

Package ‘NetFACS’

June 8, 2022

Title Network Applications to Facial Communication Data

Version 0.4.0

Date 2022-05-09

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](https://doi.org/10.1111/2041-210X.12772)>;

Carsey, T., & Harden, J. (2014) <[doi:10.4135/9781483319605](https://doi.org/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

RoxygenNote 7.1.2

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 2022-06-08 14:40:11 UTC

R topics documented:

add_inactive_single_units	3
calculate_combination_size	3
define_contexts	4
define_joint_prob	4
distribution.plot	5
element.plot	5
element.specificity	6
emotions_set	7
entropy.overall	8
equal_observations	8
event.size.plot	9
get_active_elements	10
letternet	10
multiple.netfacs	11
multiple.netfacs.network	12
multiple.network.plot	13
mutual.information	14
mutual.information.condition	15
netfacs	16
netfacs.extract	18
netfacs.network	19
netfacs.reciprocity	20
netfacs_bootstrap	21
netfacs_extract	22
netfacs_multiple	23
netfacs_randomize	25
network.conditional	25
network.plot	26
network.summary	28
network.summary.graph	29
overlap.network	30
prepare.netfacs	31
print.netfacs	33
print.netfacs_multiple	34
probability_of_combination	34
probability_of_event_size	35
sample_contexts	35
summarise_combination	36
summarise_event_size	36

`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

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	<i>Plots the observed probability for an element against the distribution of the null model</i>
-------------------	---

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	<i>Plots the observed and expected probabilities for the basic elements based on the condition</i>
--------------	--

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 *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

```
element.specificity(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
)

element.specificity(angry.face)$element
```

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	<i>Compares the observed and expected information content of the dataset</i>
-----------------	--

Description

Establishes how 'ordered' the data is: values close to 0 indicate that combinations are highly repetitive and predictable, while values close to 1 indicate that combinations are equiprobable and prediction of future combinations is difficult

Usage

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

Arguments

netfacs.data object resulting from netfacs() function

Value

Function returns the ratio of observed entropy/expected entropy. Expected entropy is based on randomization (shuffling the observed elements while maintaining the number of elements per row) and represents the maximum entropy 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.

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	<i>Check that ALL objects have the same number of observations</i>
--------------------	--

Description

```
length(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	<i>Plots the probability that a combination of a certain size appears</i>
-----------------	---

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)
```

Arguments

netfacs.data object resulting from netfacs() function

Value

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

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

`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	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	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	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	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	if not all combinations of elements are of interest (e.g., if the question only concerns single elements or dyads of elements), this variable allows to reduce the results to those combinations, increasing speed

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, should the bootstrap be parallelized (default is TRUE)
n_cores	numeric, the number cores to be used for parallelization. Default is the number of available cores minus 1.

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
)
```

Arguments

netfacs.list	list of multiple objects resulting from <code>netfacs</code> function or the <code>netfacs_multiple</code> 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)
```

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

<code>data</code>	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)
<code>condition</code>	character vector of 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
<code>test.condition</code>	level of 'condition' that is supposed to be tested
<code>null.condition</code>	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
<code>duration</code>	numeric vector that contains information on the duration of each event; if NULL, all events are assumed to have equal duration
<code>ran.trials</code>	Number of randomisations that will be performed to find the null distribution
<code>control</code>	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

<code>random.level</code>	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
<code>combination.size</code>	if not all combinations of elements are of interest (e.g., if the question only concerns single elements or dyads of elements), this variable allows to reduce the results to those combinations, increasing speed
<code>tail</code>	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'
<code>use_parallel</code>	logical, should the bootstrap be parallelized (default is TRUE)
<code>n_cores</code>	numeric, the number cores to be used for parallelization. Default is the number of available cores minus 1.

Details

Expected values are based on bootstraps of null distribution, so the values represent distribution of element co-occurrence under null condition; or permutations of the observed distribution to test it against 'random'.

The resulting object is the basis for most other functions in this package.

Value

Function returns a Result data frame that includes the combination name, how many elements it consisted of, how often it was observed, the probability it was observed under this condition, the expected probability under null condition (based on the permutation or bootstrap), effect size (difference between observed probability and expected probability), p-value (how many randomisations were more extreme), and for direct comparisons of contexts the specificity (probability that the condition is in fact the test condition if that combination is known) and probability increase (the factor by which the probability of the element is higher in the test than null condition)

'event.size.information' contains information about the observed and expected size of combination or elements per event based on the randomisations

Author(s)

Alex Mielke, Alan V. Rincon

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.extract *(Deprecated) Extract results from a netfacs object.*

Description

This function is deprecated. Please see [netfacs_extract](#) instead

Usage

```

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

```

Arguments

netfacs.data	object resulting from netfacs function.
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 value, suggesting how many times a combination should at least occur to be displayed.
min.prob	numeric value between 0 and 1, suggesting the probability at which a combination should at least occur to be displayed.
min.specificity	numeric value between 0 and 1, suggesting the specificity a combination should at least have for the test condition to be displayed.
level	deprecated. Please use combination.size instead.

Value

Function returns a dataframe that contains the results of the `netfacs` object. By default, returns all results for all observed combinations

<code>netfacs.network</code>	<i>Creates a network object out of the netfacs data</i>
------------------------------	---

Description

Takes the results of the `netfacs` object for combinations of 2 elements and turns them into a network object (igraph or sna/network) 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,  
  min.specificity = 0,  
  ignore.element = NULL  
)
```

Arguments

<code>netfacs.data</code>	object resulting from <code>netfacs</code> function
<code>link</code>	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
<code>significance</code>	numeric value, determining the p-value below which combinations are considered to be dissimilar enough from the null distribution
<code>min.count</code>	numeric value, suggesting how many times a combination should at least occur to be displayed
<code>min.prob</code>	numeric value, suggesting the probability at which a combination should at least occur to be displayed
<code>min.specificity</code>	numeric value, suggesting the specificity a combination should at least have for the test condition to be displayed
<code>ignore.element</code>	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,
  min.prob = 0,
  min.specificity = 0,
  ignore.element = NULL
)
```

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 $\text{reciprocity} = ((P[B|A]/P[A|B]) - (P[A|B] - P[B|A])) * P[A|B]$.

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

<code>netfacs_extract</code>	<i>Extract results from a netfacs object</i>
------------------------------	--

Description

Extract results from a `netfacs` object.

Usage

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

Arguments

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

`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 value, suggesting how many times a combination should at least occur to be displayed.

`min.prob` numeric value between 0 and 1, suggesting the probability at which a combination should at least occur to be displayed.

`min.specificity` numeric value between 0 and 1, suggesting the specificity a combination should at least have for the test condition to be displayed.

Value

Function returns a dataframe that contains the results of the `netfacs` object. By default, returns all results for all observed combinations

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_extract(angry.face,
  combination.size = 2,
  significance = 0.01,
  min.count = 5,
  min.prob = 0.01,
  min.specificity = 0.5
)
```

netfacs_multiple	<i>Applies the <code>netfacs</code> function across multiple levels of the condition and puts them in a list</i>
------------------	--

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

<code>data</code>	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)
<code>condition</code>	character vector of same length as 'data' that contains information on the condition each event belongs to, so probabilities can be compared across conditions
<code>duration</code>	numeric vector that contains information on the duration of each event; if NULL, all events are assumed to have equal duration
<code>ran.trials</code>	Number of randomisations that will be performed to find the null distribution
<code>control</code>	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
<code>random.level</code>	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
<code>combination.size</code>	if not all combinations of elements are of interest (e.g., if the question only concerns single elements or dyads of elements), this variable allows to reduce the results to those combinations, increasing speed
<code>tail</code>	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'
<code>use_parallel</code>	logical, should the bootstrap be parallelized (default is TRUE)
<code>n_cores</code>	numeric, the number cores to be used for parallelization. Default is the number of available cores minus 1.

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

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
)

head(emo.faces$anger$result, 5)
head(emo.faces$happy$result, 5)
```

netfacs_randomize	<i>Calculate probabilities from single randomization</i>
-------------------	--

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	<i>Produce conditional probabilities of dyads of elements, and graph object based on conditional probabilities</i>
---------------------	--

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 ($P(A+B)$), and the probability of A occurring if B is given ($P(A|B)$). It also creates a graph object that can be plotted

Usage

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

Arguments

netfacs.data	object resulting from netfacs() function
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, 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

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.net <- network.conditional(
  netfacs.data = angry.face,
  min.prob = 0.01,
  min.count = 3,
  ignore.element = "25",
  plot.bubbles = FALSE
)

conditional.net$conditional.probalities
```

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 = TRUE,  
  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 = 100,  
  combination.size = 2  
)
```

```
anger.net <- netfacs.network(  
  netfacs.data = angry.face,  
  link = "unweighted",  
  significance = 0.01,  
  min.count = 1,  
  min.prob = 0,  
  min.specificity = 0,  
  ignore.element = NULL  
)
```

```
anger.plot <- 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)
```

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 = 100,  
  combination.size = 2  
)
```

```
anger.net <- netfacs.network(  
  netfacs.data = angry.face,  
  link = "unweighted",  
  significance = 0.01,  
  min.count = 1,  
  min.prob = 0,  
  min.specificity = 0,  
  ignore.element = NULL  
)
```

```
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)
```

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 = 100,
  combination.size = 2
)
```

```
anger.net <- netfacs.network(
  netfacs.data = angry.face,
  link = "unweighted",
  significance = 0.01,
  min.count = 1,
  min.prob = 0,
  min.specificity = 0,
  ignore.element = NULL
)
```

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

overlap.network	<i>Plots the overlap of multiple conditions as bipartite network</i>
-----------------	--

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(
  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

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
)

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

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)
```

Arguments

elements	list with vectors for all elements observed together at each event
maxlen	maximum size of combinations to be considered

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

sample_contexts *Sample observations of simulated FACS data*

Description

Sample observations of simulated FACS data

Usage

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

Arguments

m A matrix with context in rows, aus in cols, and probabilities as values => result of [define_contexts](#).

n_obs Number of observations per context

jp An optional list of matrices, the same length as nrow(m) with the joint probabilities of AUs

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,
  null.count = NULL
)
```

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
null.count	Number of times a combination occurs in null 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

Index

* datasets

emotions_set, 7
letternet, 10

add_inactive_single_units, 3

calculate_combination_size, 3
cluster_fast_greedy, 27

define_contexts, 4, 35
define_joint_prob, 4
distribution.plot, 5

element.plot, 5
element.specificity, 6
emotions_set, 7
entropy.overall, 8
equal_observations, 8
event.size.plot, 9

get_active_elements, 10
ggraph, 13, 27, 30

igraph, 12

letternet, 10

multiple.netfacs, 11
multiple.netfacs.network, 12, 13
multiple.network.plot, 13
mutual.information, 14
mutual.information.condition, 15

netfacs, 11, 12, 16, 16, 18, 19, 22, 23, 30, 31
netfacs.extract, 18
netfacs.network, 19, 26, 27
netfacs.reciprocity, 20
netfacs_bootstrap, 21
netfacs_extract, 18, 22
netfacs_multiple, 11–13, 23, 30
netfacs_randomize, 25

network.conditional, 25
network.plot, 26
network.summary, 28
network.summary.graph, 29

overlap.network, 30

prepare.netfacs, 31
print.netfacs, 33
print.netfacs_multiple, 34
probability_of_combination, 3, 34
probability_of_event_size, 35

sample_contexts, 35
summarise_combination, 36
summarise_event_size, 36

tbl_graph, 12