11
set_as_rows	12
set_dim	12
set_dimnames	14
shuffle_rows	15
split_on_dim	16
t.array	18

Index 19

array2	<i>Make or reshape an array with C-style (row-major) semantics</i>
--------	--

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)
```

```
matrix2(...)
```

```
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
...	passed on to set_dim()
x	object to set dimensions on (array or atomic vector)
value	a numeric (integerish) vector of new dimensions

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	<i>Bind arrays along a specified dimension</i>
-------------	--

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 dimnames

expand_dims *Expand the shape of an array*

Description

This is the inverse operation of `base::drop()`. It is analogous to `python's numpy.expand_dims()`, but vectorized on `which_dim`.

Usage

```
expand_dims(x, which_dim = -1L)
```

Arguments

x an array. Bare vectors are treated as 1-d arrays.

which_dim numeric. Desired index position of the new axis or axes in the returned array. Negative numbers count from the back. Can be any length. Throws a warning if any duplicates are provided.

Value

the array `x` with new dim

Examples

```
x <- array(1:24, 2:4)
dim(x)
dim(expand_dims(x))
dim(expand_dims(x, 2))
dim(expand_dims(x, c(1,2)))
dim(expand_dims(x, c(1,-1)))
dim(expand_dims(x, 6)) # implicitly also expands dims 4,5
dim(expand_dims(x, 4:6))

# error, implicit expansion with negative indexes not supported
try(expand_dims(x, -6))

# supply them explicitly instead
dim(expand_dims(x, -(4:6)))
```

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

<code>X</code>	Typically, an array, but any object with a [method is accepted (e.g., dataframe, vectors)
<code>which_dim</code>	A scalar integer or character, specifying the dimension to extract from
<code>idx</code>	A numeric, boolean, or character vector to perform subsetting with.
<code>drop</code>	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)
<code>depth</code>	Scalar number, how many levels to recurse down if <code>X</code> 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 <code>dim<-</code> 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
```

map_along_dim

*Apply a function across subsets along an array dimension***Description**

map_along_dim(X, dim, func) is a simple wrapper around split_along_dim(X, dim) %>% map(func). It is conceptually and functionally equivalent to base::apply(), with the following key differences:

- it is guaranteed to return a list (base::apply() attempts to simplify the output to an array, sometimes unsuccessfully, making the output unstable)
- it accepts the compact lambda notation ~.x just like in purrr::map (and modify_along_dim())

Usage

```
map_along_dim(X, .dim, .f, ..., .drop = NULL)

map_along_rows(X, .f, ..., .drop = NULL)

map_along_cols(X, .f, ..., .drop = NULL)
```

Arguments

X an R array

.dim which dimension to map along. Passed on to split_along_dim(), and accepts all the same inputs. Valid inputs include

- positive integers (index position(s) of dimension),
- negative integers (index positions(s) of dimensions, counting from the back), or
- character vector (corresponding to array dimnames)

.f A function, string of a function name, or purrr style compact lambda syntax (e.g, `~.x + 1`)

... passed on to `.f()`

.drop passed on to `[]` when subsetting the array.

Value

An R list

Examples

```
X <- matrix2(letters[1:15], ncol = 3)

apply(X, 1, function(x) paste(x, collapse = "")) # simplifies to a vector
map_along_dim(X, 1, ~paste(.x, collapse = "")) # returns a list

identical(
  map_along_rows(X, identity),
  map_along_dim(X, 1, identity)) # TRUE

identical(
  map_along_cols(X, identity),
  map_along_dim(X, -1, identity)) # TRUE
```

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	<i>Length of DIM()</i>
------	------------------------

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	<i>Convert vector to a onehot representation (binary class matrix)</i>
---------------------	--

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

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



```

# 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(...)
```

Arguments

... arrays of various dimensions (vectors and data.frames OK too)

Value

A list of objects passed on to ..., or if a single object was supplied, then the single object shuffled

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

- | | |
|------------------------|--|
| <code>X</code> | 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 <code>which_dim</code> 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. |
| <code>which_dim</code> | 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 <code>names(dimnames(X))</code>). |
| <code>f</code> | Specify how to split the dimension.
character, integer, factor passed on to <code>base::split()</code> . 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	<i>transpose an array</i>
---------	---------------------------

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

array2, 2

bind_as_cols (bind_as_dim), 3
bind_as_dim, 3
bind_as_rows (bind_as_dim), 3
bind_on_cols (bind_as_dim), 3
bind_on_dim (bind_as_dim), 3
bind_on_rows (bind_as_dim), 3

decode_onehot (onehot_with_decoder), 9
DIM, 4
dim2<- (array2), 2
drop_dim (drop_dimnames), 5
drop_dim2 (drop_dimnames), 5
drop_dimnames, 5

expand_dims, 5
extract_cols (extract_dim), 6
extract_dim, 6
extract_rows (extract_dim), 6

keras::to_categorical, 10

map_along_cols (map_along_dim), 7
map_along_dim, 7
map_along_rows (map_along_dim), 7
matrix2 (array2), 2
modify_along_cols (modify_along_dim), 8
modify_along_dim, 8
modify_along_dim(), 7
modify_along_rows (modify_along_dim), 8

ndim, 9

onehot (onehot_with_decoder), 9
onehot_decoder (onehot_with_decoder), 9
onehot_with_decoder, 9

purrr::map, 7
purrr::map(), 9

seq_along_cols (seq_along_dim), 11
seq_along_dim, 11
seq_along_rows (seq_along_dim), 11
set_as_cols (set_as_rows), 12
set_as_rows, 12
set_dim, 12
set_dim2 (array2), 2
set_dimnames, 14
shuffle_rows, 15
split_along_cols (split_on_dim), 16
split_along_dim (split_on_dim), 16
split_along_dim(), 7
split_along_rows (split_on_dim), 16
split_on_cols (split_on_dim), 16
split_on_dim, 16
split_on_rows (split_on_dim), 16

t.array, 18