Package: littlelisteners (via r-universe)

September 10, 2024

Title Helper Functions for Hand-coded Eyetracking Data

Version 0.0.0.9000

Description This package houses frequently used functions for working with hand-coded eyetracking data.

Depends R (>= 4.1.0)

License MIT + file LICENSE

LazyData true

RoxygenNote 7.3.2

Imports dplyr, ggplot2, readr, stringr, tidyr, tidyselect, tibble, rlang, meltr

Roxygen list(markdown = TRUE)

Encoding UTF-8

URL http://www.tjmahr.com/littlelisteners/

Config/Needs/website rmarkdown

Suggests rprime, tidyverse

Repository https://tjmahr.r-universe.dev

RemoteUrl https://github.com/tjmahr/littlelisteners

RemoteRef HEAD

RemoteSha 9eaf3e9796428adf30a6f128bb497275e86b9c49

Contents

`add_aois` *Map the x and y positions of looks to Areas of Interest.*

Description

Map the x and y positions of looks to Areas of Interest.

Usage

```
add_aois(x, aois, default_onscreen = "tracked")
```

Arguments

`x`	a dataframe of looking data
`aois`	an AOI or list of AOIs
`default_onscreen`	default label to use for a look onscreen that does not fall into an AOI. Default is `"tracked"`

Details

The function current assumes the conventions used in our lab. It will create a column called `GazeByAOI` with the label of the AOI for each look. It does this by checking whether the columns `XMean` and `YMean` fall into to the boundaries of the AOI.

Offscreen looks receive `NA`.

Value

an updated dataframe

adjust_times *Adjust looking times relative to some event*

Description

This function is useful if some critical event occurs each trial, and we would like to adjust the timestamps so that they are relative to that event time.

Usage

```
adjust_times(
  data,
  time_var = quote(Time),
  event_var = NULL,
  ...,
  align = TRUE,
  fps = 60,
  ties = "first"
)
```

Arguments

data	a long data frame of looking data
time_var	a column in data with looking times (assumed to be milliseconds).
event_var	a column in data with the time of some event
...	grouping variables. The grouping variables should uniquely specify a trial of eyetracking data.
align	whether to align the eyetracking times so that the frame closest to the event time gets time = 0.
fps	the eyetracking sampling rate. Defaults to 60 frames per second.
ties	how to break ties when the smallest times are equally close to zero. Default is "first" so that the tie c(-1, 1) is aligned to c(0, 2).

Value

the looking data with the times adjusted by event times. By default, these times are aligned so that the frame closest to the event time gets value 0.

Examples

```
# Consider some raw tims from an eyetrack. For each trial, some critical
# event occurs and we have a column with the time of that event for each
# trial.
trial1 <- data.frame(trial = 1, time_ms = 1:5, event = 2)
trial2 <- data.frame(trial = 2, time_ms = 6:10, event = 8.5)
trial_times <- dplyr::bind_rows(trial1, trial2)
```

```
trial_times

# We want to adjust the times so that time 0 is time of the critical event.
adjust_times(trial_times, time_ms, event, trial, fps = 1000)

# The times are adjusted so that the frame closest to the event time gets
# the time zero. Setting `align` to `FALSE` skips this behavior.
adjust_times(trial_times, time_ms, event, trial, align = FALSE, fps = 1000)

# In the second trial there is a tie. Two frames are equally close to 0. By
# default the first frame is chosen to be zero, but setting `ties` to
# `"last"` will break ties with the later frame.
adjust_times(trial_times, time_ms, event, trial, ties = "last", fps = 1000)
```

`adjust_times_around_zero` *Adjust time values around 0*

Description

Adjust time values around 0

Usage

```
adjust_times_around_zero(data, time_col = "Time", fps = 60, ties = "first")
```

Arguments

`data`	a dataframe of eyetracking data for a single trial or a grouped dataframe where the groups define a single trial.
`time_col`	name (string) of the column with time value. Defaults to `"Time"`.
`fps`	the eyetracking sampling rate. Defaults to 60 frames per second.
`ties`	how to break ties when the smallest times are equally close to zero. Default is `"first"` so that the tie `c(-1, 1)` is aligned to `c(0, 2)`.

Value

the dataframe with updated time values

aggregate_looks *Aggregate looks*

Description

Aggregate the number of looks to each response type over some grouping variables like Subject, Time, Condition.

Usage

```
aggregate_looks(data, resp_def, formula)

aggregate_looks2(data, resp_def, resp_var, ...)
```

Arguments

data	a long data frame of looking data
resp_def	a response definition or a list of response definition.
formula	an aggregation formula. The lefthand terms will be grouping variables, and the righthand term is the column with eyetracking responses.
resp_var	Name of the column that contains eyetracking responses
...	Grouping columns.

Details

This function is the main tool for preparing eyetracking data for a growth curve analysis. For example, an aggregation formula like `Subject + Time ~ Gaze` would provide the number of looks to each image over time for each subject.

`aggregate_looks()` uses an aggregation formula like `stats::aggregate()`, whereas `aggregate_looks2()` uses column names.

Value

a dataframe of the grouping columns along with the number of looks to each response type, the proportion (and standard error) of looks to the primary response, and the proportion (and standared error) of missing data.

Examples

```
target_def <- create_response_def(
  label = "looks to target",
  primary = "Target",
  others = c("PhonologicalFoil", "SemanticFoil", "Unrelated"),
  elsewhere = "tracked",
  missing = NA)

four_image_data |>
```

```
  aggregate_looks(target_def, Subject + TrialNo ~ GazeByImageAOI)

four_image_data |>
  aggregate_looks(target_def, Subject ~ GazeByImageAOI) |>
  str()

# With column names
four_image_data |>
  aggregate_looks2(target_def, GazeByImageAOI, Subject, TrialNo)

four_image_data |>
  aggregate_looks2(target_def, GazeByImageAOI, Subject) |>
  str()

phonological_def <- create_response_def(
  label = "looks to phonological foil",
  primary = "PhonologicalFoil",
  others = c("Target", "SemanticFoil", "Unrelated"),
  elsewhere = "tracked",
  missing = NA)

# Aggregate looks to multiple response definitions at once
defs <- list(target_def, phonological_def)
four_image_data |>
  aggregate_looks(defs, Subject + BlockNo ~ GazeByImageAOI) |>
  dplyr::select(.response_def, Subject, BlockNo, Primary:PropNA) |>
  dplyr::mutate(
    Prop = round(Prop, 3),
    PropSE = round(PropSE, 3),
    PropNA = round(PropNA, 3)
  )

# Compute a growth curve
growth_curve <- four_image_data |>
  adjust_times(Time, TargetOnset, Subject, BlockNo, TrialNo) |>
  aggregate_looks(target_def, Time ~ GazeByImageAOI) |>
  dplyr::filter(-1000 <= Time, Time <= 2000)

library(ggplot2)
ggplot(growth_curve) +
  aes(x = Time, y = Prop) +
  geom_hline(linewidth = 2, color = "white", yintercept = .25) +
  geom_vline(linewidth = 2, color = "white", xintercept = 0) +
  geom_pointrange(aes(ymin = Prop - PropSE, ymax = Prop + PropSE)) +
  labs(
    y = "Proportion of looks to target",
    x = "Time relative to target onset [ms]"
   ) +
  theme_grey(base_size = 14)
```

---

assign_bins *Assign bins for downsampling looking data*

Description

Assign bins for downsampling looking data

Usage

```
assign_bins(
  data,
  bin_width = 3,
  time_var,
  ...,
  bin_col = ".bin",
  na_location = "tail",
  partial = FALSE
)
```

Arguments

data	a dataframe of looking data
bin_width	the number of items to put in each bin. Default is 3.
time_var	the name of the column representing time
...	grouping variables
bin_col	name of the column to add. Defaults to ".bin".
na_location	Where to assign NA bin numbers. "head" and "tail" respectively put the NA elements at the head and tail of the vector; "split" alternates between "tail" and "head".
partial	whether to exclude values that don't fit evenly into bins. Defaults to FALSE, so that the user is warned if a bin is incomplete.

Value

the original dataframe with an added column of bin numbers. The dataframe will be sorted by the grouping and time variables.

---

convert_datawiz_code_to_aoi

Convert to DataWiz codes to AOI names

Description

Convert to DataWiz codes to AOI names

Usage

```
convert_datawiz_code_to_aoi(xs)
```

Arguments

xs	a vector of DataWiz codes (-, ., 0, 1)

Value

the vector with NA for "-", "Target" for "1", "Distractor" for "0", and "tracked" for ".".

---

create_aoi *Create an AOI object*

Description

Create an object representing an Area of Interest (AOI). Only rectangles are supported (like a jpeg image in an experiment). Pixel (0,0) is the lower left corner of the screen.

Usage

```
create_aoi(aoi_name, x_pix, y_pix, screen_width = 1920, screen_height = 1080)
```

Arguments

aoi_name	label of the AOI
x_pix	location of the left and right edges in pixels.
y_pix	location of the bottom and top edges in pixels.
screen_width	width of the screen in pixels. Defaults to 1920.
screen_height	width of the screen in pixels. Defaults to 1080.

Value

an AOI object.

---

create_response_def *Create a response definition*

Description

A response definition controls how `aggregate_looks()` works.

Usage

```
create_response_def(
  primary,
  others,
  elsewhere = NULL,
  missing = NA,
  label = NULL
)
```

Arguments

primary	the primary response of interest
others	other responses of interest
elsewhere	responses to ignore
missing	responses that indicate missing data. Defaults to NA.
label	optional label for the response definition. Defaults to the value of primary.

Details

To deal with eyetracking data in a generic way, we need a way to describe eyetracking responses. We assume that there are four basic gaze types.

- Primary responses: A gaze to a primary or target image.
- Other responses: Gazes to competing images.
- Elsewhere looks: A gaze that is onscreen but not a primary or other response. Typically, this occurs when the participant is shifting between images.
- Missing looks: A missing or offscreen gaze.

A *response definition* is a programmatic way of describing these response types, and it allows aggregate_looks() to map gaze data onto looking counts.

Value

a response_def object

Examples

```
create_response_def(
  label = "looks to target",
  primary = "Target",
  others = c("PhonologicalFoil", "SemanticFoil", "Unrelated"),
  elsewhere = "tracked",
  missing = NA)
```

---

cycle_response_def *Create complementary response definitions*

Description

In the typical response definition, there is a primary response compared to other competitors. Oftentimes, we are interested in also comparing each of the competitors to the other images. This function quickly assembles a full cycle of response definitions.

Usage

```
cycle_response_def(response_def)
```

Arguments

response_def	a response definition to use a template for other definitions.

Value

a list of response definitions where each member of `c(response_def$primary, response_def$others)` is used as the primary response.

Examples

```
# Create one definition
def <- create_response_def(
  primary = 1,
  others = c(5, 8, 9),
  elsewhere = 0,
  missing = NA
)

# Create the full cycle of response definitions
cycle_response_def(def)
```

---

empirical_logit *Compute empirical logit*

Description

Compute empirical logit

Usage

```
empirical_logit(x, y)

empirical_logit_weight(x, y)
```

Arguments

x	vector containing number of looks to target
y	vector containing number of looks to distractors

Value

`empirical_logit` returns the empirical logit of looking to target. `empirical_logit_weight` returns weights for these values.

References

Dale Barr's Walkthrough of an "empirical logit" analysis in R

example_files *Locate the path of example eyetracking files*

Description

Locate the path of example eyetracking files

Usage

```
example_files(which = 1)
```

Arguments

which	index of the batch of example files to load

Details

This function is a wrapper over `system.file()` to locate the paths to bundled eyetracking data. These files are used to test or demonstrate functionality of the package.

The following sets of files are included:

1. `Coartic_Block1_001P00XS1` - Data for a block of trials from an eyetracking performed with a Tobii Eyetracker in an Eprime experiment.
2. `Coartic_Block2_001P00XS1` - Data for a second block of trials from the above experiment.

Value

Paths to a banch of examples files bundled with the `littlelisteners` package.

four_image_data *Example data from a Visual World experiment*

Description

Example data from a Visual World experiment

Usage

```
four_image_data
```

Format

A data frame with 20,910 rows and 25 variables

`interpolate_looks` *AOI-based gaze interpolation*

Description

Fills in windows of missing data if the same AOI is fixated at beginning and end of the missing data window. For example, the sequence `"Target", NA, NA, "Target"` would be interpolated to be `"Target", "Target", "Target", "Target"`.

Usage

```
interpolate_looks(
  x,
  window,
  fps,
  response_col,
  interp_col,
  fillable,
  missing_looks
)
```

Arguments

`x`	a dataframe of grouped eyetracking data. Each row should be a single frame of eyetracking. Use `dplyr::group_by()` to set the grouping columns for the data. The groups should specify a single trial of eyetracking data.
`window`	maximum amount of missing data (milliseconds) that can be interpolated. Only spans of missing data with less than or equal to this duration will be interpolated
`fps`	number of eyetracking frames (dataframe rows) per second
`response_col`	(character) name of the column with the eyetracking response data
`interp_col`	(character) name of a column to add to the dataframe. This column records whether each frame was interpolated (TRUE) or not (FALSE)
`fillable`	values in the response column where interpolation is legal. These would typically be AOI locations.
`missing_looks`	values that can be imputed.

Details

Use `window` to constrain the duration of missing data windows that can be filled. We conventionally use 150ms because we would not expect someone to shift their gaze from Image A to Image B to Image A in that amount of time.

Examples

```
# We have time in ms, measured at 60 fps, and
# we want to fill in gaps of 100 ms.
looks <- tibble::tribble(
  ~Subject, ~Trial, ~Time,    ~AOI,         ~Hint,
       "A",      1,  1000,  "Left",     "present",
       "A",      1,  1017,  "Left",     "present",
       "A",      1,  1034,      NA,   "legal gap",
       "A",      1,  1051,      NA,   "legal gap",
       "A",      1,  1068,      NA,   "legal gap",
       "A",      1,  1084,  "Left",     "present",
       "A",      1,  1100,      NA, "illegal gap",
       "A",      2,   983,  "Left",     "present",
       "A",      2,  1000, "Right",     "present",
       "A",      2,  1017,      NA, "illegal gap",
       "A",      2,  1034,      NA, "illegal gap",
       "A",      2,  1051,      NA, "illegal gap",
       "A",      2,  1068,      NA, "illegal gap",
       "A",      2,  1084,      NA, "illegal gap",
       "A",      2,  1100,      NA, "illegal gap",
       "A",      2,  1118,      NA, "illegal gap",
       "A",      2,  1135, "Right",     "present",
)

# Note that only the "legal gap" rows were interpolated
looks |>
  dplyr::group_by(Trial) |>
  interpolate_looks(
    window = 100,
    fps = 60,
    response_col = "AOI",
    interp_col = "Interpolated",
    fillable = c("Left", "Right"),
    missing_looks = NA
  )
```

`melt_datawiz` *Convert DataWiz data into long format*

Description

DataWiz files have several columns F0, F33, F67, etc. for each time sample. This function converts a dataframe from such a file into a long format, where there is a single time column and single column of gaze responses.

Usage

```
melt_datawiz(df, key_col = "Time", value_col = "Look")
```

Arguments

df	a dataframe created by reading a datawiz file
key_col	the name of the new column that holds the time values
value_col	the name of the new column that holds the looking data at each time sample

Value

a long data-frame

---

read_datawiz *Read eyetracking data from a datawiz file*

Description

Read eyetracking data from a datawiz file

Usage

```
read_datawiz(filename, sampling_rate = 33.3333)
```

Arguments

filename	a txt file generated by datawiz
sampling_rate	the rate of the video recording in ms. By default, the value is 33.3 for 1 frame every 33.3 ms.

Details

The files exported by DataWiz are a series of tab-separated data files all combined into a single file. This means that the header row (with the column names separated by tabs) will be repeated throughout the file. These repeated header rows are removed.

The header rows indicate the time of eyetracking samples by columns named "F0", "F33", "F67", etc. There are also columns with blank names before column "F0". These are also looking samples before "F0". This function back-fills the column names so that the first column before "F0" changes from " " to "X33", where the X indicates a negative time sample.

Value

a dataframe containing the cleaned-up eyetracking data

read_gazedata *Load a* .gazedata *file for an experiment*

Description

Loads .gazedata file created by an Eprime experiment running on a Tobii eyetracker, and performs typical data reduction on that file.

Usage

```
read_gazedata(
  gazedata_path,
  eyes = "both",
  means_need_both = FALSE,
  apply_corrections = TRUE
)
```

Arguments

gazedata_path	path to the .gazedata file that is to be parsed.
eyes	string describing which eye(s) should be selected for the Mean columns. Valid options are "both", "left", and "right". Defaults to "both". If "left" is selected, then only the left eye is used to calculate the XMean, YMean, etc. columns.
means_need_both	logical value indicating if both eyes are required to compute Mean columns. Defaults to FALSE. If FALSE, NA values are ignored, so for example, XMean could be computed from an XLeft of .25 and an XRight of NA.
apply_corrections	whether to do low-level adjustments like coding offscreen looks as NA, negative pupil diameters to NA, negative distances to NA, and flip y-axis so the origin is the lower-left corner. Defaults to TRUE. Only used as FALSE in case "raw" data is needed.

Details

We extract the columns the following columns: TrialId, RTTime, XGazePosLeftEye, XGazePosRightEye, YGazePosLeftEye, YGazePosRightEye, DistanceLeftEye, DistanceRightEye, DiameterPupilLeftEye and DiameterPupilRightEye.

Once these column values are loaded, we make three modifications to the gazedata (when apply_corrections is TRUE).

1. Gaze measurements with Validity codes greater than or equal to 1 are replaced with NA values.
2. X,Y gaze values are defined in screen proportions. Values that fall outside [0,1] are outside of the boundaries of the screen and therefore are nonsensical. Replace them with NA. We perform a similar correction on pupil diameters and eye-distances by replacing negative values with NA.

3. The origin of the screen is the upper-left-hand corner of the screen. Flip the y-values so that the origin is in a more familiar position in the lower-left-hand corner of the screen. This way, low y values are closer to the bottom of the screen.
4. Compute the mean x, y, distance and diameter values for the left and right eyes. `NA` values are ignored when computing the mean, so the pair `(XLeft = NA, XRight = .5)` yields `XMean = .5`.

Value

A dataframe containing the parsed gazedata. Each row of the dataframe contains the eye-tracking data for a single frame of time recorded during the experiment.

References

Tobii Toolbox for Matlab: Product Description & User Guide

Examples

```
gazedata_path <- example_files(1)[1]
read_gazedata(gazedata_path)
```

`se_prop` *Standard error for proportions*

Description

See http://www.r-tutor.com/elementary-statistics/interval-estimation/interval-estimate-population-proportion

Usage

```
se_prop(proportion, n_possible)
```

Arguments

`proportion`	proportions of hits
`n_possible`	numbers of total events

Value

the standard errors of the proportion estimates

`trim_to_bin_width` *Truncate times to fit bin width*

Description

Samples of eyetracking data are excluded so that the number of frames is evenly divisible by a given bin width. For example, given a bin width of 3 frames, a trial with 181 frames would lose 1 frame. The frames aligned so that a key time value have a specific position in a bin. For example, setting time 0 to position 1 will truncate the times so that time 0 will be the first frame inside of its bin.

Usage

```
trim_to_bin_width(
  data,
  bin_width = 3,
  key_time = NULL,
  key_position = 1,
  time_var,
  ...,
  min_time = NULL,
  max_time = NULL
)
```

Arguments

`data`	a dataframe of looking data
`bin_width`	the number of items to put in each bin. Default is 3.
`key_time`, `key_position`	arguments controlling the trimming. The given time value (`key_time`) will have a specific position within a bin (`key_position`). For example, given a value of 0 and position of 2, the trimming will force the frame with time 0 to fall in the second frame of its bin.
`time_var`	the name of the column representing time
`...`	grouping variables
`min_time`, `max_time`	optional arguments controlling the trimming. If used, the time values are filtered to exclude whole bins of frames before `min_time` and after `max_time`.

Value

the original dataframe with its time column trimmed to make it easier to bin time values into groups of `bin_width`.

Examples

```
data1 <- tibble::tibble(
  task = "testing",
  id = "test1",
  time = -4:6,
  frame = seq_along(time)
)

data2 <- tibble::tibble(
  task = "testing",
  id = "test2",
  time = -5:5,
  frame = seq_along(time)
)

# Number of rows per id is divisible by bin width
# and time 0 is center of its bin
dplyr::bind_rows(data1, data2) |>
  trim_to_bin_width(3, key_time = 0, key_position = 2, time, id) |>
  assign_bins(3, time, id) |>
  dplyr::group_by(id, .bin) |>
  dplyr::mutate(center_time = median(time))

# And exclude times in bins before some minimum time
dplyr::bind_rows(data1, data2) |>
  trim_to_bin_width(
    bin_width = 3,
    key_time = 0,
    key_position = 2,
    time,
    id,
    min_time = -1
) |>
  assign_bins(3, time, id)

# And exclude times in bins after some maximum time
dplyr::bind_rows(data1, data2) |>
  trim_to_bin_width(
    bin_width = 3,
    key_time = 0,
    key_position = 2,
    time, id,
    min_time = -1,
    max_time = 4
  ) |>
  assign_bins(3, time, id)
```

Index