Package ‘GrpString’

January 20, 2025

Type Package

Title Patterns and Statistical Differences Between Two Groups of Strings

Version 0.3.2

Date 2017-08-15

Author Hui Tang, Norbert J. Pienta

Maintainer Hui (Tom) Tang <htang2013@gmail.com>

Description Methods include converting series of event names to strings, finding common patterns in a group of strings, discovering featured patterns when comparing two groups of strings as well as the number and starting position of each pattern in each string, obtaining transition matrix, computing transition entropy, statistically comparing the difference between two groups of strings, and clustering string groups. Event names can be any action names or labels such as events in log files or areas of interest (AOIs) in eye tracking research.

License GPL-2

Depends R (>= 3.0.1)

Imports utils, plyr, Rcpp, cluster, graphics, stats

RoxygenNote 6.0.1

Suggests testthat

NeedsCompilation no

Repository CRAN

Date/Publication 2017-08-15 21:54:41 UTC

Contents

`GrpString-package` *Patterns and Statistical Differences Between Two Groups of Strings*

Description

Methods include converting series of event names to strings, finding common patterns in a group of strings, discovering featured patterns when comparing two groups of strings as well as the number and starting position of each pattern in each string, obtaining transition matrix, computing transition entropy, statistically comparing the difference between two groups of strings, and clustering string groups.

Event names can be any action names or labels such as events in log files or areas of interest (AOIs) in eye tracking research.

Details

Package:	GrpString
Type:	Package
Version:	0.3.2
Date:	2017-08-15
License:	GPL-2

Some functions have two or more types, e.g., one returning a data frame or a vector and the other exporting one or more than one .txt file to the current directory. The former is a simple version of the functions, while the latter can be considered as a generalized or complex version of the former one. This is because some data sets are large (e.g., many rows or columns), or it helps the users to view and manage the results when more than one data set is exported. Example function pairs are EveStr - EveString, CommonPatt - CommonPattern, and PatternInfo - FeaturedPatt.

In addition, to save the users' effort, the function EveString utilizes an input file (which can be a .txt or .csv file) instead of a data frame. This is because the input data are more convenient to be stored in a .txt or .csv file than in a data frame. We suggest that the users copy the relevant input files (including eve1d.txt and eve1d.csv) to a different directory, because the function exports files to the same directory where the input files locate.

Author(s)

Hui Tang, Norbert J. Pienta

Maintainer: Hui (Tom) Tang <htang2013@gmail.com>

Examples

```
# Discover common patterns in a group of strings
strs.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
CommonPatt(strs.vec, low = 30)
```

CommonPatt *Discovers common patterns in a group of strings - simplified version*

Description

CommonPatt finds common patterns shared by a group of strings.

A common pattern is defined as a substring with the minimum length of three that occurs at least twice among a group of strings.

Usage

```
CommonPatt(strings.vec, low = 10)
```

Arguments

`strings.vec`	String Vector.
`low`	Cutoff. It is the minimum percentage of the occurrence of patterns that the user specifies. The default value is 10.

Details

The argument 'low' ranges from 0 to 100 in percentage.

Value

The function returns a data frame containing patterns, lengths and percentages of patterns.

row name - The initial order of substrings, which can be ignored.

Column 1 - Pattern: common pattern.

Column 2 - Freq_grp: the overall frequency (times of occurrence) of each pattern.

Column 3 - Percent_grp: the ratio of Freq_grp to the number of original strings, in percent.

Column 4 - Length: the length (i.e., number of characters) of pattern.

Column 5 - Freq_str: similar to Freq_grp; but each pattern is counted only once in a string even if the string contains that pattern multiple times.

Column 6 - Percent_str: similar to Percent; but each pattern is counted only once in a string if this string contains the pattern.

Data is sorted by Length, then Freq_grp, in decreasing order.

References

1. H. Tang; E. Day; L. Kendhammer; J. N. Moore; S. A. Brown; N. J. Pienta. (2016). Eye movement patterns in solving science ordering problems. Journal of eye movement research, 9(3), 1-13.

2. J. J. Topczewski; A. M. Topczewski; H. Tang; L. Kendhammer; N. J. Pienta.(2017). NMR Spectra through the eyes of a student: eye tracking applied to NMR items. Journal of chemical education, 94(1), 29-37.

3. J. M. West; A. H. Haake; E. P. Rozanksi; K. S. Karn. (2006). EyePatterns: Software for identifying patterns and similarities across fixation sequences. In Proceedings of the Symposium on Eye-tracking Research & Applications, ACM Press, New York, 149-154.

See Also

`CommonPattern`

Examples

```
# Simple strings, non-default cutoff
strs.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
CommonPatt(strs.vec, low = 30)
```

`CommonPattern` *Discovers common patterns in a group of strings - full version*

Description

CommonPattern finds common patterns shared by a group of strings.

It is the extended version of CommonPatt and the users have more options.

A common pattern is defined as a substring with the minimum length of three that occurs at least twice among a group of strings.

Usage

```
CommonPattern(strings.vec, low = 5, high = 25, interval = 5, eveChar.df)
```

Arguments

`strings.vec`	String vector.
`low`	The lowest cutoff. It is the minimum percentage of the occurrence of patterns that the user specifies. The default value is 5.
`high`	The highest cutoff, which is the maximum percentage of the occurrence of patterns that the user specifies. The default value is 25.
`interval`	The increment percentage from the lowest to the highest cutoff. The default value is 5.
`eveChar.df`	Data frame that stores the event name - character conversion key (optional).

Details

The arguments 'low', 'high' and 'interval' range from 0 to 100 in percentage.

Value

A set of .txt files are exported into the current directory that contain patterns, lengths, percents of patterns, and converted event names if eveChar.fle is included. The names of these files are the name of strings.vec appended with the percentages. In addition, a file with all patterns that occurred at least 2 times is exported.

row name - The initial order of substrings, which can be ignored.

Column 1 - Pattern: common pattern.

Column 2 - Freq_grp: the overall frequency (times of occurrence) of each pattern.

Column 3 - Percent_grp: the ratio of Freq_grp to the number of original strings, in percent.

Column 4 - Length: the length (i.e., number of characters) of pattern.

Column 5 - Freq_str: similar to Freq_grp; but each pattern is counted only once in a string even if the string contains that pattern multiple times.

Column 6 - Percent_str: similar to Percent; but each pattern is counted only once in a string if this string contains the pattern.

Column 7 - Event_name (optional): sequence of event names converted back from pattern string

Data is sorted by Length, then Freq_grp, in decreasing order.

Note

The time to run this function can be relatively long (from seconds to minutes depending on the number and lengths of strings as well as the performance of computers).

References

1. H. Tang; E. Day; L. Kendhammer; J. N. Moore; S. A. Brown; N. J. Pienta. (2016). Eye movement patterns in solving science ordering problems. Journal of eye movement research, 9(3), 1-13.

2. J. J. Topczewski; A. M. Topczewski; H. Tang; L. Kendhammer; N. J. Pienta.(2017). NMR Spectra through the eyes of a student: eye tracking applied to NMR items. Journal of chemical education, 94(1), 29-37.

3. J. M. West; A. H. Haake; E. P. Rozanksi; K. S. Karn. (2006). EyePatterns: Software for identifying patterns and similarities across fixation sequences. In Proceedings of the Symposium on Eye-tracking Research & Applications, ACM Press, New York, 149-154.

See Also

CommonPatt

Examples

```
# simple strings
strs.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
CommonPattern(strs.vec, low = 30, high = 50, interval = 20)

# None-default cutoff values, with conversion back
data(eventChar.df)
data(str1)
s0 <- str1[5:15]
CommonPattern(s0, low = 20, high = 30, interval = 10, eveChar.df = eventChar.df)
```

DupRm *Removes successive duplicates in strings*

Description

DupRm removes successive duplicated characters in each string in a group.

Usage

```
DupRm(strings.vec)
```

Arguments

strings.vec	String Vector.

Value

Returns a string vector with successive duplicates been removed.

That is, each string in the export vector is "collapsed".

Examples

```
# Simple example
dup1 <- "000<<<<<DDDFFF333333qqqqqKKKKK33FFF"
dup3 <- "aaBB111^^~~~555667777000000!!!###$$$$$$&&&(((((***)))))@@@@@>>>>99"
dup13 <- c(dup1, dup3)
DupRm(dup13)
```

`event1s.df` *Data frame containing event names*

Description

A data frame containing event names, There are 45 rows. Each row has 26 event names.

Usage

```
data(event1s.df)
```

Format

A data frame with 45 observations or rows.

Note

The event names are from an eye tracking study. Thus, each event name is actually an area of interst (AOI).

Examples

```
data(event1s.df)
```

`eventChar.df` *Event name - character conversion key*

Description

A data frame where each element in column 'event' (event name) corresponds to an element in column 'char' (character), which can be a letter, digit, or a special character.

Usage

```
data(eventChar.df)
```

Format

A data frame with 16 observations on the following 2 variables.

`event` a character vector

`char` a character vector

Examples

```
data(eventChar.df)
```

`EveS` *Converts sequence of event names to a string*

Description

EveS converts event names in a vector to a single string based on the conversion key.

Usage

```
EveS(eves.vec, eveName.vec, char.vec)
```

Arguments

`eves.vec`	Vector that stores event names to be converted.
`eveName.vec`	Event name vector in a conversion key.
`char.vec`	Character vector in a conversion key.

Details

The lengths of eveName.vec and char.vec are the same.

Each element (event name) in eveName.vec corresponds to an element (character) in char.vec.

An element in char.vec can be a letter, digit, or a special character.

Value

The function returns a string.

See Also

`EveStr`, `EveString`

Examples

```
event.vec <- c("aoi_1", "aoi_2", "aoi_3", "aoi_2", "aoi_1")
eve.names <- c("aoi_1", "aoi_2", "aoi_3")
labels <- c("a", "b", "c")
EveS(event.vec, eve.names, labels)
```

EveStr — *Converts sequences of event names to strings - same length*

Description

EveStr converts event names in a data frame to a string vector. In the data frame, each row, which has the same number of event names, is converted to a string based on the conversion key. A string vector is exported. As a result, in the vector, each converted string has the same length.

Usage

```
EveStr(eveName.df, eveName.vec, char.vec)
```

Arguments

`eveName.df`	Data frame that stores event names to be converted.
`eveName.vec`	Event name vector in a conversion key.
`char.vec`	Character vector in a conversion key.

Details

The lengths of eveName.vec and char.vec are the same.

Each element (event name) in eveName.vec corresponds to an element (character) in char.vec.

An element in char.vec can be a letter, digit, or a special character.

Value

The function returns a string vector.

See Also

`EveS`, `EveString`

Examples

```
# small number of event names
event.df <- data.frame(c("aoi_1", "aoi_2"),
                     c("aoi_1", "aoi_3"),
                     c("aoi_3", "aoi_5"))
event.name.vec <- c("aoi_1", "aoi_2", "aoi_3", "aoi_4", "aoi_5")
label.vec <- c("a", "b", "c", "d", "e")
EveStr(event.df, event.name.vec, label.vec)

# more event names
data(event1s.df)
data(eventChar.df)
EveStr(event1s.df, eventChar.df$event, eventChar.df$char)
```

EveString — *Converts sequences of event names to strings - generalized*

Description

EveString converts event names in a data frame to a string vector.

In the data frame, each row, which can have different number of event names, is converted to a string based on the conversion key. As a result, in the vector, converted strings may have different lengths.

Usage

```
EveString(eveName.file, eveName.vec, char.vec)
```

Arguments

`eveName.file`	File that stores event names to be converted.
`eveName.vec`	Vector of event names in a conversion key.
`char.vec`	Characters vector in a conversion key.

Details

In general, it is not convenient to deal with data frames where different rows have different numbers of elements. Thus, it is easier to use a text file than to use a data frame when storing different numbers of event names in rows. As a result, this function utilizes a .txt or .csv file (for eveName.file) and handles such task to save users' effort.

Value

The function returns a vector containing converted strings that generally have different lengths.

If not all event names are converted to characters, a warning message will be printed out.

Note

eveName.file is the name of a file. Thus quote signs are needed when a file name (and its directory) is directly used in the function.

If the example is used, the eveName.file will be eve1d.txt, which is located in your R library. The users may copy eve1d.txt to a directory that can be easily found.

See Also

`EveS`, `EveStr`

Examples

```
data(eventChar.df)
event1d <- paste(path.package("GrpString"), "/extdata/eve1d.txt", sep = "")
EveString(event1d, eventChar.df$event, eventChar.df$char)
```

`FeaturedPatt` *Discovers featured patterns in two groups of strings*

Description

FeaturedPatt discovers featured patterns that are in one group of strings.

Usage

```
FeaturedPatt(grp1_pattern, grp2_pattern, grp1_string, grp2_string)
```

Arguments

`grp1_pattern`	Patterns shared by a certain percent of strings in string group 1.
`grp2_pattern`	Patterns shared by a certain percent of strings in string group 2.
`grp1_string`	String group 1.
`grp2_string`	String group 2.

Details

A (common) pattern is defined as a substring with the minimum length of three that occurs at least twice among a group of strings.

In practice, a pattern usually is not shared by all the strings in a group. Thus, featured patterns may be obtained from two pattern vectors, each of which contains patterns that are shared by a certain percent of strings in a group. As a result, featured patterns can possibly appear in both groups of strings, athough ideally, a featured pattern should only appears in one of the two groups of strings.

Value

The function exports five text files:

File that lists featured patterns: column 1 for string group 1; column 2 for string group 2.

Four files that contain information about each group of patterns in each group of strings.

The information includes the number of each of the patterns in each string and the starting

positions of the first occurring patterns, as well as the lengths of original strings.

If a pattern does not appear in a string, -1 is returned.

In the above four files: the first column contains original strings; the second column contains the length of strings; the third column contains the number of featured patterns each string has; each of the columns from the fourth is the starting position of a pattern that first appears in a string.

In addition, messages are printed out for the four situations of each pattern group in each string group. The messages include the number and the ratio of strings that have at least one featured pattern.

See Also

`PatternInfo`, `CommonPatt`, `CommonPattern`

Examples

```
data(str1)
data(str2)
data(p1_20up)
data(p2_25up)
FeaturedPatt(p1_20up, p2_25up, str1, str2)
```

HistDif *Customizes the positions of legend and p value in a histogram*

Description

The positions of legend and p value in the histogram generated from function StrDif may not be ideal for different (permutations on differences of normalized Levenshtein distances) situations. HistDif customizes the positions of legend and p value in the histogram of the statistical difference of two groups of strings.

Usage

```
HistDif(dif.vec, obsDif, pvalue, o.x = 0.01, o.y = 0, p.x = 0.015, p.y = 0)
```

Arguments

`dif.vec`	Vector containing differences of normalized Levenshtein differences (LD) from the permutation test.
`obsDif`	The "observed" or original difference between between-group and within-group normalized LD.
`pvalue`	p value of the permutation test.
`o.x`	x coordinate of the legend in the histogram, default is 0.01.
`o.y`	y coordinate of the legend in the histogram, default is 0.
`p.x`	x coordinate of the p value in the histogram, default is 0.015.
`p.y`	y coordinate of the p value in the histogram, default is 0.

Details

The default values of o.y and p.y are 0. They are actually related to the number of permutations (num_perm): o.y is above 0.2 * num_perm, and p.y is below 0.2 * num_perm. If non-default values are used, the values become absolute y coordinates.

See Also

`StrDif`

Examples

```
# simple example, use the vectors of ld difference values obtained from StrDif
strs1.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
strs2.vec <- c("xYZdkfAxDa", "ef1563xy", "BC9Dzy35X", "AkeC1fxz", "65CyAdC", "Dfy3f69k")
ld.dif.vec <- StrDif(strs1.vec, strs2.vec, num_perm = 500, p.x = 0.025)
HistDif(dif.vec = ld.dif.vec, obsDif = 0.00751, pvalue = 0.35600,
        o.x = 0.025, p.x = 0.040, p.y = 75)
```

p1_20up *Patterns from string group 1*

Description

Patterns that occur at least 20 percent compared to the number of strings in string group 1. It can be obtained from one of the exported files from CommonPattern(str1).

Usage

```
data(p1_20up)
```

Format

The format is: chr [1:32] "212" "202" "BAB" "D0D" "F0F" "020" "B0B" "010" "404" "C0C" ...

Examples

```
data(p1_20up)
```

p2_25up *Patterns from string group 2*

Description

Patterns that occur at least 25 percent compared to the number of strings in string group 2. It can be obtained from one of the exported files from CommonPattern(str2).

Usage

```
data(p2_25up)
```

Format

The format is: chr [1:32] "0D0D" "0E0E" "E0E0" "D0D" "E0E" "F0F" "B0B" "0C0" "0D0" ...

Examples

```
data(p2_25up)
```

PatternInfo *Discovers pattern information in one group of strings*

Description

PatternInfo discovers the starting position of each pattern that occurs first or last as well as the number of patterns in each string.

Usage

```
PatternInfo(patterns, strings, rev = FALSE)
```

Arguments

`patterns`	Pattern vector.
`strings`	String vector.
`rev`	Determine whether returning the starting positions of patterns that occur first or last in strings. Default is first.

Value

Returns a data frame, which contains the length of each string, and the starting position of each pattern in each string.

See Also

`FeaturedPatt`

Examples

```
# simple strings and patterns
strs.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
patts <- c("ABC", "123")
PatternInfo(patts, strs.vec)

# simple strings and patterns, starting position of last pattern
strs.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
patts <- c("ABC", "123")
PatternInfo(patts, strs.vec, rev = TRUE)
```

str1 *String group 1*

Description

A vector containing 45 strings that have different lengths. It also can be obtained in the export file from the example in function EveString.

Usage

```
data(str1)
```

Format

The format is: chr [1:45] "D02F0E20DEDC0C30BDC0E45G050A0B5050A06BG0BA5607BA" ...

Examples

```
data(str1)
```

str2 *String group 2*

Description

A vector containing 29 strings that have different lengths.

Usage

```
data(str2)
```

Format

The format is: chr [1:29] "G21A1C14C2D0D21D2123201D23D21234320431212412421AB3EGEGE0E4G4B5G6A" ...

Examples

```
data(str2)
```

StrDif *Statistically compares the difference between two groups of strings*

Description

StrDif tests whether the difference between two groups of strings is statistically significant or not. The difference is based on normalized Levenshtein distances (LDs) between strings. A permutation test is used as the statistical method.

Usage

```
StrDif(grp1_string, grp2_string, num_perm = 1000,
       o.x = 0.01, o.y = 0, p.x = 0.015, p.y = 0)
```

Arguments

`grp1_string`	String group (vector) 1.
`grp2_string`	String group (vector) 2.
`num_perm`	Number of permutations. The default is 1000.
`o.x`	x coordinate of the legend in the histogram, default is 0.01.
`o.y`	y coordinate of the legend in the histogram, default is 0.
`p.x`	x coordinate of the p value in the histogram, default is 0.015.
`p.y`	y coordinate of the legend in the histogram, default is 0.

Details

The default values of o.y and p.y are 0. They are actually related to num_perm: o.y is above 0.2 * num_perm, and p.y is below 0.2 * num_perm. If non-default values are used, the values become absolute y coordinates.

Value

The function generates a histogram that demonstrates the distribution of the differences of LDs, the original difference, and the p value.

The function also returns a vector containing differences of normalized LDs. The total number of differences is num_perm (number of permutations).

Differences are calculated by subtracting within-group LD from between-group LD. They range from -1 to 1. The "observed" difference is the difference from the original data set.

Note

1. Because the number of permutations is usually large (default is 1000), and so is the number of elements in the vector returned from the function, it's better for the user to use a vector to store the returned results, instead of printing out directly. See the examples.

2. The positions of legend and p value in the histogram generated from function StrDif may not be ideal for different (permutations on differences of normalized Levenshtein distances) situations. Thus, this package includes another function, HistDif, to customize the positions of legend and p value in the histogram.

3. The time to run this function can be relatively long (from seconds to minutes depending on the number and lengths of strings as well as the computer performance).

4. Acknowledgement: The first version of this function was developed with significant help from Dr. Rhonda DeCook in the Department of Statistics and Actuarial Science at the University of Iowa.

References

1. H. Tang; J. J. Topczewski; A. M. Topczewski; N. J. Pienta. Permutation Test for Groups of Scanpaths Using Normalized Levenshtein Distances and Application in NMR Questions. In Proceedings of the Symposium on Eye Tracking Research and Applications, Santa Barbara, CA, March 28-30, 2012; ACM Press: New York; pp 169-172.

2. M. Feusner; B. Lukoff. (2008). Testing for statistically significant differences between groups of scan patterns. In Proceedings of the Symposium on Eye-tracking Research & Applications, ACM Press, New York, 43-46.

See Also

`HistDif`

Examples

```
# simple stings, non-default permutation number and p-value position
strs1.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
strs2.vec <- c("xYZdkfAxDa", "ef1563xy", "BC9Dzy35X", "AkeC1fxz", "65CyAdC", "Dfy3f69k")
ld.dif.vec <- StrDif(strs1.vec, strs2.vec, num_perm = 500, p.x = 0.025)

# longer strings
data(str1)
data(str2)
s1 <- str1[1:6]
s2 <- str2[1:6]
ld.dif12.vec <- StrDif(s1, s2, num_perm = 500)
```

StrHclust — *Hierarchical cluster of a group of strings*

Description

StrHclust discovers clusters of the strings in a group.

Usage

```
StrHclust(strings.vec, nclust = 2)
```

Arguments

`strings.vec`	String Vector.
`nclust`	Number of clusters. Default is 2.

Value

Returns a data frame with the specific cluster assigned to each string.

A Hierarchical dendrogram is also exported.

See Also

`StrKclust`

Examples

```
# Simple strings
strs3.vec <- c("ABCDdefABCDa", "AC3aABCD", "ACD1AB3", "xYZfgAxZY", "gf56xZYx", "AkfxzYZg")
StrHclust(strs3.vec)
```

StrKclust — *K-means clustering of a group of strings*

Description

StrKclust discovers clusters of the strings in a group.

Usage

```
StrKclust(strings.vec, nclust = 2, nstart = 1, shade = FALSE)
```

Arguments

`strings.vec`	String Vector.
`nclust`	Number of clusters. Default is 2.
`nstart`	Number of random data sets chosen to start. Default is 1.
`shade`	Whether shade the ellipses in the cluster plot or not. Default is false.

Value

Returns a data frame with the specific cluster assigned to each string.

A cluster plot is also exported.

See Also

`StrHclust`

Examples

```
# Simple strings
strs3.vec <- c("ABCDdefABCDa", "AC3aABCD", "ACD1AB3", "xYZfgAxZY", "gf56xZYx", "AkfxzYZg")
StrKclust(strs3.vec)
```

`TransEntro` *Transition entropy of a group of strings*

Description

TransEntro computes the overall transition entropy of all the strings in a group.

Usage

```
TransEntro(strings.vec)
```

Arguments

`strings.vec`	String Vector.

Details

Entropy is calculated using the Shannon entropy formula: -sum(freqs * log2(freqs)). Here, freqs are transition frequencies, which are the values in the normalized transition matrix exported by function TransMx in this package. The formula is equivalent to the function entropy.empirical in the 'entropy' package when unit is set to log2.

Value

Returns a single number.

Note

Strings with less than 2 characters are not included for computation of entropy.

References

I. Hooge; G. Camps. (2013) Scan path entropy and arrow plots: capturing scanning behavior of multiple observers. Frontiers in Psychology.

See Also

TransEntropy, TransMx

Examples

```
# simple strings
stra.vec <- c("ABCDdefABCDa", "def123DC", "A", "123aABCD", "ACD13", "AC1ABC", "3123fe")
TransEntro(stra.vec)
```

TransEntropy *Transition entropy of each string in a group*

Description

TransEntropy computes the transition entropy of each of the strings in a group.

Usage

```
TransEntropy(strings.vec)
```

Arguments

strings.vec	String Vector.

Details

Entropy is calculated using the Shannon entropy formula: -sum(freqs * log2(freqs)). Here, freqs are transition frequencies, which are the values in the normalized transition matrix exported by function TransMx in this package. The formula is equivalent to the function entropy.empirical in the 'entropy' package when unit is set to log2.

Value

Returns a number vector.

Note

Strings with less than 2 characters are not included for computation of entropy.

References

I. Hooge; G. Camps. (2013) Scan path entropy and arrow plots: capturing scanning behavior of multiple observers. Frontiers in Psychology.

See Also

TransEntro, TransMx

Examples

```
# default values
stra.vec <- c("ABCDdefABCDa", "def123DC", "A", "123aABCD", "ACD13", "AC1ABC", "3123fe")
TransEntropy(stra.vec)
```

TransInfo *Transitions in one group of strings*

Description

TransInfo discovers transitions of two adjacent characters in strings.

A transition is defined as a substring (in the forward order) with length of 2 characters.

Usage

```
TransInfo(strings.vec, type1 = "letters", type2 = "digits")
```

Arguments

strings.vec	String Vector.
type1	The first type of transition. Default value is letter.
type2	The second type of transition. Default value is digit.

Value

The function returns a data frame, which contains the numbers of type1 transition, type2 transition, and transitions belonging to neither type1 nor type2.

Note

Strings with less than 2 characters are not included due to the definition of transition.

References

1. H. Tang; E. Day; L. Kendhammer; J. N. Moore; S. A. Brown; N. J. Pienta. (2016) Eye movement patterns in solving science ordering problems. Journal of eye movement research, 9(3), 1-13.

2. J. J. Topczewski; A. M. Topczewski; H. Tang; L. Kendhammer; N. J. Pienta.(2017) NMR Spectra through the eyes of a student: eye tracking applied to NMR items. Journal of chemical education, 94(1), 29-37.

See Also

TransMx

Examples

```
# default values
strs.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
TransInfo(strs.vec)

# non-default values
str1.vec <- c("ABCABEF", "CDCDAB")
TransInfo(str1.vec, type1 = "AB", type2 = "CD")
```

TransMx *Transition matrices in one group of strings*

Description

TransMx discovers transition matrix of a string vector and the related information.

A transition is defined as a substring (in the forward order) with length of 2 characters.

Usage

```
TransMx(strings.vec, indiv = FALSE)
```

Arguments

strings.vec	String Vector. If a string has fewer than 2 characters, that string will be ignored.
indiv	Whether exports transition matrix for each string into the current directory. Default value is FALSE.

Value

The function returns a list, which contains the transition matrix, the normalized matrix, and the sorted numbers of transitions.

If indiv = TRUE, a set of mx.txt files are exported into the current directory that contain transition matrix for each string. The names of these files are the name of strings.vec appended with the orders of the strings in the string vector. If a string has fewer than 2 characters, the corresponding mx.txt file will be skipped.

Note

Strings with less than 2 characters are not included due to the definition of transition.

See Also

TransInfo

Examples

```
# simple strings
strs.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
TransMx(strs.vec)

# simple strings, export an individual transition matrix for each string
strs.vec <- c("ABCDdefABCDa", "def123DC", "123aABCD", "ACD13", "AC1ABC", "3123fe")
TransMx(strs.vec, indiv = TRUE)
```

Index