Package 'cultevo'

May 8, 2026

Title Tools, Measures and Statistical Tests for Cultural Evolution

Version 1.0.2

Date 2018-04-24

Description Provides tools for measuring the compositionality of signalling systems (in particular the information-theoretic measure due to Spike (2016) <http://hdl.handle.net/1842/25930> and the Mantel test for distance matrix correlation (after Dietz 1983) <doi:10.1093/sysbio/32.1.21>), functions for computing string and meaning distance matrices as well as an implementation of the Page test for monotonicity of ranks (Page 1963) <doi:10.1080/01621459.1963.10500843> with exact p-values up to k = 22.

URL https://kevinstadler.github.io/cultevo/

BugReports https://github.com/kevinstadler/cultevo/issues

Encoding UTF-8

License MIT + file LICENSE

Imports combinat, grDevices, graphics, Hmisc, pspearman, stats, stringi, utils

Suggests memoise, knitr, rmarkdown

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Kevin Stadler [aut, cre]

Maintainer Kevin Stadler <a00425926@unet.univie.ac.at>

Repository CRAN

Date/Publication 2018-04-24 10:28:19 UTC

Contents

binaryfeaturematrix 2
check.dist 3
count.substring.occurrences 4

enumerate.meaningcombinations 4
enumerate.substrings 5
hammingdists 6
mantel.test 7
normalisedlevenshteindists 11
orderinsensitivedists 11
page.test 12
read.dist 14
repmatrix 14
segment.string 15
shuffle.locations 16
sm.compositionality 16
ssm.compositionality 21
temperature.colors 22
wrap.meaningdistfunction 23

Index **24**

`binaryfeaturematrix` *Convert a meaning matrix to a binary 'meaning-feature-present' matrix.*

Description

Transforms a meaning matrix to 'wide' format where, instead of having a column for every meaning dimension store all possible meaning values, every possible value for any dimension is treated as its own categorical 'meaning feature' whose presence or absence is represented by a logical `TRUE/FALSE` value in its own meaning feature column.

Usage

```
binaryfeaturematrix(meanings, rownames = NULL)
```

Arguments

`meanings`	a matrix or data frame with meaning dimensions along columns and different meaning combinations along rows (such as created by `enumerate.meaningcombinations`).
`rownames`	optional character vector of the same length as the number of rows of `meanings`.

Details

Given a matrix or data frame with meaning dimensions along columns and different combinations of meaning feature values along rows, creates a a matrix with the same number of rows but with one column for every possible value for every meaning dimension.

All meaning dimensions and values are treated *categorically*, i.e. as factors with no gradual notion of meaning feature similarity, neither within nor across the original meaning dimensions. Information about which feature values correspond to which meaning dimensions is essentially discarded in

this representation, but could in principle be recovered through the patterns of (non)-co-occurrence of different meaning features.

In order for the resulting meaning columns to be interpretable, the column names of the result are of the structure `columnname=value`, based on the column names of the input meaning matrix (see Examples).

Value

A matrix of `TRUE`/`FALSE` values with as many rows as `meanings` and one column for every column-value combination in `meanings`.

Examples

```
enumerate.meaningcombinations(c(2, 2))
binaryfeaturematrix(enumerate.meaningcombinations(c(2, 2)))
```

---

check.dist *Check or fix a distance matrix.*

Description

Checks or fixes the given distance matrix specification and returns an equivalent, symmetric `matrix` object with 0s in the diagonal.

Usage

```
check.dist(x)
```

Arguments

`x`	an object (or list of objects) specifying a distance matrix

Details

If the argument is a matrix, check whether it is a valid specification of a distance matrix and return it, making it symmetric if it isn't already.

If the argument is a list, calls `check.dist` on every of its elements and returns a list of the results.

For all other object types, attempts to coerce the argument to a `dist` object and return the corresponding distance matrix (see above).

Value

a symmetric `matrix` object (or list of such objects) of the same dimension as `x`

See Also

`dist`

`count.substring.occurrences`
Count occurences of all possible substrings in one more strings.

Description

Count occurences of all possible substrings in one more strings.

Usage

```
count.substring.occurrences(strings, sortbylength = FALSE)
```

Arguments

`strings`	a list or vector of character sequences
`sortbylength`	logical indicating whether the substring columns should be ordered according to the (decreasing) length of the substrings. Default is to leave them in the original order in which they occur in the given strings.

Value

A matrix with the original strings along rows and all substrings of those strings along columns. The cell values indicate whether (and how many times) the substring is contained in each of the strings.

Examples

```
count.substring.occurrences(c("asd", "asdd", "foo"))
```

`enumerate.meaningcombinations`
Enumerate meaning combinations.

Description

Enumerates all possible combinations of meanings for a meaning space of the given dimensionality.

Usage

```
enumerate.meaningcombinations(dimensionality, uniquelabels = TRUE,
  offset = 0)
```

Arguments

`dimensionality`	either a) a vector of integers specifying the number of different possible values for every meaning dimension, or b) a list or other (potentially ragged) 2-dimensional data structure listing the possible meaning values for every dimension
`uniquelabels`	logical, determines whether the same integers can be reused across meaning dimensions or not. When `uniquelabels = FALSE`, the resulting matrix will be very reminiscent of tables listing all binary combinations of factors. Ignored when `dimensionality` specifies the meaning values
`offset`	a constant that is added to all meaning specifiers. Ignored when `dimensionality` specifies the meaning values

Details

The resulting matrix can be passed straight on to `hammingdists` and other meaning distance functions created by `wrap.meaningdistfunction`.

Value

A matrix that has as many columns as there are dimensions, with every row specifying one of the possible meaning combinations. The entries of the first dimension cycle slowest (see examples).

See Also

`hammingdists`

Examples

```
enumerate.meaningcombinations(c(2, 2))
enumerate.meaningcombinations(c(3, 4))
enumerate.meaningcombinations(c(2, 2, 2, 2))
enumerate.meaningcombinations(8) # trivial
enumerate.meaningcombinations(list(shape=c("square", "circle"), color=c("red", "blue")))
```

`enumerate.substrings` *Enumerate all substrings of a string.*

Description

Enumerate all substrings of a string.

Usage

```
enumerate.substrings(string)
```

Arguments

`string`	a character string

Value

a vector containing all substrings of the string (including duplicates)

Examples

```
enumerate.substrings("abccc")
```

`hammingdists` *Pairwise Hamming distances between matrix rows.*

Description

Returns a distance matrix giving all pairwise Hamming distances between the rows of its argument `meanings`, which can be a matrix, data frame or vector. Vectors are treated as matrices with a single column, so the distances in its return value can only be 0 or 1.

Usage

```
hammingdists(meanings)
```

Arguments

`meanings`	a matrix with the different dimensions encoded along columns, and all combinations of meanings specified along rows. The data type of the cells does not matter since distance is simply based on equality (with the exception of `NA` values, see below.

Details

This function behaves differently from calling `dist(meanings, method="manhattan")` in how `NA` values are treated: specifying a meaning component as `NA` allows you to *ignore* that dimension for the given row/meaning combinations, (instead of counting a difference between `NA` and another value as a distance of 1).

Value

A distance matrix of type `dist` with `n*(n-1)/2` rows/columns, where n is the number of rows in `meanings`.

See Also

`dist`

Examples

```
# a 2x2 design using strings
print(strings <- matrix(c("a1", "b1", "a1", "b2", "a2", "b1", "a2", "b2"),
  ncol=2, byrow=TRUE))
hammingdists(strings)

# a 2x3 design using integers
print(integers <- matrix(c(0, 0, 0, 1, 0, 2, 1, 0, 1, 1, 1, 2), ncol=2, byrow=TRUE))
hammingdists(integers)

# a 3x2 design using factors (ncol is always the number of dimensions)
print(factors <- data.frame(colour=c("red", "red", "green", "blue"),
                            animal=c("dog", "cat", "dog", "cat")))
hammingdists(factors)

# if some meaning dimension is not relevant for some combinations of
# meanings (e.g. optional arguments), specifying them as NA in the matrix
# will make them not be counted towards the hamming distance! in this
# example the value of the second dimension does not matter (and does not
# count towards the distance) when the the first dimension has value '1'
print(ignoredimension <- matrix(c(0, 0, 0, 1, 1, NA), ncol=2, byrow=TRUE))
hammingdists(ignoredimension)

# trivial case of a vector: first and last two elements are identical,
# otherwise a difference of one
hammingdists(c(0, 0, 1, 1))
```

`mantel.test` *Perform one or more Mantel permutation tests.*

Description

Perform correlation tests between pairs of distance matrices. The Mantel test is different from classical correlation tests (such as those implemented by `cor.test`) in that the null distribution (and significance level) are obtained through randomisation. The null distribution is generated by shuffling the locations (matrix rows and columns) of one of the matrices to calculate an empirical null distribution for the given data set.

Usage

```
mantel.test(x, y, ...)

## Default S3 method:
mantel.test(x, y, plot = FALSE, method = c("spearman",
  "kendall", "pearson"), trials = 9999, omitzerodistances = FALSE, ...)

## S3 method for class 'formula'
mantel.test(x, y, groups = NULL,
```

```
  stringdistfun = utils::adist, meaningdistfun = hammingdists, ...)

## S3 method for class 'list'
mantel.test(x, y, plot = FALSE, ...)

## S3 method for class 'mantel'
plot(x, xlab = "generation", ...)
```

Arguments

x	a formula, distance matrix, or list of distance matrices (see below)
y	a data frame, distance matrix, or list of distance matrices of the same length as x
...	further arguments which are passed on to the default method (in particular plot, method, trials and omitzerodistances)
plot	logical: immediately produce a plot of the test results (default: FALSE)
method	correlation coefficient to be computed. Passed on to cor, so one of "spearman", "kendall", or, inadvisable in the case of ties: "pearson". Following Dietz (1983), "spearman" is used as a default that is both powerful and robust across different distance measures.
trials	integer: maximum number of random permutations to be computed (see Details).
omitzerodistances	logical: if TRUE, the calculation of the correlation coefficient omits pairs of off-diagonal cells which contain a 0 in the *second* distance matrix argument. (For the formula interface, this is the matrix which specifies the meaning distances.)
groups	when x is a formula: column name by which the data in y is split into separate data sets to run several Mantel tests on
stringdistfun	when x is a formula: edit distance function used to compute the distance matrix from the specified string column. Supports any edit distance function that returns a distance matrix from a vector or list of character strings. Default is Levenshtein distance (adist), other options from this package include normalisedlevenshteindists() and orderinsensitivedists().
meaningdistfun	when x is a formula: meaning distance function used to compute the distance matrix from the specified meaning columns. Defaults to Hamming distances between meanings (hammingdists()), custom meaning functions can be created easily using wrap.meaningdistfunction().
xlab	the x axis label used when plotting the result of several Mantel tests next to each other

Details

If the number of possible permutations of the matrices is reasonably close to the number of permutations specified by the trials parameter, a deterministic enumeration of all the permutations will be carried out instead of random sampling: such a deterministic test will return an exact p-value.

plot() called on a data frame of class mantel plots a visualisation of the test results (in particular, the distribution of the permutated samples against the veridical correlation coefficient). If the

veridical correlation coefficient is plotted in blue it means that it was higher than all other coefficients generated by random permutations of the data. When the argument contains the result of more than one Mantel tests, a side-by-side boxplot visualisation shows the mean and standard deviation of the randomised samples (see examples). Additional parameters `...` to `plot()` are passed on to `plot.default`.

Value

A dataframe of class `mantel`, with one row per Mantel test carried out, containing the following columns:

`method` Character string: type of correlation coefficient used

`statistic` The veridical correlation coefficient between the entries in the two distance matrices

`rsample` A list of correlation coefficients calculated from the permutations of the input matrices

`mean` Average correlation coefficient produced by the permutations

`sd` Standard deviation of the sampled correlation coefficients

`p.value` Empirical p-value computed from the Mantel test: let `ngreater` be the number of correlation coefficients in `rsample` greater than or equal to `statistic`, then `p.value` is `(ngreater+1)/(length(rsample)+1`

`p.approx` The theoretical p-value that would correspond to the standard `z` score as calculated above.

`is.unique.max` Logical, `TRUE` iff the veridical correlation coefficient is greater than any of the coefficients calculated for the permutations. If this is true, then `p.value == 1 / (length(rsample)+1)`

Multiple `mantel` objects can easily be combined by calling `rbind(test1, test2, ...)`.

Methods (by class)

- `default`: Perform Mantel correlation test on two distance matrices. The distance matrices can either be of type `dist`, plain R matrices or any object that can be interpreted by `check.dist`. The order of the two matrices does not matter unless `omitzerodistances = TRUE`, in which case cells with a 0 in the *second* matrix are omitted from the calculation of the correlation coefficient. For consistency it is therefore recommended to always pass the string distance matrix first, meaning distance matrix second.
- `formula`: This function can be called with raw experimental result data frames, distance matrix calculation is taken care of internally. `x` is a formula of the type `s ~ m1 + m2 + ...` where `s` is the column name of the character strings in data frame or matrix `y`, while `m1` etc. are the column names specifying the different meaning dimensions. To calculate the respective distances, the function `stringdistfun` is applied to the strings, `meaningdistfun` to the meaning columns.
- `list`: When `x` is a list of distance matrices, and `y` is either a single distance matrix or a list of distance matrices the same length as `x`: runs a Mantel test for every pairwise combination of distance matrices in `x` and `y` and returns a `mantel` object with as many rows.

References

Dietz, E. J. 1983 "Permutation Tests for Association Between Two Distance Matrices." *Systematic Biology* 32 (1): 21-–26. https://doi.org/10.1093/sysbio/32.1.21.

North, B. V., D. Curtis and P. C. Sham. 2002 "A Note on the Calculation of Empirical P Values from Monte Carlo Procedures." *The American Journal of Human Genetics* 71 (2): 439-–41. https://doi.org/10.1086/341527.

See Also

cor, adist, hammingdists, normalisedlevenshteindists, orderinsensitivedists

Examples

```
# small distance matrix, Mantel test run deterministically
mantel.test(dist(1:7), dist(1:7))


## Not run:
# run test on smallest distance matrix which requires a random
# permutation test, and plot it
plot(mantel.test(dist(1:8), dist(1:8), method="kendall"))

## End(Not run)

## Not run:
# 2x2x2x2 design
mantel.test(hammingdists(enumerate.meaningcombinations(c(2, 2, 2, 2))),
  dist(1:16), plot=TRUE)

## End(Not run)

# using the formula interface in combination with a data frame:
print(data <- cbind(word=c("aa", "ab", "ba", "bb"),
  enumerate.meaningcombinations(c(2, 2))))

mantel.test(word ~ Var1 + Var2, data)

## Not run:
# pass a list of distance matrices as the first argument, but just one
# distance matrix as the second argument: this runs separate tests on
# the pairwise combinations of the first and second argument
result <- mantel.test(list(dist(1:8), dist(sample(8:1)), dist(runif(8))),
  hammingdists(enumerate.meaningcombinations(c(2, 2, 2))))

# print the result of the three independently run permutation tests
print(result)

# show the three test results in one plot
plot(result, xlab="group")

## End(Not run)
```

normalisedlevenshteindists *Compute the normalised Levenshtein distances between strings.*

Description

Compute the normalised Levenshtein distances between strings.

Usage

```
normalisedlevenshteindists(strings)
```

Arguments

strings	a vector or list of strings

Value

A distance matrix specifying all pairwise normalised Levenshtein distances between the strings.

See Also

dist

Examples

```
normalisedlevenshteindists(c("abd", "absolute", "asdasd", "casd"))
```

orderinsensitivedists *Calculate the bag-of-characters similarity between strings.*

Description

Calculate the bag-of-characters similarity between strings.

Usage

```
orderinsensitivedists(strings = NULL, split = NULL,
  segmentcounts = segment.counts(strings, split))
```

Arguments

strings	a vector or list of strings
split	boundary sequency at which to segment the strings (default splits the string into all its constituent characters)
segmentcounts	if custom segmentation is required, the pre-segmented strings can be passed as this argument (which is a list of lists)

Value

a distance matrix

See Also

dist

Examples

```
orderinsensitivedists(c("xxxx", "asdf", "asd", "dsa"))
```

---

page.test *Page test for monotonicity of ranks.*

Description

Given N replications of k different treatments/conditions, tests whether the *median ordinal ranks* m_i of the treatments are identical

$$m_1 = m_2 = \ldots = m_k$$

against the alternative hypothesis

$$m_1 \leq m_2 \leq \ldots \leq m_k$$

where *at least one* of the inequalities is a strict inequality (Siegel and Castellan 1988, p.184). Given that even a single point change in the distribution of ranks across conditions represents evidence against the null hypothesis, the Page test is simply a test for *some ordered differences in ranks*, but not a 'trend test' in any meaningful way (see also the Page test tutorial).

Usage

```
page.test(data, verbose = TRUE)

page.L(data, verbose = TRUE, ties.method = "average")

page.compute.exact(k, N, L)
```

Arguments

data	a matrix with the different conditions along its k columns and the N replications along rows. Conversion of the data to ordinal ranks is taken care of internally.
verbose	whether to print the final rankings based on which the L statistic is computed
ties.method	how to resolve tied ranks. Passed on to rank, should be left on "average" (the default).
k	number of conditions/generations
N	number of replications/chains
L	value of the Page L statistic

Details

Tests the given matrix for monotonically *increasing* ranks across k linearly ordered conditions (along columns) based on N replications (along rows). To test for monotonically *decreasing* ranks, either reverse the order of columns, or simply invert the rank ordering by calling - on the entire dataset.

Exact p-values are computed for k up to 22, using the pre-computed null distributions from the pspearman package. For larger k, p-values are computed based on a Normal distribution approximation (Siegel and Castellan, 1988).

Value

page.test returns a list of class pagetest (and htest) containing the following elements:

statistic value of the L statistic for the data set

parameter a named vector specifying the number of conditions (k) and replications (N) of the data (which is the number of columns and rows of the data set, respectively)

p.value significance level

p.type whether the computed p-value is "exact" or "approximate"

Functions

- page.test: See above.
- page.L: Calculate Page's L statistic for the given dataset.
- page.compute.exact: Calculate exact significance levels of the Page L statistic. Returns a single numeric indicating the null probability of the Page statistic with the given k, N being greater or equal than the given L.

References

Siegel, S., and N. J. Castellan, Jr. (1988). Nonparametric Statistics for the Behavioral Sciences. McGraw-Hill.

See Also

rank, Page test tutorial

Examples

```
# exact p value computation for N=4, k=4
page.test(t(replicate(4, sample(4))))

# exact p value computation for N=4, k=10
page.test(t(replicate(4, sample(10))))

# approximate p value computation for N=4, k=23
result <- page.test(t(replicate(4, sample(23))), verbose = FALSE)

print(result)
```

```
# raw calculation of the significance levels
page.compute.exact(6, 4, 322)
```

`read.dist` *Read a distance matrix from a file or data frame.*

Description

Read a distance matrix from a file or data frame.

Usage

```
read.dist(data, el1.column = 1, el2.column = 2, dist.columns = 3)
```

Arguments

`data`	a filename, data frame or matrix
`el1.column`	the column name or id specifying the first element
`el2.column`	the column name or id specifying the second element
`dist.columns`	the column name(s) or id(s) specifying the distance(s) between the two corresponding elements

Value

a distance matrix (or list of distance matrixes when there is more than one `dist.columns`) of type `matrix`

Examples

```
read.dist(cbind(c(1,1,1,2,2,3), c(2,3,4,3,4,4), 1:6, 6:1), dist.columns=c(3,4))
```

`repmatrix` *Extend a matrix by repetition of elements.*

Description

Returns a new matrix, where the entries of the original matrix are repeated along both dimensions.

Usage

```
repmatrix(x, times = 1, each = 1, times.row = times, times.col = times,
  each.row = each, each.col = each, ...)
```

Arguments

x	a matrix
times	how often the matrix should be replicated next to itself
each	how often individual cells should be replicated next to themselves
times.row	number of vertical repetitions of the matrix, overrides times
times.col	number of horizontal repetitions of the matrix, overrides times
each.row	number of vertical repetitions of individual elements, overrides each
each.col	number of horizontal repetitions of individual elements, overrides each
...	not used

Value

A matrix, which will have times*each times more rows and columns than the original matrix.

See Also

rep

Examples

```
repmatrix(diag(4))
repmatrix(diag(4), times=2)
repmatrix(diag(4), each=2)
repmatrix(diag(3), times=2, each=2)
repmatrix(diag(4), each.row=2)
repmatrix(diag(4), times.row=2)
```

---

segment.string *Split strings into their constituent segments.*

Description

Split strings into their constituent segments (and count them).

Usage

```
segment.string(x, split = NULL)

segment.counts(x, split = NULL)
```

Arguments

x	one or more strings to be split (and, optionally, counted)
split	the boundary character or sequence at which to segment the string(s). The default, NULL, splits the string after every character.

Functions

- segment.string: Returns a list (of the same length as x), each item a vector of character vectors.
- segment.counts: Calculate the frequency of individual characters in one or more strings. Returns a matrix with one row for every string in x.

Examples

```
segment.string(c("asd", "fghj"))

segment.string(c("la-dee-da", "lala-la"), "-")
segment.counts(c("asd", "aasd", "asdf"))
```

---

shuffle.locations *Permute the rows and columns of a square matrix.*

Description

Returns the given matrix with rows and columns permuted in the same order.

Usage

```
shuffle.locations(m, perm = sample.int(dim(m)[1]))
```

Arguments

m	a matrix with an equal number of rows and columns
perm	vector of indices specifying the new order of rows/columns

Value

a matrix of the same size as m

---

sm.compositionality *Spike's segmentation and measure of additive compositionality.*

Description

Implementation of the Spike-Montague segmentation and measure of additive compositionality (Spike 2016), which finds the most predictive associations between meaning features and substrings. Computation is deterministic and fast.

Usage

```
sm.compositionality(x, y, groups = NULL, strict = FALSE)

sm.segmentation(x, y, strict = FALSE)
```

Arguments

x	a list or vector of character sequences specifying the signals to be analysed. Alternatively, x can also be a formula of the format s ~ m1 + m2 + ..., where s and m1, m2, etc. specify the column names of the signals and meaning features found in the data frame that is passed as the second argument.
y	a matrix or data frame with as many rows as there are signals, indicating the presence/value of the different meaning dimensions along columns (see section Meaning data format). If x is a formula, the y data frame can contain any number of columns, but only the ones whose column name is specified in the formula will be considered.
groups	a list or vector with as many items as strings, used to split strings and meanings into data sets for which compositionality measures are computed separately.
strict	logical: if TRUE, perform additional filtering of candidate segments. In particular, it removes combinations of segments (across meanings) which overlap in at least one of the strings where they co-occur. For convenience, it also removes segments which are shorter substrings of longer candidates (for the same meaning feature).

Details

The algorithm works on compositional meanings that can be expressed as sets of categorical meaning features (see below), and does not take the order of elements into account. Rather than looking directly at how complex meanings are expressed, the measure really captures the degree to which a homonymy- and synonymy-free signalling system exists at the level of *individual semantic features*.

The segmentation algorithm provided by `sm.segmentation()` scans through all sub-strings found in `strings` to find the pairings of meaning features and sub-strings whose respective presence is *most predictive of each other*. Mathematically, for every meaning feature $f \in M$, it finds the sub-string s_{ij} from the set of strings S that yields the highest *mutual predictability* across all signals,

$$mp(f, S) = \max_{s_{ij} \in S} P(f|s_{ij}) \cdot P(s_{ij}|f) \ .$$

Based on the mutual predictability levels obtained for the individual meaning features, `sm.compositionality` then computes the mean mutual predictability weighted by the individual features' relative frequencies of attestation, i.e.

$$mp(M, S) = \sum_{f \in M} freq_f \cdot mp(f, S) \ ,$$

as a measure of the overall compositionality of the signalling system.

Since mutual predictability is determined seperately for every meaning feature, the most predictive sub-strings posited for different meaning features as returned by `sm.segmentation()` can overlap, and even coincide completely. Such results are generally indicative of either limited data (in particular frequent co-occurrence of the meaning features in question), or spurious results in the absence of a consistent signalling system. The latter will also be indicated by the significance level of the given mutual predictability.

Value

`sm.segmentation` provides detailed information about the most predictably co-occurring segments for every meaning feature. It returns a data frame with one row for every meaning feature, in descending order of the mutual predictability from (and to) their corresponding string segments. The data frame has the following columns:

`N` The number of signal-meaning pairings in which this meaning feature was attested.

`mp` The highest mutual predictability between this meaning feature and one (or more) segments that was found.

`p` Significance levels of the given mutual predictability, i.e. the probability that the given mutual predictability level could be reached by chance. The calculation depends on the frequency of the meaning feature as well as the number and relative frequency of all substrings across all signals (see below).

`ties` The number of substrings found in `strings` which have this same level of mutual predictability with the meaning feature.

`segments` For `strict=FALSE`: a list containing the `ties` substrings in descending order of their length (the ordering is for convenience only and not inherently meaningful). When `strict=TRUE`, the lists of segments for each meaning feature are all of the same length, with a meaningful relationship of the order of segments across the different rows: every set of segments which are found in the same position for each of the different meaning features constitute a valid segmentation where the segments occurrences in the actual signals do not overlap.

`sm.compositionality` calculates the weighted average of the mutual predictability of all meaning features and their most predictably co-occurring strings, as computed by `sm.segmentation`. The function returns a data frame of three columns: `N` is the total number of signals (utterances) on which the computation was based, `M` the number of distinct meaning features attested across all signals, and `meanmp` the mean mutual predictability across all these features, weighted by the features' relative frequency. When `groups` is not `NULL`, the data frame contains one row for every group.

Null distribution and p-value calculation

A perfectly unambiguous mapping between a meaning feature to a specific string segment will always yield a mutual predictability of `1`. In the absence of such a regular mapping, on the other hand, chance co-occurrences of strings and meanings will in most cases stop the mutual predictability from going all the way down to `0`. In order to help distinguish chance co-occurrence levels from significant signal-meaning associations, `sm.segmentation()` provides significance levels for the mutual predictability levels obtained for each meaning feature.

What is the baseline level of association between a meaning feature and a set of sub-strings that we would expect to be due to chance co-occurrences? This depends on several factors, from the number of data points on which the analysis is based to the frequency of the meaning feature in question and, perhaps most importantly, the overall makeup of the different substrings that are present in the signals. Since every substring attested in the data is a candidate for signalling the presence of a meaning feature, the absolute number of different substrings greatly affects the likelihood of chance signal-meaning associations. (Diversity of the set of substrings is in turn heavily influenced by the size of the underlying alphabet, a factor which is often not appreciated.)

For every candidate substring, the degree of association with a specific meaning feature that we would expect by chance is again dependent on the absolute number of signals in which the substring is attested.

Starting from the simplest case, take a meaning that is featured in m of the total n signals (where $0 < m \leq n$). Assume next that there is a string segment that is attested in s of these signals (where again $0 < s \leq n$). The degree of association between the meaning feature and string segment is dependent on the number of times that they co-occur, which can be no more than $c_{max} = min(m, s)$ times. The null probability of getting a given number of co-occurrences can be obtained by considering all possible reshufflings of the meaning feature in question across all signals: if s signals contain a given substring, how many of s randomly drawn signals from the pool of n signals would contain the meaning feature if a total of m signals in the pool did? Approached from this angle, the likelihood of the number of co-occurrences follows the hypergeometric distribution, with c being the number of successes when taking s draws without replacement from a population of size n with fixed number of successes m.

For every number of co-occurrences $c \in [0, c_{max}]$, one can compute the corresponding mutual probability level as $p(c|s) \cdot p(c|m)$ to obtain the null distribution of mutual predictability levels between a meaning feature and *one* substring of a particular frequency s:

$$Pr(mp = p(c|s) \cdot p(c|m)) = f(k = c; N = n, K = m, n = s)$$

From this, we can now derive the null distribution for the entire set of attested substrings as follows: making the simplifying assumption that the occurrences of different substrings are independent of each other, we first aggregate over the null distributions of all the individual substrings to obtain the mean probability $p = Pr(X \geq mp)$ of finding a given mutual predictability level at least as high as mp for one randomly drawn string from the entire population of substrings. Assuming the total number of candidate substrings is $|S|$, the overall null probability that at least one of them would yield a mutual predictability at least as high is

$$Pr(X \geq 0), X \equiv B(n = |S|, p = p) \ .$$

Note that, since the null distribution also depends on the frequency with which the meaning feature is attested, the significance levels corresponding to a given mutual predictability level are not necessarily identical for all meaning features, even within one analysis.

(In theory, one can also compute an overall p-value of the weighted mean mutual predictability as calculated by `sm.compositionality`. However, the significance levels for the individual meaning features are much more insightful and should therefore be consulted directly.)

Meaning data format

The `meanings` argument can be a matrix or data frame in one of two formats. If it is a matrix of logicals (`TRUE`/`FALSE` values), then the columns are assumed to refer to meaning *features*, with individual cells indicating whether the meaning feature is present or absent in the signal represented by that row (see `binaryfeaturematrix()` for an explanation). If `meanings` is a data frame or matrix of any other type, it is assumed that the columns specify different meaning dimensions, with the cell values showing the levels with which the different dimensions can be realised. This dimension-based representation is automatically converted to a feature-based one by calling `binaryfeaturematrix()`. As a consequence, whatever the actual types of the columns in the meaning matrix, *they will be treated as categorical factors* for the purpose of this algorithm, also discarding any explicit knowledge of which 'meaning dimension' they might belong to.

References

Spike, M. 2016 *Minimal requirements for the cultural evolution of language*. PhD thesis, The University of Edinburgh. http://hdl.handle.net/1842/25930.

See Also

binaryfeaturematrix(), ssm.compositionality()

Examples

```
# perfect communication system for two meaning features (which are marked
# as either present or absent)
sm.compositionality(c("a", "b", "ab"),
  cbind(a=c(TRUE, FALSE, TRUE), b=c(FALSE, TRUE, TRUE)))
sm.segmentation(c("a", "b", "ab"),
  cbind(a=c(TRUE, FALSE, TRUE), b=c(FALSE, TRUE, TRUE)))

# not quite perfect communication system
sm.compositionality(c("as", "bas", "basf"),
  cbind(a=c(TRUE, FALSE, TRUE), b=c(FALSE, TRUE, TRUE)))
sm.segmentation(c("as", "bas", "basf"),
  cbind(a=c(TRUE, FALSE, TRUE), b=c(FALSE, TRUE, TRUE)))

# same communication system, but force candidate segments to be non-overlapping
# via the 'strict' option
sm.segmentation(c("as", "bas", "basf"),
  cbind(a=c(TRUE, FALSE, TRUE), b=c(FALSE, TRUE, TRUE)), strict=TRUE)


# the function also accepts meaning-dimension based matrix definitions:
print(twobytwoanimals <- enumerate.meaningcombinations(c(animal=2, colour=2)))

# note how there are many more candidate segments than just the full length
# ones. the less data we have, the more likely it is that shorter substrings
# will be just as predictable as the full segments that contain them.
sm.segmentation(c("greendog", "bluedog", "greencat", "bluecat"), twobytwoanimals)

# perform the same analysis, but using the formula interface
print(twobytwosignalingsystem <- cbind(twobytwoanimals,
  signal=c("greendog", "bluedog", "greencat", "bluecat")))

sm.segmentation(signal ~ colour + animal, twobytwosignalingsystem)

# since there is no overlap in the constituent characters of the identified
# 'morphemes', they are all tied in their mutual predictiveness with the
# (shorter) substrings they contain
#
# to reduce the pool of candidate segments to those which are
# non-overlapping and of maximal length, again use the 'strict=TRUE' option:

sm.segmentation(signal ~ colour + animal, twobytwosignalingsystem, strict=TRUE)
```

`ssm.compositionality` *Find a segmentation that maximises the overall string coverage across all signals.*

Description

This algorithm builds on Spike's measure of compositionality (see `sm.compositionality`), except instead of simply determining which segment(s) have the highest mutual predictability for each meaning feature separately, it attempts to find a combination of non-overlapping segments for each feature that maximises the overall string coverage over all signals. In other words, it tries to find a segmentation which can account for (or 'explain') as much of the string material in the signals as possible.

Usage

```
ssm.compositionality(x, y, groups = NULL)

ssm.segmentation(x, y, mergefeatures = FALSE, verbose = FALSE)
```

Arguments

`x`	a list or vector of character sequences
`y`	a matrix or data frame with as many rows as there are strings (see section Meaning data format)
`groups`	a list or vector with as many items as strings, used to split the signals and meanings into data sets for which the compositionality measures are computed separately.
`mergefeatures`	logical: if `TRUE`, `ssm.segmentation` will try to improve on the initial solution by incrementally merging pairs of meaning features as long as doing so improves the overall string coverage of the segmentation.
`verbose`	logical: if `TRUE`, messages detailed information about the number of segment combinations considered for every coverage computed.

Details

For large data sets and long strings, this computation can get very slow. If the attested signals are such that no perfect segmentation is possible, this algorithm is not guaranteed to find any segmentation (as no such segmentation might exist).

See Also

`sm.compositionality`

Examples

```
ssm.segmentation(c("as", "bas", "basf"),
  cbind(a=c(TRUE, FALSE, TRUE), b=c(FALSE, TRUE, TRUE)))


# signaling system where one meaning distinction is not encoded in the signals
print(threebytwoanimals <- enumerate.meaningcombinations(list(animal=c("dog", "cat", "tiger"),
  colour=c("col1", "col2"))))

ssm.segmentation(c("greendog", "bluedog", "greenfeline", "bluefeline", "greenfeline", "bluefeline"),
  threebytwoanimals)

# the same analysis again, but allow merging of features
ssm.segmentation(c("greendog", "bluedog", "greenfeline", "bluefeline", "greenfeline", "bluefeline"),
  threebytwoanimals, mergefeatures=TRUE)
```

`temperature.colors` *Create a vector of 'temperature' colors (from blue over white to red).*

Description

Create a vector of 'temperature' colors (from blue over white to red).

Usage

```
temperature.colors(mn, mx = NULL, intensity = 1)
```

Arguments

`mn`	integer: when `mx` is not specified, total number of colors (>1) in the palette. when `mx` is specified: 'coldest' temperature (see examples)
`mx`	integer: 'warmest' temperature (see examples)
`intensity`	saturation of the most extreme color(s), in the range `[0,1]`.

See Also

gray, hsv, rainbow

Examples

```
# full intensity
image(as.matrix(1:7), z=as.matrix(1:7), col=temperature.colors(7))
# half intensity
image(as.matrix(1:7), z=as.matrix(1:7), col=temperature.colors(7, intensity=0.5))
# skewed palette with more negative than positive temperature colors
image(as.matrix(1:7), z=as.matrix(1:7), col=temperature.colors(-4, 2))
```

`wrap.meaningdistfunction` *Make a meaning distance function vectorisable.*

Description

This function takes as its only argument a function `f(m1, m2)` which returns a single numeric indicating the distance between two 'meanings' `m1, m2` (which are themselves most likely vectors or lists). Based on `f`, this function returns a function `g(mm)` which takes as its only argument a matrix or data frame `mm` with the meaning elements (equivalent to the ones in `m1, m2`) along columns and different meaning combinations (like `m1, m2, ...`) along rows. This function returns a distance matrix of class `dist` containing all pairwise distances between the rows of `mm`. The resulting function g can be passed to other functions in this package, in particular `mantel.test`.

Usage

```
wrap.meaningdistfunction(pairwisemeaningdistfun)
```

Arguments

`pairwisemeaningdistfun`
: a function of two arguments returning a single numeric indicating the semantic distance between its arguments

Details

The meaning distance function should be commutative, i.e. `f(a,b) = f(b,a)`, and meanings should have a distance of zero to themselves, i.e. `f(a,a) = 0`.

Value

A function that takes a meaning matrix and returns a corresponding distance matrix of class `dist`.

Examples

```
trivialdistance <- function(a, b) return(a - b)

trivialmeanings <- as.matrix(3:1)
trivialdistance(trivialmeanings[1], trivialmeanings[2])
trivialdistance(trivialmeanings[1], trivialmeanings[3])
trivialdistance(trivialmeanings[2], trivialmeanings[3])

distmatrixfunction <- wrap.meaningdistfunction(trivialdistance)
distmatrixfunction(trivialmeanings)
```

Index

adist, *8*, *10*

binaryfeaturematrix, 2
binaryfeaturematrix(), *19*, *20*

check.dist, 3, *9*
cor, *8*, *10*
cor.test, *7*
count.substring.occurrences, 4

dist, *3*, *6*, *9*, *11*, *12*, *23*

enumerate.meaningcombinations, *2*, 4
enumerate.substrings, 5

gray, *22*

hammingdists, *5*, 6, *10*
hammingdists(), *8*
hsv, *22*

mantel.test, 7, *23*

normalisedlevenshteindists, *10*, 11
normalisedlevenshteindists(), *8*

orderinsensitivedists, *10*, 11
orderinsensitivedists(), *8*

page.compute.exact *(*page.test*)*, 12
page.L *(*page.test*)*, 12
page.test, 12
plot.default, *9*
plot.mantel *(*mantel.test*)*, 7

rainbow, *22*
rank, *12*, *13*
read.dist, 14
rep, *15*
repmatrix, 14

segment.counts *(*segment.string*)*, 15
segment.string, 15
shuffle.locations, 16
sm.compositionality, 16, *21*
sm.segmentation *(*sm.compositionality*)*, 16
ssm.compositionality, 21
ssm.compositionality(), *20*
ssm.segmentation *(*ssm.compositionality*)*, 21

temperature.colors, 22

wrap.meaningdistfunction, *5*, 23
wrap.meaningdistfunction(), *8*