Package 'NetFACS'

January 20, 2025

Title Network Applications to Facial Communication Data

Version 0.5.0

Date 2022-12-06

Description Functions to analyze and visualize communication data,
based on network theory and resampling methods.
Farine, D. R. (2017) <doi:10.1111/2041-210X.12772>;
Carsey, T., & Harden, J. (2014) <doi:10.4135/9781483319605>.
Primarily targeted at datasets of facial expressions coded with the Facial Action Coding System.
Ekman, P., Friesen, W. V., & Hager, J. C. (2002). ``Facial action coding system -
investigator's guide'' <https://www.paulekman.com/facial-action-coding-system/>.

License Apache License (>= 2.0)

Encoding UTF-8

LazyData true

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Depends R (>= 3.5.0)

Imports arrangements, doParallel, dplyr, igraph, ggplot2, ggraph,
magrittr, patchwork, parallel, picante, rlang, Rfast, tibble,
tidygraph, tidyr, vctrs, methods

RoxygenNote 7.2.1

NeedsCompilation yes

ByteCompile true

VignetteBuilder knitr

Config/testthat/edition 3

Author Alex Mielke [aut],
Bridget M. Waller [aut],
Claire Perez [aut],
Alan V. Rincon [aut, cre],
Julie Duboscq [aut],
Jerome Micheletta [aut]

Maintainer Alan V. Rincon <avrincon1@gmail.com>

Repository CRAN

Date/Publication 2022-12-06 17:32:35 UTC

Contents

`add_inactive_single_units` *Add inactive (missing) single units*

Description

Add inactive (missing) single units

Usage

```
add_inactive_single_units(d, single.units)
```

Arguments

`d`	A dataframe, result of `probability_of_combination`
`single.units`	A character vector of single AUs

`calculate_combination_size` *Calculate combination size*

Description

Calculate combination size

Usage

```
calculate_combination_size(x)
```

Arguments

`x`	A character vector of AU combinations, sep by _

Value

A vector

`conditional_probabilities` *Summarise dyadic combination of elements*

Description

For all dyadic combinations that appear in the test dataset, this function returns the probability of A occurring (P(A)), the probability of B occurring (P(B)), the probability of A and B occurring simultaneously (P(AandB)) and, the probability of A given B (P(A|B)).

Usage

```
conditional_probabilities(netfacs.data)
```

Arguments

`netfacs.data`	An object of class `netfacs` or `netfacs_multiple`

Value

A summary `tibble`

See Also

`network_conditional`

Examples

```
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 50,
  combination.size = 2
)

conditional_probabilities(angry.face)
```

define_contexts *Define truth for AUs active in different contexts*

Description

Define truth for AUs active in different contexts

Usage

```
define_contexts(aus, n_active_aus, contexts = NULL, au_fidelity = 1)
```

Arguments

aus	A character vector of AUs
n_active_aus	A numeric vector, the same length as contexts, indicating the number of AUs active per context.
contexts	A character vector of contexts
au_fidelity	A number between 1 and 0.5, indicating the probability that an AU is active in a context.

Value

A matrix of probabilities with contexts in rows and AUs in columns

define_joint_prob *Joint probability distribution of AUs*

Description

Joint probability distribution of AUs

Usage

```
define_joint_prob(aus, n_jp = 2, min_jp = 0.5)
```

Arguments

aus	A character vector of AUs
n_jp	Number of joint probabilities >0
min_jp	Minimum joint probability. Must be between 0 and 1

`distribution.plot` *Plots the observed probability for an element against the distribution of the null model*

Description

The function takes all single elements in a netfacs object, and plots the distribution of probabilities under the null hypothesis, marking where the observed probability falls

Usage

```
distribution.plot(netfacs.data)
```

Arguments

`netfacs.data`	object resulting from netfacs() function

Value

Function returns a ggplot showing for each element the distribution of expected probabilities (blue) and the observed probability (black line)

Examples

```
### how do angry facial expressions differ from non-angry ones?
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 100,
  combination.size = 2
)

# show distribution of AU4
distribution.plot(netfacs.data = angry.face)$"4"
```

`element.plot` *Plots the observed and expected probabilities for the basic elements based on the condition*

Description

The function takes all single elements in a netfacs object, and plots the observed value and the expected value based on all randomisations

Usage

```
element.plot(netfacs.data)
```

Arguments

netfacs.data	object resulting from netfacs() function

Value

Function returns a ggplot showing for each element the observed probability and expected probability

Examples

```
### how do angry facial expressions differ from non-angry ones?
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 100,
  combination.size = 2
)
# plot all
element.plot(netfacs.data = angry.face)
```

---

element.specificity *(Defunct) Tests how much each element increases the specificity of all combinations it is in*

Description

This function is defunct Please see specificity_increase instead

Usage

```
element.specificity(netfacs.data)
```

Arguments

netfacs.data	object resulting from netfacs function

emotions_set *Letter Data*

Description

Data from the Extended Cohn-Kanade database, FACS data and emotions for posed images

Usage

```
data(emotions_set)
```

Format

An object of class.

References

Lucey P, Cohn JF, Kanade T, Saragih J, Ambadar Z, Matthews I (2010) The extended Cohn-Kanade dataset (CK+): A complete dataset for action unit and emotion-specified expression. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, CVPRW 2010. pp 94-101

entropy.overall *(Deprecated) Calculate information content of the dataset*

Description

This function is deprecated. Please see `entropy_overall` instead

Usage

```
entropy.overall(x, netfacs.data)
```

Arguments

x	An object of class `netfacs` or simply a binary matrix of 0s and 1s, with elements in columns and events in rows.
netfacs.data	deprecated. Please use x instead.

`entropy_overall` *Calculate information content of the dataset*

Description

Compares the observed and expected information content of the dataset.

Usage

```
entropy_overall(x)
```

Arguments

`x`	An object of class `netfacs` or simply a binary matrix of 0s and 1s, with elements in columns and events in rows.

Value

Function returns a summary `tibble` containing the observed entropy, expected entropy and entropy ratio (observed / expected) of the dataset. Observed entropy is calculated using Shannon's information entropy formula $-\sum_{i=1}^{n} p_i \log(p_i)$. Expected entropy is based on randomization (shuffling the observed elements while maintaining the number of elements per row) and represents the maximum entropy that a dataset with the same properties as this one can reach. Ratios closer to 0 are more ordered; ratios closer to 1 are more random.

References

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Examples

```
### how do angry facial expressions differ from non-angry ones?
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 100,
  combination.size = 2
)

entropy_overall(angry.face)
```

equal_observations *Check that ALL objects have the same number of observations*

Description

lenght(vector), nrow(matrix), nrow(dataframe)

Usage

```
equal_observations(x, ...)
```

Arguments

x	Object to compare number of observations
...	Additional objects to compare number of observations

Value

Logical

event_size_plot *Plots the probability that a combination of a certain size appears*

Description

The function takes all combination size in a netfacs object, and plots the distribution of ratios between the observed value and all randomisations

Usage

```
event_size_plot(netfacs.data)

event.size.plot(netfacs.data)
```

Arguments

netfacs.data	object resulting from netfacs() function

Value

Function returns a ggplot showing for each combination size the observed and expected probabilities of occurrance

Examples

```
### how do angry facial expressions differ from non-angry ones?
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 100,
  combination.size = 2
)

event_size_plot(angry.face)
```

---

get_active_elements *Extract active elements from matrix*

Description

Extract active elements from matrix

Usage

```
get_active_elements(m)
```

Arguments

m	A binary matrix where 1 indicates an element was active. colnames(m) must contain the element names

Value

A list of vectors

---

get_data *Extract used data from a* netfacs *object*

Description

Extract used data from a netfacs object

Usage

```
get_data(x, condition = "all")
```

Arguments

x	extract data from the test condition of a netfacs object
condition	one of "all" (default), "test" or "null".

is.netfacs *Checks if argument is a* netfacs *object*

Description

Checks if argument is a netfacs object

Usage

```
is.netfacs(x)
```

Arguments

x	An R object

is.netfacs_multiple *Checks if argument is a* netfacs_multiple *object*

Description

Checks if argument is a netfacs_multiple object

Usage

```
is.netfacs_multiple(x)
```

Arguments

x	An R object

is.netfacs_specificity *Checks if argument is a* netfacs_specificity *object*

Description

Checks if argument is a netfacs_specificity object

Usage

```
is.netfacs_specificity(x)
```

Arguments

x	An R object

letternet *Letter Data*

Description

Data from the German, English, and French Versions of The Communist Manifesto, to have large datasets to test different functions in this package for now

Usage

```
data(letternet)
```

Format

An object of class.

References

Marx & Engels, 'The Communist Manifesto'

multiple.netfacs *(Deprecated) Applies the* netfacs *function across multiple levels of the condition and puts them in a list*

Description

This function is deprecated. Please see `netfacs_multiple` instead

Usage

```
multiple.netfacs(
  data,
  condition = NULL,
  duration = NULL,
  ran.trials = 1000,
  control = NULL,
  random.level = NULL,
  combination.size = NULL,
  tail = "upper.tail",
  use_parallel = TRUE,
  n_cores = 2
)
```

Arguments

data	A binary matrix with one column per element, and one row per event, consisting of 1 (element was active during that event) and 0 (element was not active).
condition	character vector of same length as 'data' that contains information on the condition each event belongs to, so probabilities can be compared across conditions
duration	A numeric vector that contains information on the duration of each event; if NULL, all events are assumed to have equal duration.
ran.trials	Number of randomisations that will be performed to find the null distribution.
control	A list of vectors that are used as control variables. During bootstraps, the ratio of events in each level will be adapted. So, for example, if in the test distribution, there are three angry participants for each happy participant, the null distribution will maintain that ratio.
random.level	A character vector of the level on which the randomization should take place. If NULL, the randomization takes place on the event level (i.e., every row can either be selected or not); if a vector is provided, the randomization takes place on the levels of that vector rather than individual events.
combination.size	A positive integer, indicating the maximum combination size of element combinations. Higher numbers will increase computation time. Default is 2.
tail	Either 'upper.tail' (proportion of null probabilities that are larger than observed probabilities), or 'lower.tail' (proportion of null probabilities that are smaller than observed probabilities); default is 'upper.tail'.
use_parallel	Logical, indicating whether randomization or bootstrap should be parallelized (default is TRUE)
n_cores	Numeric, indicating the number cores to be used for parallelization. Default is 2.

Value

Function returns for each level of the condition a list equivalent to the results of the netfacs function; can be used to create multiple networks and graphs at the same time

---

multiple_netfacs_network

Creates network objects out of the netfacs data

Description

Takes the results of the nefacs object for combinations of 2 elements and turns them into a network object (class igraph and tbl_graph) that can be used for further plotting and analyses

Usage

```
multiple_netfacs_network(
  netfacs.list,
  link = "unweighted",
  significance = 0.01,
  min.count = 1,
  min.prob = 0,
  ignore.element = NULL
)

multiple.netfacs.network(
  netfacs.list,
  link = "unweighted",
  significance = 0.01,
  min.count = 1,
  min.prob = 0,
  ignore.element = NULL
)
```

Arguments

`netfacs.list`	list of multiple objects resulting from `netfacs` function or the `netfacs_multiple` function
`link`	determines how nodes/elements are connected. 'unweighted' gives a 1 to significant connections and 0 to all others; 'weighted' gives the difference between observed and expected probability of co-occurrence; 'raw' just uses the observed probability of co-occurrence; 'SRI' uses the simple ratio index/affinity (probability of co-occurrence/ (probabilities of each element and the combination))
`significance`	numeric value, determining the p-value below which combinations are considered to be dissimilar enough from the null distribution
`min.count`	numeric value, suggesting how many times a combination should at least occur to be displayed
`min.prob`	numeric value, suggesting the probability at which a combination should at least occur to be displayed
`ignore.element`	vector of elements that will not be considered for the network, e.g. because they are too common or too rare or their interpretation is not relevant here

Value

Function returns a network object where the nodes are the elements, edges represent their co-occurrence, and the vertex and edge attributes contain all additional information from the netfacs object

Examples

```
data(emotions_set)
emo.faces <- netfacs_multiple(
```

```
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  ran.trials = 10, # only for example
  combination.size = 2
)

emo.nets <- multiple_netfacs_network(emo.faces)
```

`multiple_network_plot` *Plots networks for multiple conditions*

Description

The function takes multiple network objects and plots them next to each other while keeping the element positions etc constant. Uses ggraph function

Usage

```
multiple_network_plot(netfacs.graphs, sig.level = 0.01, sig.nodes.only = FALSE)

multiple.network.plot(netfacs.graphs, sig.level = 0.01, sig.nodes.only = FALSE)
```

Arguments

`netfacs.graphs`	List of network objects resulting from `netfacs_multiple` function or `multiple_netfacs_network` function
`sig.level`	Numeric between 0 and 1. P value used to determine whether nodes are significant. Default = 0.01.
`sig.nodes.only`	Logical. Should only nodes that were significant in _at least_ one of the networks be included in the plots? Default = FALSE.

Value

Function returns a ggraph plot showing connections between nodes in the different networks. Elements that are significantly more likely to occur than expected are large, non-significant elements are small, and absent elements are absent.

Examples

```
data(emotions_set)
emo.faces <- netfacs_multiple(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  duration = NULL,
  ran.trials = 10, # only for example
  control = NULL,
  random.level = NULL,
  combination.size = 2
```

```
)

emo.nets <- multiple_netfacs_network(emo.faces, min.count = 5)
multiple_network_plot(emo.nets)
```

`mutual.information` *Calculates the pointwise mutual information of units with each other*

Description

Calculates the pointwise mutual information of units with each other

Usage

```
mutual.information(netfacs.data)
```

Arguments

`netfacs.data`	object resulting from netfacs() function

Value

Function returns a dataframe that includes all combinations, their occurrence counts and probabilities, and the pointwise mutual information (standardised between -1 and 1). 1 means seeing one necessitates seeing the other, -1 means one precludes the other

Examples

```
### how do angry facial expressions differ from non-angry ones?

data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = NULL,
  test.condition = NULL,
  ran.trials = 100,
  combination.size = 4
)

mutual.information(angry.face)
```

`mutual.information.condition` *Tests how much each element increases the specificity of all combinations it is in*

Description

The function takes all elements and dyadic combinations of elements in a netfacs object, goes through all combinations these elements are in, and compares the specificity (strength with which the combination identifies the test condition) of all combinations with the element and the same combinations without the element, to test how much specificity the element adds when added to a signal. Only works for netfacs objects based on comparison between conditions.

Usage

```
mutual.information.condition(netfacs.data)
```

Arguments

`netfacs.data`	object resulting from netfacs() function

Value

Function returns a list with two data frames that include all elements and first-order combinations that occur at all, the number of combinations that each element/combination is part of, and how much adding this element to a combination adds on average to its specificity, and how often it occurs

Examples

```
### how do angry facial expressions differ from non-angry ones?

data(emotions_set)
angry.face <- netfacs(
 data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  null.condition = NULL,
  ran.trials = 100,
  combination.size = 4
)

head(mutual.information.condition(angry.face), 20)
```

netfacs *Create probability distribution of combinations of elements in the data*

Description

The netfacs function underlies most other functions in this package.
It takes the data set and reports the observed and expected probabilities that elements and combinations of elements occur in this data set, and whether this differs from a null condition.

Usage

```
netfacs(
  data,
  condition = NULL,
  test.condition = NULL,
  null.condition = NULL,
  duration = NULL,
  ran.trials = 1000,
  control = NULL,
  random.level = NULL,
  combination.size = 2,
  tail = "upper.tail",
  use_parallel = TRUE,
  n_cores = 2
)
```

Arguments

data	A binary matrix with one column per element, and one row per event, consisting of 1 (element was active during that event) and 0 (element was not active).
condition	A character vector the same length as 'data' that contains information on the condition each event belongs to, so probabilities can be compared across conditions; if NULL, all events will be tested against a random null condition based on permutations.
test.condition	A string, indicating the level of 'condition' that is supposed to be tested.
null.condition	A string, indicating the level of 'condition' that is used to create the null distribution of values; if NULL, all levels that are not the test condition will be used.
duration	A numeric vector that contains information on the duration of each event; if NULL, all events are assumed to have equal duration.
ran.trials	Number of randomisations that will be performed to find the null distribution.
control	A list of vectors that are used as control variables. During bootstraps, the ratio of events in each level will be adapted. So, for example, if in the test distribution, there are three angry participants for each happy participant, the null distribution will maintain that ratio.

`random.level` A character vector of the level on which the randomization should take place. If NULL, the randomization takes place on the event level (i.e., every row can either be selected or not); if a vector is provided, the randomization takes place on the levels of that vector rather than individual events.

`combination.size` A positive integer, indicating the maximum combination size of element combinations. Higher numbers will increase computation time. Default is 2.

`tail` Either 'upper.tail' (proportion of null probabilities that are larger than observed probabilities), or 'lower.tail' (proportion of null probabilities that are smaller than observed probabilities); default is 'upper.tail'.

`use_parallel` Logical, indicating whether randomization or bootstrap should be parallelized (default is `TRUE`)

`n_cores` Numeric, indicating the number cores to be used for parallelization. Default is 2.

Details

If the 'condition' and 'test.condition' arguments are specified, the null distribution of probability values are based on bootstraps of the null condition. If the 'condition' argument is not specified, the null distribution is based on random permutations of the data.

For a general overview on how to use the netfacs function and package see `vignette("netfacs_tutorial")`.

Value

An object of class `netfacs`, which contains the probabilities of observing element combinations in the data, along with other useful information. The resulting object is the basis for most other functions in this package.

Author(s)

Alex Mielke, Alan V. Rincon

References

Mielke, A., Waller, B. M., Perez, C., Rincon, A. V., Duboscq, J., & Micheletta, J. (2021). NetFACS: Using network science to understand facial communication systems. *Behavior Research Methods*. https://doi.org/10.3758/s13428-021-01692-5

See Also

`netfacs_multiple`, `netfacs_extract`, `conditional_probabilities`

Examples

```
### how do angry facial expressions differ from non-angry ones?

data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
```

```
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  null.condition = NULL,
  duration = NULL,
  ran.trials = 100,
  control = NULL,
  random.level = NULL,
  combination.size = 5,
  tail = "upper.tail",
  use_parallel = TRUE,
  n_cores = 2
)

head(angry.face$result, 20)
angry.face$event.size.information
```

`netfacs.reciprocity` *Calculate reciprocity of probabilities that two elements appear together*

Description

For all dyadic combinations that ever appear, this function calculates how reciprocal the conditional probabilities (i.e. probability of A given B, and B given A) of the two elements are. Combinations that are highly reciprocal indicate that the two elements always occur together and might represent a fixed combination, while low reciprocity might indicate that one element is an extension of the other. Values approaching -1 indicate that one element is strongly dependent on the other, but this is not reciprocated; values around 0 indicate that neither is conditional on the other; and values approaching 1 indicate that both values are conditional on each other. If P[A|B] is the larger conditional probability, the reciprocity is calculated as reciprocity = ((P[B|A]/P[A|B]) - (P[A|B] - P[B|A])) * P[A|B].

Usage

```
netfacs.reciprocity(netfacs.data)
```

Arguments

`netfacs.data`	object resulting from netfacs() function

Value

Function returns a data frame with each combination, the reciprocity of conditional occurrence from -1 (one element entirely depends on the other, but not vice versa) to 1 (both elements always occur together)

The directions and conditional probabilities of both elements are also returned

Examples

```
### how do angry facial expressions differ from non-angry ones?
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 100,
  combination.size = 2
)

netfacs.reciprocity(angry.face)
```

---

netfacs_bootstrap *Calculate expected probability from single bootstrap*

Description

Calculate expected probability from single bootstrap

Usage

```
netfacs_bootstrap(
  subject,
  subject.weight,
  null.subjects,
  null.elements,
  test.combinations,
  max.combination.size,
  max.event.size
)
```

Arguments

`subject`	A character vector of unique subjects present in the data
`subject.weight`	A numeric vector of weights to be used when sampling subjects
`null.subjects`	A denoting the subject of null.elements
`null.elements`	A list of active elements in the null condition
`test.combinations`	A vector denoting AU combinations that are present in the test data
`max.combination.size`	A positive integer indicating the maximum AU combination size considered in the bootstrap
`max.event.size`	A positive integer indicating the maximum event size to be considered

Value

A list of bootstrapped probabilities for combinations and event sizes

`netfacs_extract` *Extract results from a netfacs object*

Description

Extract results from a `netfacs` object.

Usage

```
netfacs_extract(
  netfacs.data,
  combination.size = NULL,
  significance = 1,
  min.count = 0,
  min.prob = 0
)

netfacs.extract(
  netfacs.data,
  combination.size = NULL,
  significance = 1,
  min.count = 0,
  min.prob = 0
)
```

Arguments

`netfacs.data`	An object of class `netfacs`.
`combination.size`	Numeric, denoting the combination size(s) that should be extracted. If `NULL` (default), all combination sizes are returned.
`significance`	Numeric value between 0 and 1, determining the p-value below which combinations are considered to be dissimilar enough from the null distribution.
`min.count`	Numeric, denoting the minimum number of times an element combination occurred.
`min.prob`	Numeric value between 0 and 1, denoting the minimum probability an element combination occurred to be displayed.

Value

Function returns a `tibble` data.frame that contains the results of the `netfacs` object. By default, returns all results for all observed combinations, but can optionally pre-filter results.

Examples

```
### how do angry facial expressions differ from non-angry ones?
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 10,
  combination.size = 2
)

netfacs_extract(angry.face)
```

`netfacs_multiple` *Applies the* `netfacs` *function across multiple levels of the condition and puts them in a list*

Description

Take dataset and report observed and expected likelihood that elements and combinations of elements occur in this dataset, and whether this differs from a null condition. Expected values are based on bootstraps of null distribution, so the values represent distribution of element co-occurrence under null condition. The resulting object is the basis for most other functions in this package.

Usage

```
netfacs_multiple(
  data,
  condition,
  duration = NULL,
  ran.trials = 1000,
  control = NULL,
  random.level = NULL,
  combination.size = 2,
  tail = "upper.tail",
  use_parallel = TRUE,
  n_cores = 2
)
```

Arguments

`data`	A binary matrix with one column per element, and one row per event, consisting of 1 (element was active during that event) and 0 (element was not active).
`condition`	character vector of same length as 'data' that contains information on the condition each event belongs to, so probabilities can be compared across conditions
`duration`	A numeric vector that contains information on the duration of each event; if NULL, all events are assumed to have equal duration.

`ran.trials` Number of randomisations that will be performed to find the null distribution.

`control` A list of vectors that are used as control variables. During bootstraps, the ratio of events in each level will be adapted. So, for example, if in the test distribution, there are three angry participants for each happy participant, the null distribution will maintain that ratio.

`random.level` A character vector of the level on which the randomization should take place. If NULL, the randomization takes place on the event level (i.e., every row can either be selected or not); if a vector is provided, the randomization takes place on the levels of that vector rather than individual events.

`combination.size`
A positive integer, indicating the maximum combination size of element combinations. Higher numbers will increase computation time. Default is 2.

`tail` Either 'upper.tail' (proportion of null probabilities that are larger than observed probabilities), or 'lower.tail' (proportion of null probabilities that are smaller than observed probabilities); default is 'upper.tail'.

`use_parallel` Logical, indicating whether randomization or bootstrap should be parallelized (default is `TRUE`)

`n_cores` Numeric, indicating the number cores to be used for parallelization. Default is 2.

Value

An object of class `netfacs_multiple`, which contains the probabilities of observing element combinations in one condition vs. all other conditions, along with other useful information. The resulting object is the basis for most other functions in this package.

See Also

`netfacs`, `netfacs_extract`,

Examples

```
data(emotions_set)
emo.faces <- netfacs_multiple(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  ran.trials = 10, # only for example
  combination.size = 2
)

netfacs_extract(emo.faces)
```

`netfacs_network` *Creates a network object out of the netfacs data*

Description

Takes the results of the nefacs object for combinations of 2 elements and turns them into a network object (igraph) that can be used for further plotting and analyses

Usage

```
netfacs_network(
  netfacs.data,
  link = "unweighted",
  significance = 0.01,
  min.count = 1,
  min.prob = 0,
  ignore.element = NULL
)

netfacs.network(
  netfacs.data,
  link = "unweighted",
  significance = 0.01,
  min.count = 1,
  min.prob = 0,
  ignore.element = NULL
)
```

Arguments

`netfacs.data`	object resulting from `netfacs` function
`link`	determines how nodes/elements are connected. 'unweighted' gives a 1 to significant connections and 0 to all others; 'weighted' gives the difference between observed and expected probability of co-occurrence; 'raw' just uses the observed probability of co-occurrence
`significance`	numeric value, determining the p-value below which combinations are considered to be dissimilar enough from the null distribution
`min.count`	numeric value, suggesting how many times a combination should at least occur to be displayed
`min.prob`	numeric value, suggesting the probability at which a combination should at least occur to be displayed
`ignore.element`	vector of elements that will not be considered for the network, e.g. because they are too common or too rare or their interpretation is not relevant here

Value

Function returns a network object where the nodes are the elements, edges represent their co-occurrence, and the vertex and edge attributes contain all additional information from the netfacs object

Examples

```
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 100,
  combination.size = 2
)

anger.net <- netfacs_network(
  netfacs.data = angry.face,
  link = "unweighted",
  significance = 0.01,
  min.count = 1
)
```

---

netfacs_randomize *Calculate probabilities from single randomization*

Description

Calculate probabilities from single randomization

Usage

```
netfacs_randomize(m, test.combinations, max.combination.size, max.event.size)
```

Arguments

m	A numeric matrix
test.combinations	A vector of AU combinations observed in test data
max.combination.size	A positive integer
max.event.size	A Positive integer

Value

A list of randomized probabilities for combinations and event sizes

network.conditional *(Deprecated) Produce conditional probabilities of dyads of elements, and graph object based on conditional probabilities*

Description

This function is deprecated. Please see `network_conditional` instead

Usage

```
network.conditional(
  netfacs.data,
  min.prob = 0,
  min.count = 0,
  ignore.element = NULL,
  plot.bubbles = FALSE
)
```

Arguments

`netfacs.data`	object resulting from `netfacs` or `conditional_probabilities` functions.
`min.prob`	minimum conditional probability that should be shown in the graph
`min.count`	minimum number of times that a combination should occur before being included in the graph
`ignore.element`	string vector, can be used to exclude certain elements when creating the plots
`plot.bubbles`	if TRUE (default), then the nodes in the network plots will be surrounded by bubbles; if FALSE, the edges connect the names directly

Value

Function returns a dataframe that includes all dyadic combinations and their observed and conditional probabilities

network_conditional *Create a network based on conditional probabilities of dyads of elements*

Description

This is a convenience function to create and visualize a network of conditional probabilities for all dyadic element combinations of a `netfacs` object. Conditional probabilities are calculated using the `conditional_probabilities` function.

Usage

```
network_conditional(
  netfacs.data,
  min.prob = 0,
  min.count = 0,
  ignore.element = NULL,
  plot.bubbles = TRUE
)
```

Arguments

netfacs.data	object resulting from netfacs or conditional_probabilities functions.
min.prob	minimum conditional probability that should be shown in the graph
min.count	minimum number of times that a combination should occur before being included in the graph
ignore.element	string vector, can be used to exclude certain elements when creating the plots
plot.bubbles	if TRUE (default), then the nodes in the network plots will be surrounded by bubbles; if FALSE, the edges connect the names directly

Value

Function returns named list that includes a tbl_graph network and a ggraph plot.

See Also

netfacs, conditional_probabilities

Examples

```
### how do angry facial expressions differ from non-angry ones?
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 100,
  combination.size = 2
)

conditional.probs <- conditional_probabilities(angry.face)

network_conditional(
  netfacs.data = conditional.probs,
  min.prob = 0.01,
  min.count = 3,
  ignore.element = "25",
  plot.bubbles = FALSE
)
```

network_plot *Plots a network object*

Description

Plots the network created using the netfacs_network function; for networks with clear clusterin of elements, clusters can get different colours

Usage

```
network_plot(
  netfacs.graph,
  title = "network",
  clusters = FALSE,
  plot.bubbles = FALSE,
  hide.unconnected = TRUE
)

network.plot(
  netfacs.graph,
  title = "network",
  clusters = FALSE,
  plot.bubbles = FALSE,
  hide.unconnected = TRUE
)
```

Arguments

netfacs.graph	igraph network object resulting from netfacs_network
title	string of the graph's main title
clusters	if TRUE, cluster_fast_greedy is used to establish possible clusters in the dataset
plot.bubbles	if TRUE, then the nodes in the network plots will be surrounded by bubbles; if FALSE, the edges connect the names directly
hide.unconnected	if TRUE, then the nodes that do not have any significant connections will be hidden in the plot

Value

Function returns a ggraph plot of the network, where the size of nodes indicates how often they occur on their own, and edges indicate significant co-occurrence between them

Examples

```
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 10,
  combination.size = 2
)

anger.net <- netfacs_network(
  netfacs.data = angry.face,
  link = "unweighted",
  significance = 0.01,
  min.count = 1
)

network_plot(anger.net,
             title = "Angry Faces",
             clusters = FALSE,
             plot.bubbles = TRUE)
```

`network_summary` *Returns all kinds of network measures for the netfacs network*

Description

Calculates node level centrality measures from the network object

Usage

```
network_summary(netfacs.graph)

network.summary(netfacs.graph)
```

Arguments

`netfacs.graph`	igraph network object resulting from netfacs_network function

Value

Function returns a data frame with the element, its 'strength' (mean probability of co-occurrence), 'eigenvector' centrality (connection to other highly connected elements), 'betweenness' centrality (number of connections running through the element), and a number of other network measures

Examples

```
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 10,
  combination.size = 2
)

anger.net <- netfacs_network(
  netfacs.data = angry.face,
  link = "unweighted",
  significance = 0.01,
  min.count = 1
)

network_summary(anger.net)
```

---

network_summary_graph *Returns all kinds of graph-level network measures for the netfacs network*

Description

Calculates graph level summary measures from the network object

Usage

```
network_summary_graph(netfacs.net)

network.summary.graph(netfacs.net)
```

Arguments

netfacs.net	igraph network object resulting from netfacs_network function

Value

Function returns a dataframe with the number of elements in the graph, the number of connected edges, mean strength of connections, transitivity (mean number of closed triads), diameter (furthest path between two elements), degree centralization, and mean distance between elements

Examples

```
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
```

```
  test.condition = "anger",
  ran.trials = 10,
  combination.size = 2
)

anger.net <- netfacs_network(
  netfacs.data = angry.face,
  link = "unweighted",
  significance = 0.01,
  min.count = 1
)

network_summary_graph(anger.net)
```

`overlap.network` *(Deprecated) Plots the overlap of multiple conditions as bipartite network.*

Description

This function is deprecated. Please see `overlap_network` instead

Usage

```
overlap.network(
  netfacs.list,
  min.prob = 0,
  min.count = 5,
  significance = 0.01,
  specificity = 0.1,
  ignore.element = NULL,
  clusters = FALSE,
  plot.bubbles = FALSE
)
```

Arguments

`netfacs.list`	list of objects resulting from `netfacs` or `netfacs_multiple`
`min.prob`	minimum conditional probability that should be shown in the graph
`min.count`	minimum number of times that a combination should occur before being included in the graph
`significance`	sets the level of significance that combinations have to pass before added to the network
`specificity`	for the 'reduced' graph, select only elements that surpass this context specificity value
`ignore.element`	string vector, can be used to exclude certain elements when creating the plots

clusters	boolean; if TRUE, the cluster_fast_greedy algorithm is used to detect underlying community structure, based on the occurrence probability network
plot.bubbles	if TRUE, then the nodes in the network plots will be surrounded by bubbles; if FALSE, the edges connect the names directly

---

`overlap_network` *Plots the overlap of multiple conditions as bipartite network*

Description

The function takes multiple netfacs objects and plots how different elements connect the conditions, based on the conditional probabilities that the element occurs in the condition and that the condition is seen when the element is present

Usage

```
overlap_network(
  x,
  min.prob = 0,
  min.count = 5,
  significance = 0.01,
  specificity = 0.1,
  ignore.element = NULL,
  clusters = FALSE,
  plot.bubbles = TRUE
)
```

Arguments

x	list of objects resulting from specificity or netfacs
min.prob	minimum conditional probability that should be shown in the graph
min.count	minimum number of times that a combination should occur before being included in the graph
significance	sets the level of significance that combinations have to pass before added to the network
specificity	for the 'reduced' graph, select only elements that surpass this context specificity value
ignore.element	string vector, can be used to exclude certain elements when creating the plots
clusters	boolean; if TRUE, the cluster_fast_greedy algorithm is used to detect underlying community structure, based on the occurrence probability network
plot.bubbles	if TRUE, then the nodes in the network plots will be surrounded by bubbles; if FALSE, the edges connect the names directly

Value

Function returns a ggraph plot where each condition is connected to those elements that occur significantly in this condition, and each element is connected to each condition under which it occurs significantly more than expected. Creates four graphs: context specificity, occurrence in that context, a combined graph, and a 'reduced' graph where edges are only included if they pass the 'specificity' value set by the user

Examples

```
data(emotions_set)
emo.faces <- netfacs_multiple(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  ran.trials = 10,
  combination.size = 2
)
# calculate element specificity
spec <- specificity(emo.faces)

overlap <- overlap_network(spec,
                           min.prob = 0.01,
                           min.count = 3,
                           significance = 0.01,
                           specificity = 0.5,
                           ignore.element = "25",
                           clusters = TRUE,
                           plot.bubbles = TRUE)
```

---

possible_combinations *Calculate all possible combinations of elements*

Description

Takes a vector of elements and returns a vector with all possible combinations

Usage

```
possible_combinations(elements, maxlen, sep = "_")
```

Arguments

elements	A vector of elements
maxlen	maximum size of combinations to be considered
sep	String. Separator used for showing combinations of elements

Value

A vector with all element combinations

`prepare.netfacs` *Take data that are not currently in format and turn them into the correct format for netfacs function*

Description

The `netfacs` function requires data to be entered with the element data as a matrix of each element by each event, with occurrence marked as 1 and non-occurrence marked as 0.
This is often not the case, so this function transforms data in other routine formats to have the right look.
Specifically, users can define whether they want to enter 'photos', which indicates that all elements in an event are simply strung together in a vector; or they define 'video', in which case it is assumed that each element has a start and an end point in a specified video

Usage

```
prepare.netfacs(
  elements,
  type = c("video", "photo"),
  video.id = NULL,
  start.time = NULL,
  duration = NULL,
  separator = ",",
  frame.duration = NULL
)
```

Arguments

`elements`	vector with either one element per index (for videos) or all elements that occurred in the whole event (for photos)
`type`	either 'video' or 'photo'. If 'photo', the function separates the string and returns a matrix of the correct dimensions. If 'video', the function creates a matrix using the highest common factor of all 'durations' and for each of those 'frames' assigns whether each element was present or absent
`video.id`	name of the video, so all cases are treated together. For photos, can be entered so that photos can be matched to IDs after
`start.time`	for videos, time when the element is first active
`duration`	for videos, how long is the element active for
`separator`	for photos, how are elements separated in the list
`frame.duration`	for videos, how long is a 'frame' supposed to last? If NULL, frame duration is the shortest 'duration' of any element specified

Details

The assumption for this function is that for photos, elements are stored like this:
'AU1/AU2/AU3/AU4'
'AU1/AU3/AU4'
'AU1/AU2'

For videos, the assumption is that they are stored in a data frame like this:
element = AU1, video.id = 1, start.time = 0.5, duration = 2sec

Value

Function returns a list with element.matrix (the matrix of elements and when they occurred) and video.info (the supporting information, e.g. video names, durations, frames etc)

Examples

```
# for a photo
au.photos <- c(
  "AU1/AU5/AU9",
  "AU1/AU2",
  "AU1/AU2/AU10",
  "AU1/AU2",
  "AU5/AU17/AU18",
  "AU6/AU12"
)
au.names <- c("photo1", "photo2", "photo3", "photo4", "photo5", "photo6")
au.prepared <- prepare.netfacs(
  elements = au.photos,
  type = "photo",
  video.id = au.names,
  separator = "/"
)
au.prepared$element.matrix
au.prepared$video.info

# for a video
aus <- c(
  "AU1", "AU5", "AU9",
  "AU1", "AU2",
  "AU1", "AU2", "AU10",
  "AU1", "AU2",
  "AU5", "AU17", "AU18",
  "AU6", "AU12"
)
video.names <- c(
  rep("video1", 3),
  rep("video2", 2),
  rep("video3", 3),
  rep("video4", 2),
  rep("video5", 3),
  rep("video6", 2)
```

```
)
start.times <- c(
  0.1, 0.2, 0.3,
  0.1, 0.3,
  0.1, 0.4, 0.4,
  0.1, 0.2,
  0.1, 0.5, 0.6,
  0.1, 0.2
)
durations <- rep(0.3, times = length(start.times))
frame.dur <- 0.05
au.prepared <- prepare.netfacs(
  elements = aus,
  type = "video",
  video.id = video.names,
  start.time = start.times,
  duration = durations,
  frame.duration = frame.dur
)
head(au.prepared$element.matrix)
head(au.prepared$video.info)
```

`print.netfacs` *Print method for objects of class* `netfacs`

Description

Print method for objects of class `netfacs`

Usage

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

Arguments

`x`	An object of class `netfacs`
`...`	Additional arguments that would be passed to or from other methods

---

`print.netfacs_multiple` *Print method for objects of class* `netfacs_multiple`

Description

Print method for objects of class `netfacs_multiple`

Usage

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

Arguments

x	An object of class netfacs_multiple
...	Additional arguments that would be passed to or from other methods

---

probability_of_combination

Calculate probabilities of single elements and combinations occurring

---

Description

Calculate probabilities of single elements and combinations occurring

Usage

```
probability_of_combination(elements, maxlen, sep = "_")
```

Arguments

elements	A vector with all elements observed together at an event. Or a list of vectors or a binary matrix with elements as colnames()
maxlen	maximum size of combinations to be considered
sep	String. Separator used for showing combinations of elements

Value

Function returns a dataframe with observed probabilities for each combination in the dataset

---

probability_of_event_size

Count number of event sizes

Description

Count number of event sizes

Usage

```
probability_of_event_size(elements, max.event.size)
```

Arguments

elements	A list of vectors containing active elements or a binary matrix with events in rows
max.event.size	A positive integer

Value

A named vector, including probabilities for event sizes that were not observed in the data

---

sim_facs *Simulate FACS data*

Description

Simulate FACS data

Usage

```
sim_facs(m, n_obs = 10, jp = NULL)
```

Arguments

m	A matrix with condition asrownames, elements as colnames, and probabilities of observing an element as values.
n_obs	Number of observations per condition to simulate
jp	An optional list of matrices, the same length as nrow(m) with the joint probabilities of elements

Examples

```
elements <- as.character(1:10)
conditions <- letters[1:2]
# randomly generate probability of elements
probabilities <-
  sapply(elements, function(x) {
    p <- runif(length(conditions))
    setNames(round(p, 1), nm = conditions)
  })
sim_facs(probabilities)
```

specificity *Specificity*

Description

Calculate specificity of element combinations to a given condition

Usage

```
specificity(
  x,
  condition,
  test.condition = NULL,
  null.condition = NULL,
  combination.size = NULL,
  upsample = TRUE
)
```

Arguments

x	A binary matrix, with AUs as colnames, or an object of class netfacs
condition	A character condition vector
test.condition	A string, denoting the test condition. If NULL (default) specificity is calculated for all conditions.
null.condition	A string, denoting the null condition. If NULL (default) all observations not part of the test.condition will be considered part of the null.
combination.size	A positive integer, indicating the maximum combination size of element combinations. If NULL (default), the maximum combination size observed in the x is used.
upsample	Logical. Should minority condition(s) be upsampled? TRUE by default.

Details

Specificity values are biased when the number of observations per condition is highly imbalanced. When upsample = TRUE (recommended), the condition(s) with fewer observations are randomly upsampled to match the number of observations in the most common condition prior to the specificity calculation. This procedure minimizes the bias in the specificity results.

Value

A data frame

Examples

```
specificity(
  x = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger"
)
```

`specificity_increase` *Tests how much each element increases the specificity of all combinations it is in*

Description

The function takes all elements and dyadic combinations of elements in a netfacs object, goes through all combinations these elements are in, and compares the specificity (strength with which the combination identifies the test condition) of all combinations with the element and the same combinations without the element, to test how much specificity the element adds when added to a signal. Only works for netfacs objects based on comparison between conditions.

Usage

```
specificity_increase(x)
```

Arguments

x	object resulting from `specificity` function

Value

Function returns a list with two data frames that include all elements and first-order combinations that occur at all, the number of combinations that each element/combination is part of, and how much adding this element to a combination adds on average to its specificity, and how often it occurs

Examples

```
### how do angry facial expressions differ from non-angry ones?
data(emotions_set)
angry.face <- netfacs(
  data = emotions_set[[1]],
  condition = emotions_set[[2]]$emotion,
  test.condition = "anger",
  ran.trials = 10,
  combination.size = 2
)

spec <- specificity(angry.face)
specificity_increase(spec)
```

summarise_combination *Summarise combination results from bootstrap*

Description

Summarise combination results from bootstrap

Usage

```
summarise_combination(
  combination,
  combination.size,
  observed.prob,
  boot.prob,
  tail,
  test.count
)
```

Arguments

combination	A vector of AU combinations
combination.size	A vector denoting the number of active AUs in combination
observed.prob	A vector with probability of combination in test data
boot.prob	A matrix with boot probabilities of a given combination in columns
tail	upper.tail or lower.tail,
test.count	Number of times a combination occurs in test dataset

Value

A dataframe

summarise_event_size *Summarise event size probabilities*

Description

Summarise event size probabilities

Usage

```
summarise_event_size(observed.prob, boot.prob)
```

Arguments

observed.prob	A named vector with probabilities of event sizes.
boot.prob	A matrix with boot probabilities of a given event size. Combination size in rows, trials in columns.

Value

A dataframe

---

upsample *Up sample*

Description

Randomly up-sample the minority condition(s) to have the same number of observations as the majority condition. Random samples are added to the existing observations of the minority conditions

Usage

```
upsample(x, condition, .name = "condition")
```

Arguments

x	A data.frame or something coercible to one
condition	A character vector the same length as 'x' denoting which condition each observation belongs to
.name	A string used to name the condition column

Value

A tibble

Examples

```
d <- data.frame(
  condition = c(rep("a", times = 7), rep("b", times = 3)),
  x = sample(0:1, size = 10, replace = TRUE),
  y = sample(0:1, size = 10, replace = TRUE)
)

upsample(x = d, condition = d$condition)
```

validate_condition *Check that condition arguments are formatted correctly*

Description

Check that condition arguments are formatted correctly

Usage

```
validate_condition(data, condition, test.condition, null.condition)
```

Arguments

data	data passed by the user
condition	condition passed by the user
test.condition	condition passed by the user
null.condition	condition passed by the user

validate_data *Check that 'data' argument is formatted correctly*

Description

Check that 'data' argument is formatted correctly

Usage

```
validate_data(data)
```

Arguments

data	data passed by the user

Value

data as a matrix

Index