Package 'behavr'

September 1, 2017

Title Canonical Data Structure for Behavioural Data

Date 2017-07-25

Version 0.3.0.9003

Description
Implements an S3 class based on data.table to store and process efficiently ethomics (high-throughput behavioural) data.

Imports data.table,
hms,
methods

Suggests testthat

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/rethomics/behavr

BugReports https://github.com/rethomics/behavr/issues

RoxygenNote 6.0.1

Roxygen list(markdown = TRUE)

R topics documented:

| behavr | *An S3 class, based on data.table, to store ethomics data* |

Description

In modern behavioural biology, it is common to record long time series of several *variables* (such as position, angle, fluorescence and many others) on multiple individuals. In addition to large multivariate time series, each individual is associated with a set of *metavariables* (i.e. sex, genotype, treatment and lifespan), which, together, form the *metadata*. Metavariables are crucial in so far as they generally "contain" the biological question. During analysis, it is therefore important to be able to access, alter and compute interactions between both variables and metavariables. behavr is a class that facilitates manipulation and storage of metadata and data in the same object. It is designed to be both memory-efficient and user-friendly. For instance, it abstracts joins between data and metavariables.

Usage

```
behavr(x, metadata)

setbehavr(x, metadata)

is.behavr(x)
```

Arguments

x	data.table containing all measurements
metadata	data.table containing the metadata

Details

A behavr table is a data.table. Therefore, it can be used by any function that would work on a data.frame or a data.table. Most of the operation such as variable creation, subsetting and joins are inherited from the data.table [] operator, following the convention DT[i,j,by] (see data table package for detail). These operations are applied on the data. Metadata can be accessed using meta=TRUE: DT[i,j,by, meta=TRUE], which allows extraction of subsets, creation of metavariables, etc.

Both x and metadata should have a **column set as key** with **the same name** (typically named id). behavr() copies x, whilst setbehavr() uses reference. metadata is always copied.

References

- The relevant rethomic tutorial section – about metavariables and variables in this context

See Also

- data.table – on which behavr is based
- xmv – to join metavariables
- rejoin – to join all metadata
- bind_behavr_list – to merge several behavr tables

Examples

```
# We generate some metadata and data
set.seed(1)
met <- data.table::data.table(id = 1:5,
                              condition = letters[1:5],
                              sex = c("M", "M", "M", "F", "F"),
                              key = "id")
data <- met[  ,
              list(t = 1L:100L,
                   x = rnorm(100),
                   y = rnorm(100),
                   eating = runif(100) > .5 ),
              by = "id"]
# we store them together in a behavr object d
# d is a copy of the data
d <- behavr(data, met)
print(d)
summary(d)

#' # we can also convert data to a behavr table without copy:
setbehavr(data, met)
print(data)
summary(data)

### Operations are just like in data.table
# row subsetting:
d[t < 10]
# column subsetting:
d[, .(id, t, x)]
# making new columns inline:
d[, x2 := 1 - x]
### Using `meta = TRUE` applies the operation on the metadata
# making new metavariables:
d[, treatment := interaction(condition,sex), meta = TRUE]
d[meta = TRUE]
```

---

bin_apply   *Bin a variable (typically time) and compute an aggregate for each bin*

Description

This function is typically used to summarise (i.e. computing an aggregate of) a variable (y) for bins of a another variable x (typically time).

Usage

```
bin_apply(data, y, x = t, x_bin_length = mins(30), wrap_x_by = NULL,
  FUN = mean, string_xy = FALSE, ...)

bin_apply_all(data, ...)
```

Arguments

data	[data.table](#) or [behavr](#) table (see details)
y	variable to be aggregated
x	variable to be binned
x_bin_length	length of the bins (same unit as x)
wrap_x_by	numeric value defining wrapping period. NULL, the default, means no wrapping.
FUN	function used to aggregate (e.g. [mean](#), [median](#), [sum](#) and so on)
string_xy	logical, whether the names of the variables are quoted
...	additional arguments to be passed to FUN

Details

bin_apply expects data from a single individual, whilst bin_apply_all works on multiple individuals identified by a unique key. wrapping is typically used to compute averages across several periods. For instance,wrap_x_by = days(1), means bins will aggregate values across several days. In this case, the resulting 'x" can be interpreted as "time relative to the onset of the day" (i.e. Zeitgeber Time).

See Also

- [behavr](#) – the documentation of the behavr object

Examples

```
query <- data.frame(experiment_id = "toy_experiment",
                     region_id = 1:5)
dt <- toy_activity_data(query, duration = days(4))

# average by 30min time bins, default
dt_binned <- bin_apply_all(dt, moving)
# equivalent to
dt_binned <- dt[, bin_apply(.SD, moving), by = "id"]

# More advanced usage
dt <- toy_dam_data(query, duration = days(4))
```

```
# nsum activity per 60 minutes
dt_binned <- bin_apply_all(dt,
                           activity,
                           x = t,
                           x_bin_length = mins(60),
                           FUN = sum)


# average activity. time in ZT
dt_binned <- bin_apply_all(dt,
                           activity,
                           x = t,
                           wrap_x_by = days(1)
                           )
```

---

bind_behavr_list   *Put together a list of [behavr](#) tables*

Description

Bind all rows of both data and metadata from a list of [behavr](#) tables into a single one. It checks keys, number and names of columns are the same across all data. In addition, it forbids to bind metadata if it would result in duplicates (same id in two different metadata).

Usage

```
bind_behavr_list(l)
```

Arguments

l   list of [behavr](#)

Value

a single [behavr](#) object

See Also

• [behavr](#) – the documentation of the behavr object

Examples

```
met <- data.table::data.table(id = 1:5,
                              condition = letters[1:5],
                              sex = c("M", "M", "M", "F", "F"),
                              key = "id")
data <- met[,list(t = 1L:100L,
                  x = rnorm(100),
                  y = rnorm(100),
                  eating = runif(100) > .5),
```

```
                  by = "id"]
d1 <- behavr(data, met)

met[,id := id+5]
data[,id := id+5]
data.table::setkeyv(met, "id")
data.table::setkeyv(data, "id")

d2 <- behavr(data, met)
d_all <- bind_behavr_list(list(d1, d2))
print(d_all)
```

---

meta   *Retrieve and set metadata*

Description

This function returns the metadata from a behavr table.

Usage

```
meta(x)

setmeta(x, new)
```

Arguments

x	behavr object
new	a new metadata table

Value

a data.table representing the metadata in x

See Also

- behavr – the documentation of the behavr object
- xmv – to join metavariables

Examples

```
set.seed(1)
met <- data.table::data.table(id = 1:5,
                               condition = letters[1:5],
                               sex = c("M", "M", "M", "F", "F"),
                               key = "id")
data <- met[,
           list(t = 1L:100L,
                x = rnorm(100),
```

```
                     y = rnorm(100),
                     eating = runif(100) > .5 ),
               by = "id"]

d <- behavr(data, met)
## show metadata
meta(d)
# same as:
d[meta = TRUE]
## set metadata
m <- d[meta = TRUE]
# only id > 2 is kept
setmeta(d, m[id < 3])
meta(d)
```

---

print.behavr   *Print and summarise a [behavr](#) table*

Description

Print and summarise a [behavr](#) table

Usage

```
## S3 method for class 'behavr'
print(x, ...)

## S3 method for class 'behavr'
summary(object, detailed = F, ...)
```

Arguments

x, object	[behavr](#) table
...	arguments passed on to further method
detailed	whether summary should be consise

See Also

- [behavr](#) – to generate x
- [print.default](#)
- [summary.default](#)

rejoin	*Join data and metadata*

Description

This function joins the data of a behavr table to its own metadata. When dealing with large data sets, it is preferable to keep metadata and data separate until a summary of data is computed. Indeed, joining many metavariables to very long time series may result in unnecessary – and prohibitively – large memory usage.

Usage

```
rejoin(x)
```

Arguments

x   behavr object

Value

a data.table

See Also

- behavr – to formally create a behavr object

Examples

```
set.seed(1)
met <- data.table::data.table(id = 1:5,
                              condition = letters[1:5],
                              sex = c("M", "M", "M", "F", "F"),
                              key = "id")
data <- met[,
            list(t = 1L:100L,
                 x = rnorm(100),
                 y = rnorm(100),
                 eating = runif(100) > .5 ),
            by = "id"]

d <- behavr(data, met)
summary_d <- d[, .(test = mean(x)), by = id]
rejoin(summary_d)
```

---

stitch_on	*Stitch behavioural data by putting together the same individuals recorded over different experiments on the basis of a user defined identifier*

Description

This function can merge rows of data from the same individual that was recorded over multiple experiments. A usual scenario in which stitch_on can be used is when an experiment is interrupted and a new recording is started on the same biological subjects. Stitching assumes the users has defined a *unique id* in the metadata that refers to a specific individual. Then, if any data that comes from the same unique id, it is merged.

Usage

```
stitch_on(x, on, time_ref = "datetime", use_time = F, time_variable = "t")
```

Arguments

x	behavr object
on	name of a metavariable serving as a unique id (per individual)
time_ref	name of a metavariable used to align time (e.g. "date", or "datetime")
use_time	whether to use time as well as date
time_variable	name of the variable describing time

Details

When several rows of the metadata match a unique id (several experiments), the first (in time) experiment is used as the reference id. The data from the following one(s) will be added with a time lag equals to the difference between the values of time_ref. When data is not aligned to circadian time, it makes sense to set use_time = TRUE. Otherwise, the assumption is that the time is already aligned to a circadian reference, so only the date is used.

Value

a behavr table

See Also

- behavr – to formally create a behavr object

Examples

```
set.seed(1)
met1 <- data.table::data.table(uid = 1:5,id = 1:5,
                                condition = letters[1:5],
                                sex=c("M","M","M","F", "F"),
                                key="id")
met2 <- data.table::data.table(uid = 1:4,id = 6:9,
                                condition = letters[1:4],
                                sex=c("M","M","M","F"),
                                key="id")
met1[, datetime := as.POSIXct("2015-01-02")]
met2[, datetime := as.POSIXct("2015-01-03")]
met <- rbind(met1, met2)
data.table::setkeyv(met, "id")
t <- 1L:100L
data <- met[,list(t=t, x=rnorm(100),y=rnorm(100), eating=runif(100) > .5 ),by="id"]
d <- behavr(data,met)
summary(d)
d2 <- stitch_on(d, on ="uid")
summary(d2)
```

---

time_conversion   *Time conversion utilities*

Description

Trivial functions to convert time to seconds – as behavr uses second as a conventional unit of time.

Usage

```
days(x)

hours(x)

mins(x)
```

Arguments

x   numeric vector to be converted in second

Details

Most functions in the rethomics framework will use seconds as a unit of time. It is always preferable to call a function like my_function(days(1.5)) rather than my_function(60*60*24*1.5).

Value

number of seconds corresponding to x (1d = 86400s, 1h = 3600s and 1min = 60s)

| toy_activity_data | *Generate toy activity and sleep data mimicking Drosophila behaviour in tubes* |

Description

This function generates random data that emulates some of the features of fruit fly activity and sleep. This is designed **exclusively to provide material for examples and tests** as it generates "realistic" datasets of arbitrary length.

Usage

```
toy_activity_data(query = NULL, seed = 1, rate_range = 1/c(60, 10),
  duration = days(5), sampling_period = 10, ...)

toy_ethoscope_data(...)

toy_dam_data(...)
```

Arguments

query	[data.frame](#) where every row defines an animal. Typically queries have, at least, the columns experiment_id and region_id. The default value (NULL), will generate data for a single animal.
seed	random seed used (see [set.seed](#))
rate_range	parameter defining the boundaries of the rate at which animals wake up. It will be uniformly distributed between animals, but fixed within each animal.
duration	length (in seconds) of the data to generate
sampling_period	
	sampling period (in seconds) of the resulting data
...	additional arguments to be passed to simulate_animal_activity

Value

a [behavr](#) table with the query columns as metavariables. In addition to id and t columns different methods will output different variables:

- toy_activity_data will have asleep and moving (1/10s)
- toy_dam_data will have activity (1/60s)
- toy_ethoscope_data will have xy_dist_log10x1000, has_interacted and x (2/1s)

See Also

- [behavr](#) – to formally create a behavr object

Examples

```
# just one animal, no query needed
dt <- toy_ethoscope_data(duration = days(3))

# advanced, using a query
query<- data.frame(experiment_id = "toy_experiment",
                   region_id = 1:10,
                   condition = c("A", "B"))


# Data that could come from loadEthoscopeData:
dt <- toy_ethoscope_data(query, duration = days(1))
print(dt)

# Some DAM-like data
dt <- toy_dam_data(query, seed = 2, duration = days(3))
print(dt)

# data where behaviour is annotated e.g. by a classifier
dt <- toy_activity_data(query, 3)
print(dt)
```

---

xmv   *Expand a metavariable and map it against the data*

Description

This function eXpands a MetaVariable from a parent [behavr](#) object. That is, it matches this variable (from metadata) to the data *by id*.

Usage

```
xmv(var)
```

Arguments

var   the name of the variable to be extracted

Details

This function *can only be called within between the* [] *of a parent* [behavr](#) object. It is intended to facilitate operations between data and metadata. For instance, when one wants to modify a column of the data according to a metavariable.

Value

a vector of the same type as var, but of the same length as the number of row in the parent data. As each row of data is matched against metadata for this specific variable.

See Also

- behavr – to formally create a behavr object
- rejoin – to join all metadata with data

Examples

```
#### First, we create some data

library(data.table)
set.seed(1)
data <- data.table(
                   id = rep(c("A", "B"), times = c(10,26)),
                   t = c(1:10, 5:30),
                   x = rnorm(36), key = "id"
                   )

metadata = data.table(id = c("A", "B"),
                      treatment = c("w", "z"),
                      lifespan = c(19, 32),
                      ref_x = c(1, 0),
                      key = "id")
dt <- behavr(data, metadata)
summary(dt)

#### Subsetting using metadata

dt[xmv(treatment) == "w"]
dt[xmv(treatment) == "w"]
dt[xmv(lifespan) < 30]

#### Allocating new columns using metavariable

# Just joining lifespan (not necessary)
dt[, lif := xmv(lifespan)]
print(dt)
# Anonymously (more useful)
dt[, x2 := x - xmv(ref_x)]
print(dt)
```

Index