Package ‘stringdist’

March 13, 2018

Maintainer Mark van der Loo <mark.vanderloo@gmail.com>

License GPL-3

Title Approximate String Matching and String Distance Functions

LazyData no

Type Package

LazyLoad yes

Description Implements an approximate string matching version of R's native 'match' function. Can calculate various string distances based on edits (Damerau-Levenshtein, Hamming, Levenshtein, optimal sting alignment), qgrams (q-gram, cosine, jaccard distance) or heuristic metrics (Jaro, Jaro-Winkler). An implementation of soundex is provided as well. Distances can be computed between character vectors while taking proper care of encoding or between integer vectors representing generic sequences.

Version 0.9.4.7

Depends R (>= 2.15.3)

Imports parallel

URL https://github.com/markvanderloo/stringdist

BugReports https://github.com/markvanderloo/stringdist/issues

Date 2018-03-05

Suggests testthat

RoxygenNote 6.0.1

NeedsCompilation yes

Author Mark van der Loo [aut, cre],
Jan van der Laan [ctb],
R Core Team [ctb],
Nick Logan [ctb]

Repository CRAN

Date/Publication 2018-03-13 17:12:48 UTC

R topics documented:

`stringdist-package` *A package for string distance calculation and approximate string matching.*

Description

A package for string distance calculation and approximate string matching.

Introduction

The **stringdist** package offers fast and platform-independent string metrics. Its main purpose is to compute various string distances and to do approximate text matching between character vectors. As of version 0.9.3, it is also possible to compute distances between sequences represented by integer vectors.

A typical use is to match strings that are not precisely the same. For example

```
amatch(c("hello","g'day"),c("hi","hallo","ola"),maxDist=2)
```

returns `c(2,NA)` since `"hello"` matches closest with `"hallo"`, and within the maximum (optimal string alignment) distance. The second element, `"g'day"`, matches closest with `"ola"` but since the distance equals 4, no match is reported.

A second typical use is to compute string distances. For example

```
stringdist(c("g'day"),c("hi","hallo","ola"))
```

Returns `c(5,5,4)` since these are the distances between `"g'day"` and respectively `"hi"`, `"hallo"`, and `"ola"`.

A third typical use would be to compute a `dist` object. The command

```
stringdistmatrix(c("foo","bar","boo","baz"))
```

returns an object of class `dist` that can be used by clustering algorithms such as `stats::hclust`.

A fourth use is to compute string distances between general sequences, represented as integer vectors (which must be stored in a `list`):

```
seq_dist( list(c(1L,1L,2L)), list(c(1L,2L,1L),c(2L,3L,1L,2L)) )
```

The above code yields the vector `c(1,2)` (the first shorter first argument is recycled over the longer second argument)

Besides documentation for each function, the main topics documented are:

- `stringdist-metrics` – string metrics supported by the package
- `stringdist-encoding` – how encoding is handled by the package
- `stringdist-parallelization` – on multithreading

Acknowledgements

- The code for the full Damerau-Levenshtein distance was adapted from Nick Logan's public github repository.
- C code for converting UTF-8 to integer was copied from the R core for performance reasons.
- The code for soundex conversion and string similarity was kindly contributed by Jan van der Laan.

Citation

If you would like to cite this package, please cite the R Journal Paper:

- M.P.J. van der Loo (2014). The `stringdist` package for approximate string matching. R Journal 6(1) pp 111-122

Or use `citation('stringdist')` to get a bibtex item.

`amatch` *Approximate string matching*

Description

Approximate string matching equivalents of `R`'s native `match` and `%in%`.

Usage

```
amatch(x, table, nomatch = NA_integer_, matchNA = TRUE, method = c("osa",
  "lv", "dl", "hamming", "lcs", "qgram", "cosine", "jaccard", "jw", "soundex"),
  useBytes = FALSE, weight = c(d = 1, i = 1, s = 1, t = 1), maxDist = 0.1,
  q = 1, p = 0, bt = 0, nthread = getOption("sd_num_thread"))

ain(x, table, ...)
```

Arguments

`x` elements to be approximately matched: will be coerced to `character` unless it is a list consisting of `integer` vectors.

`table` lookup table for matching. Will be coerced to `character` unless it is a list consting of `integer` vectors.

`nomatch` The value to be returned when no match is found. This is coerced to integer.

`matchNA` Should `NA`'s be matched? Default behaviour mimics the behaviour of base `match`, meaning that `NA` matches `NA` (see also the note on `NA` handling below).

`method` Matching algorithm to use. See `stringdist-metrics`.

`useBytes` Perform byte-wise comparison. See `stringdist-encoding`.

`weight` For `method='osa'` or `'dl'`, the penalty for deletion, insertion, substitution and transposition, in that order. When `method='lv'`, the penalty for transposition is ignored. When `method='jw'`, the weights associated with characters of `a`, characters from `b` and the transposition weight, in that order. Weights must be positive and not exceed 1. `weight` is ignored completely when `method='hamming'`, `'qgram'`, `'cosine'`, `'Jaccard'`, `'lcs'`, or `'soundex'`.

`maxDist` Elements in `x` will not be matched with elements of `table` if their distance is larger than `maxDist`. Note that the maximum distance between strings depends on the method: it should always be specified.

`q` q-gram size, only when method is `'qgram'`, `'jaccard'`, or `'cosine'`.

`p` Winklers penalty parameter for Jaro-Winkler distance, with $0 \leq p \leq 0.25$. Only when method is `'jw'`

`bt` Winkler's boost threshold. Winkler's penalty factor is only applied when the Jaro distance is larger than `bt`. Applies only to `method='jw'` and `p>0`.

`nthread` Number of threads used by the underlying C-code. A sensible default is chosen, see `stringdist-parallelization`.

`...` parameters to pass to `amatch` (except `nomatch`)

Details

`ain` is currently defined as

```
ain(x,table,...) <- function(x,table,...) amatch(x, table, nomatch=0,...) > 0
```

Value

`amatch` returns the position of the closest match of `x` in `table`. When multiple matches with the same smallest distance metric exist, the first one is returned. `ain` returns a `logical` vector of length `length(x)` indicating wether an element of `x` approximately matches an element in `table`.

Note on `NA` handling

`R`'s native `match` function matches `NA` with `NA`. This may feel inconsistent with `R`'s usual `NA` handling, since for example `NA==NA` yields `NA` rather than `TRUE`. In most cases, one may reason about the behaviour under `NA` along the lines of "if one of the arguments is `NA`, the result shall be `NA`", simply because not all information necessary to execute the function is available. One uses special functions such as `is.na`, `is.null` *etc.* to handle special values.

The `amatch` function mimics the behaviour of `match` by default: `NA` is matched with `NA` and with nothing else. Note that this is inconsistent with the behaviour of `stringdist` since `stringdist` yields `NA` when at least one of the arguments is `NA`. The same inconsistency exists between `match` and `adist`. In `amatch` this behaviour can be controlled by setting `matchNA=FALSE`. In that case, if any of the arguments in `x` is `NA`, the `nomatch` value is returned, regardless of whether `NA` is present in `table`. In `match` the behaviour can be controlled by setting the `incomparables` option.

Examples

```
# lets see which sci-fi heroes are stringdistantly nearest
amatch("leia",c("uhura","leela"),maxDist=5)

# we can restrict the search
amatch("leia",c("uhura","leela"),maxDist=1)

# we can match each value in the find vector against values in the lookup table:
amatch(c("leia","uhura"),c("ripley","leela","scully","trinity"),maxDist=2)

# setting nomatch returns a different value when no match is found
amatch("leia",c("uhura","leela"),maxDist=1,nomatch=0)

# this is always true if maxDist is Inf
ain("leia",c("uhura","leela"),maxDist=Inf)

# Let's look in a neighbourhood of maximum 2 typo's (by default, the OSA algorithm is used)
ain("leia",c("uhura","leela"), maxDist=2)
```

phonetic *Phonetic algorithms*

Description

Translate strings to phonetic codes. Similar sounding strings should get similar or equal codes.

Usage

```
phonetic(x, method = c("soundex"), useBytes = FALSE)
```

Arguments

`x`	a character vector whose elements are phonetically encoded.
`method`	name of the algorithm used. The default is `"soundex"`.
`useBytes`	Perform byte-wise comparison. `useBytes=TRUE` is faster but may yield different results depending on character encoding. For more information see the documentation of `stringdist`.

Details

Currently, only the soundex algorithm is implemented. Note that soundex coding is only meaningful for characters in the ranges a-z and A-Z. Soundex coding of strings containing non-printable ascii or non-ascii characters may be system-dependent and should not be trusted. If non-ascii or non-printable ascii charcters are encountered, a warning is emitted.

Value

The returns value depends on the method used. However, all currently implemented methods return a character vector of the same length of the input vector. Output characters are in the system's native encoding.

References

- The Soundex algorithm implemented is the algorithm used by the National Archives. This algorithm differs slightly from the original algorithm patented by R.C. Russell (US patents 1261167 (1918) and 1435663 (1922)).

See Also

`printable_ascii`, `stringdist-package`

Examples

```
# The following examples are from The Art of Computer Programming (part III, p. 395)
# (Note that our algorithm is specified different from the one in TACP, see references.)
phonetic(c('Euler','Gauss','Hilbert','Knuth','Lloyd','Lukasiewicz','Wachs'),method='soundex')
```

---

printable_ascii *Detect the presence of non-printable or non-ascii characters*

Description

Detect the presence of non-printable or non-ascii characters

Usage

```
printable_ascii(x)
```

Arguments

x	a character vector

Details

Printable ASCII characters consist of space, `A-Z`, `a-z`, `0-9` and the characters

``! "" # $ % & ' ( ) * + , . / : ; < = > ? @ [ ] \ ^ _ ` { | } ~ -``

Note that this excludes tab (as it is a control character).

Value

A `logical` indicating which elements consist solely of printable ASCII characters.

Examples

```
# define o-umlaut
ouml <- intToUtf8("0x00F6")
x <- c("Motorhead", paste0("Mot",ouml,"rhead"))
# second element contains a non-ascii character
printable_ascii(x)

# Control characters (like carriage return) are also excluded
printable_ascii("abc\r")
```

`qgrams` *Get a table of qgram counts from one or more character vectors.*

Description

Get a table of qgram counts from one or more character vectors.

Usage

```
qgrams(..., .list = NULL, q = 1L, useBytes = FALSE,
  useNames = !useBytes)
```

Arguments

`...` any number of (named) arguments, that will be coerced to character with `as.character`.

`.list` Will be concatenated with the `...` argument(s). Useful for adding character vectors named `'q'` or `'useNames'`.

`q` size of q-gram, must be non-negative.

`useBytes` Determine byte-wise qgrams. `useBytes=TRUE` is faster but may yield different results depending on character encoding. For `ASCII` it is identical. See also `stringdist` under Encoding issues.

`useNames` Add q-grams as column names. If `useBytes=useNames=TRUE`, the q-byte sequences are represented as 2 hexadecimal numbers per byte, separated by a vertical bar (`|`).

Value

A table with q-gram counts. Detected q-grams are column names and the argument names as row names. If no argument names were provided, they will be generated.

Details

The input is converted to `character`. If `useBytes=TRUE`, each element is converted to `utf8` and then to `integer` as in `stringdist`. Next,the data is passed to the underlying routine.

Strings with less than `q` characters and elements containing `NA` are skipped. Using `q=0` therefore counts the number of empty strings `""` occuring in each argument.

See Also

`stringdist`, `amatch`

Examples

```
qgrams('hello world',q=3)

# q-grams are counted uniquely over a character vector
qgrams(rep('hello world',2),q=3)

# to count them separately, do something like
x <- c('hello', 'world')
lapply(x,qgrams, q=3)

# output rows may be named, and you can pass any number of character vectors
x <- "I will not buy this record, it is scratched"
y <- "My hovercraft is full of eels"
z <- c("this", "is", "a", "dead","parrot")
qgrams(A = x, B = y, C = z,q=2)

# a tonque twister, showing the effects of useBytes and useNames
x <- "peter piper picked a peck of pickled peppers"
qgrams(x, q=2)
qgrams(x, q=2, useNames=FALSE)
qgrams(x, q=2, useBytes=TRUE)
qgrams(x, q=2, useBytes=TRUE, useNames=TRUE)
```

`seq_amatch` *Approximate matching for integer sequences.*

Description

For a list of integer vectors x, find the closest matches in a list of integer or numeric vectors in table.

Usage

```
seq_amatch(x, table, nomatch = NA_integer_, matchNA = TRUE,
  method = c(”osa”, ”lv”, ”dl”, ”hamming”, ”lcs”, ”qgram”, ”cosine”,
  ”jaccard”, ”jw”), weight = c(d = 1, i = 1, s = 1, t = 1), maxDist = 0.1,
  q = 1, p = 0, bt = 0, nthread = getOption(”sd_num_thread”))

seq_ain(x, table, ...)
```

Arguments

x — (list of) integer or numeric vector(s) to be approximately matched. Will be converted with as.integer.

table — (list of) integer or numeric vector(s) serving as lookup table for matching. Will be converted with as.integer.

nomatch — The value to be returned when no match is found. This is coerced to integer.

matchNA — Should NA's be matched? Default behaviour mimics the behaviour of base match, meaning that NA matches NA. With NA, we mean a missing entry in the list, represented as NA_integer_. If one of the integer sequences stored in the list has an NA entry, this is just treated as another integer (the representation of NA_integer_).

method — Matching algorithm to use. See stringdist-metrics.

weight — For method='osa' or 'dl', the penalty for deletion, insertion, substitution and transposition, in that order. When method='lv', the penalty for transposition is ignored. When method='jw', the weights associated with integers in elements of a, integers in elements of b and the transposition weight, in that order. Weights must be positive and not exceed 1. weight is ignored completely when method='hamming', 'qgram', 'cosine', 'Jaccard', or 'lcs'.

maxDist — Elements in x will not be matched with elements of table if their distance is larger than maxDist. Note that the maximum distance between strings depends on the method: it should always be specified.

q — q-gram size, only when method is 'qgram', 'jaccard', or 'cosine'.

p — Winklers penalty parameter for Jaro-Winkler distance, with $0 \leq p \leq 0.25$. Only when method is 'jw'

bt — Winkler's boost threshold. Winkler's penalty factor is only applied when the Jaro distance is larger than bt. Applies only to method='jw' and p>0.

nthread — Number of threads used by the underlying C-code. A sensible default is chosen, see stringdist-parallelization.

... — parameters to pass to seq_amatch (except nomatch)

Value

seq_amatch returns the position of the closest match of x in table. When multiple matches with the same minimal distance metric exist, the first one is returned. seq_ain returns a logical vector of length length(x) indicating wether an element of x approximately matches an element in table.

Notes

seq_ain is currently defined as

seq_ain(x,table,...) <- function(x,table,...) amatch(x, table, nomatch=0,...) > 0

All input vectors are converted with as.integer. This causes truncation for numeric vectors (e.g. pi will be treated as 3L).

See Also

seq_dist, seq_sim, seq_qgrams

Examples

```
x <- list(1:3,c(3:1),c(1L,3L,4L))
table <- list(
  c(5L,3L,1L,2L)
  ,1:4
)
seq_amatch(x,table,maxDist=2)

# behaviour with missings
seq_amatch(list(c(1L,NA_integer_,3L),NA_integer_), list(1:3),maxDist=1)


## Not run:
# Match sentences based on word order. Note: words must match exactly or they
# are treated as completely different.
#
# For this example you need to have the 'hashr' package installed.
x <- "Mary had a little lamb"
x.words <- strsplit(x,"[[:blank:]]+")
x.int <- hashr::hash(x.words)
table <- c("a little lamb had Mary",
           "had Mary a little lamb")
table.int <- hashr::hash(strsplit(table,"[[:blank:]]+"))
seq_amatch(x.int,table.int,maxDist=3)

## End(Not run)
```

seq_dist *Compute distance metrics between integer sequences*

Description

seq_dist computes pairwise string distances between elements of a and b, where the argument with less elements is recycled. seq_distmatrix computes the distance matrix with rows according to a and columns according to b.

Usage

```
seq_dist(a, b, method = c("osa", "lv", "dl", "hamming", "lcs", "qgram",
  "cosine", "jaccard", "jw"), weight = c(d = 1, i = 1, s = 1, t = 1), q = 1,
  p = 0, bt = 0, nthread = getOption("sd_num_thread"))

seq_distmatrix(a, b, method = c("osa", "lv", "dl", "hamming", "lcs", "qgram",
  "cosine", "jaccard", "jw"), weight = c(d = 1, i = 1, s = 1, t = 1), q = 1,
  p = 0, bt = 0, useNames = c("names", "none"),
  nthread = getOption("sd_num_thread"))
```

Arguments

a (list of) integer or numeric vector(s). Will be converted with as.integer (target)

b (list of) integer or numeric vector(s). Will be converted with as.integer (source). Optional for seq_distmatrix.

method Distance metric. See stringdist-metrics

weight For method='osa' or 'dl', the penalty for deletion, insertion, substitution and transposition, in that order. When method='lv', the penalty for transposition is ignored. When method='jw', the weights associated with characters of a, characters from b and the transposition weight, in that order. Weights must be positive and not exceed 1. weight is ignored completely when method='hamming', 'qgram', 'cosine', 'Jaccard', or 'lcs'

q Size of the q-gram; must be nonnegative. Only applies to method='qgram', 'jaccard' or 'cosine'.

p Penalty factor for Jaro-Winkler distance. The valid range for p is 0 <= p <= 0.25. If p=0 (default), the Jaro-distance is returned. Applies only to method='jw'.

bt Winkler's boost threshold. Winkler's penalty factor is only applied when the Jaro distance is larger than bt Applies only to method='jw' and p>0.

nthread Maximum number of threads to use. By default, a sensible number of threads is chosen, see stringdist-parallelization.

useNames label the output matrix with names(a) and names(b)?

Value

seq_dist returns a numeric vector with pairwise distances between a and b of length max(length(a),length(b).

For seq_distmatrix there are two options. If b is missing, the dist object corresponding to the length(a) X length(a) distance matrix is returned. If b is specified, the length(a) X length(b) distance matrix is returned.

If any element of a or b is NA_integer_, the distance with any matched integer vector will result in NA. Missing values in the sequences themselves are treated as a number and not treated specially (Also see the examples).

Notes

Input vectors are converted with as.integer. This causes truncation for numeric vectors (e.g. pi will be treated as 3L).

See Also

seq_sim, seq_amatch, seq_qgrams

Examples

```
# Distances between lists of integer vectors. Note the postfix 'L' to force
# integer storage. The shorter argument is recycled over (\code{a})
a <- list(c(102L, 107L))                        # fu
b <- list(c(102L,111L,111L),c(102L,111L,111L))  # foo, fo
seq_dist(a,b)

# translate strings to a list of integer sequences
a <- lapply(c("foo","bar","baz"),utf8ToInt)
seq_distmatrix(a)

# Note how missing values are treated. NA's as part of the sequence are treated
# as an integer (the representation of NA_integer_).
a <- list(NA_integer_,c(102L, 107L))
b <- list(c(102L,111L,111L),c(102L,111L,NA_integer_))
seq_dist(a,b)

## Not run:
# Distance between sentences based on word order. Note: words must match exactly or they
# are treated as completely different.
#
# For this example you need to have the 'hashr' package installed.
a <- "Mary had a little lamb"
a.words <- strsplit(a,"[[:blank:]]+")
a.int <- hashr::hash(a.words)
b <- c("a little lamb had Mary",
           "had Mary a little lamb")
b.int <- hashr::hash(strsplit(b,"[[:blank:]]+"))
seq_dist(a.int,b.int)

## End(Not run)
```

seq_qgrams *Get a table of qgram counts for integer sequences*

Description

Get a table of qgram counts for integer sequences

Usage

```
seq_qgrams(..., .list = NULL, q = 1L)
```

Arguments

`...`	Any number of (named) arguments that will be coerced with `as.integer`
`.list`	Will be concatenated with the `...` argument(s). Useful for adding integer vectors named 'q'.
`q`	The size of q-gramming.

Value

A `matrix` containing q-gram profiles. Columns 1 to q contain the encountered q-grams. The ensuing (named) columns contain the q-gram counts per vector. Run the example for a simple overview.

Missing values in integer sequences are treated as any other number.

See Also

`seq_dist`, `seq_amatch`

Examples

```

# compare the 2-gram overlap between sequences 1:3 and 2:4
seq_qgrams(x = 1:3, y=2:4,q=2)

# behavior when NA's are present.
seq_qgrams(1:3,c(1,NA,2),NA_integer_)
```

seq_sim *Compute similarity scores between sequences of integers*

Description

Compute similarity scores between sequences of integers

Usage

```
seq_sim(a, b, method = c("osa", "lv", "dl", "hamming", "lcs", "qgram",
  "cosine", "jaccard", "jw"), q = 1, ...)
```

Arguments

a	list of integer vectors (target)
b	list of integer vectors (source). Optional for seq_distmatrix.
method	Method for distance calculation. The default is "osa", see stringdist-metrics.
q	Size of the q-gram; must be nonnegative. Only applies to method='qgram', 'jaccard' or 'cosine'.
...	additional arguments are passed on to seq_dist.

Value

A numeric vector of length max(length(a),length(b)). If one of the entries in a or b is NA_integer_, all comparisons with that element result in NA. Missings occurring within the sequences are treated as an ordinary number (the representation of NA_integer_).

See Also

seq_dist, seq_amatch

Examples

```
L1 <- list(1:3,2:4)
L2 <- list(1:3)
seq_sim(L1,L2,method="osa")

# note how missing values are handled (L2 is recycled over L1)
L1 <- list(c(1L,NA_integer_,3L),2:4,NA_integer_)
L2 <- list(1:3)
seq_sim(L1,L2)
```

stringdist — *Compute distance metrics between strings*

Description

`stringdist` computes pairwise string distances between elements of `a` and `b`, where the argument with less elements is recycled. `stringdistmatrix` computes the string distance matrix with rows according to `a` and columns according to `b`.

Usage

```
stringdist(a, b, method = c("osa", "lv", "dl", "hamming", "lcs", "qgram",
  "cosine", "jaccard", "jw", "soundex"), useBytes = FALSE, weight = c(d = 1,
  i = 1, s = 1, t = 1), maxDist = Inf, q = 1, p = 0, bt = 0,
  nthread = getOption("sd_num_thread"))

stringdistmatrix(a, b, method = c("osa", "lv", "dl", "hamming", "lcs",
  "qgram", "cosine", "jaccard", "jw", "soundex"), useBytes = FALSE,
  weight = c(d = 1, i = 1, s = 1, t = 1), maxDist = Inf, q = 1, p = 0,
  bt = 0, useNames = c("none", "strings", "names"), ncores = 1,
  cluster = NULL, nthread = getOption("sd_num_thread"))
```

Arguments

`a` R object (target); will be converted by `as.character`

`b` R object (source); will be converted by `as.character` This argument is optional for `stringdistmatrix` (see section `Value`).

`method` Method for distance calculation. The default is `"osa"`, see `stringdist-metrics`.

`useBytes` Perform byte-wise comparison, see `stringdist-encoding`.

`weight` For `method='osa'` or `'dl'`, the penalty for deletion, insertion, substitution and transposition, in that order. When `method='lv'`, the penalty for transposition is ignored. When `method='jw'`, the weights associated with characters of `a`, characters from `b` and the transposition weight, in that order. Weights must be positive and not exceed 1. `weight` is ignored completely when `method='hamming'`, `'qgram'`, `'cosine'`, `'Jaccard'`, `'lcs'`, or `soundex`.

`maxDist` [DEPRECATED AND WILL BE REMOVED|2016] Currently kept for backward compatibility. It does not offer any speed gain. (In fact, it currently slows things down when set to anything different from `Inf`).

`q` Size of the q-gram; must be nonnegative. Only applies to `method='qgram'`, `'jaccard'` or `'cosine'`.

`p` Penalty factor for Jaro-Winkler distance. The valid range for `p` is `0 <= p <= 0.25`. If `p=0` (default), the Jaro-distance is returned. Applies only to `method='jw'`.

`bt` Winkler's boost threshold. Winkler's penalty factor is only applied when the Jaro distance is larger than `bt`. Applies only to `method='jw'` and `p>0`.

`nthread` Maximum number of threads to use. By default, a sensible number of threads is chosen, see `stringdist-parallelization`.

`useNames` Use input vectors as row and column names?

`ncores` [DEPRECATED AND WILL BE REMOVED|2016]. Use `nthread` in stead. This argument is ignored.

`cluster` [DEPRECATED AND WILL BE REMOVED|2016]. A custom cluster, created with `makeCluster`.

Value

For `stringdist`, a vector with string distances of size `max(length(a),length(b))`.

For `stringdistmatrix`: if both `a` and `b` are passed, a `length(a)xlength(b) matrix`. If a single argument `a` is given an object of class `dist` is returned.

Distances are nonnegative if they can be computed, `NA` if any of the two argument strings is `NA` and `Inf` when `maxDist` is exceeded or, in case of the hamming distance, when the two compared strings have different length.

See Also

`stringsim`, `qgrams`, `amatch`

Examples

```
# Simple example using optimal string alignment
stringdist("ca","abc")

# computing a 'dist' object
d <- stringdistmatrix(c('foo','bar','boo','baz'))
# try plot(hclust(d))

# The following gives a matrix
stringdistmatrix(c("foo","bar","boo"),c("baz","buz"))

# An example using Damerau-Levenshtein distance (multiple editing of substrings allowed)
stringdist("ca","abc",method="dl")

# string distance matching is case sensitive:
stringdist("ABC","abc")

# so you may want to normalize a bit:
stringdist(tolower("ABC"),"abc")

# stringdist recycles the shortest argument:
stringdist(c('a','b','c'),c('a','c'))

# stringdistmatrix gives the distance matrix (by default for optimal string alignment):
stringdist(c('a','b','c'),c('a','c'))

# different edit operations may be weighted; e.g. weighted substitution:
```

```
stringdist('ab','ba',weight=c(1,1,1,0.5))

# Non-unit weights for insertion and deletion makes the distance metric asymetric
stringdist('ca','abc')
stringdist('abc','ca')
stringdist('ca','abc',weight=c(0.5,1,1,1))
stringdist('abc','ca',weight=c(0.5,1,1,1))

# Hamming distance is undefined for
# strings of unequal lengths so stringdist returns Inf
stringdist(”ab”,”abc”,method=”h”)
# For strings of eqal length it counts the number of unequal characters as they occur
# in the strings from beginning to end
stringdist(”hello”,”HeLl0”,method=”h”)

# The lcs (longest common substring) distance returns the number of
# characters that are not part of the lcs.
#
# Here, the lcs is either 'a' or 'b' and one character cannot be paired:
stringdist('ab','ba',method=”lcs”)
# Here the lcs is 'surey' and 'v', 'g' and one 'r' of 'surgery' are not paired
stringdist('survey','surgery',method=”lcs”)


# q-grams are based on the difference between occurrences of q consecutive characters
# in string a and string b.
# Since each character abc occurs in 'abc' and 'cba', the q=1 distance equals 0:
stringdist('abc','cba',method='qgram',q=1)

# since the first string consists of 'ab','bc' and the second
# of 'cb' and 'ba', the q=2 distance equals 4 (they have no q=2 grams in common):
stringdist('abc','cba',method='qgram',q=2)

# Wikipedia has the following example of the Jaro-distance.
stringdist('MARTHA','MATHRA',method='jw')
# Note that stringdist gives a  _distance_ where wikipedia gives the corresponding
# _similarity measure_. To get the wikipedia result:
1 - stringdist('MARTHA','MATHRA',method='jw')

# The corresponding Jaro-Winkler distance can be computed by setting p=0.1
stringdist('MARTHA','MATHRA',method='jw',p=0.1)
# or, as a similarity measure
1 - stringdist('MARTHA','MATHRA',method='jw',p=0.1)

# This gives distance 1 since Euler and Gauss translate to different soundex codes.
stringdist('Euler','Gauss',method='soundex')
# Euler and Ellery translate to the same code and have distance 0
stringdist('Euler','Ellery',method='soundex')
```

stringdist-encoding *String metrics in* **stringdist**

Description

This page gives an overview of encoding handling in **stringst**.

Encoding in stringdist

All character strings are stored as a sequence of bytes. An encoding system relates a byte, or a short sequence of bytes to a symbol. Over the years, many encoding systems have been developed, and not all OS's and softwares use the same encoding as default. Similarly, depending on the system R is running on, R may use a different encoding for storing strings internally.

The **stringdist** package is designed so users in principle need not worry about this. Strings are converted to `UTF-32` (unsigned integer) by default prior to any further computation. This means that results are encoding-independent and that strings are interpreted as a sequence of symbols, not as a sequence of pure bytes. In functions where this is relevant, this may be switched by setting the `useBytes` option to `TRUE`. However, keep in mind that results will then likely depend on the system R is running on, except when your strings are pure ASCII. Also, for multi-byte encodings, results for byte-wise computations will usually differ from results using encoded computations.

Prior to **stringdist** version 0.9, setting `useBytes=TRUE` could give a significant performance enhancement. Since version 0.9, translation to integer is done by C code internal to **stringdist** and the difference in performance is now negligible.

Unicode normalisation

In `utf-8`, the same (accented) character may be represented as several byte sequences. For example, an u-umlaut can be represented with a single byte code or as a byte code representing `'u'` followed by a modifier byte code that adds the umlaut. The stringi package of Gagolevski and Tartanus offers unicode normalisation tools.

Some tips on character encoding and transliteration

Some algorithms (like soundex) are defined only on the printable ASCII character set. This excludes any character with accents for example. Translating accented characters to the non-accented ones is a form of transliteration. On many systems running R (but not all!) you can achieve this with

`iconv(x,to="ASCII//TRANSLIT")`,

where `x` is your character vector. See the documentation of `iconv` for details.

The `stringi` package (Gagolewski and Tartanus) should work on any system. The command `stringi::stri_trans_general(x,"Latin-ASCII")` transliterates character vector `x` to ASCII.

References

- The help page of `Encoding` describes how R handles encoding.

- The help page of iconv has a good overview of base R's encoding conversion options. The capabilities of iconv depend on the system R is running on. The **stringi** package offers platform-independent encoding and normalization tools.

See Also

- Functions using re-encoding: stringdist, stringdistmatrix, amatch, ain, qgrams
- Encoding related: printable_ascii

stringdist-metrics *String metrics in* **stringdist**

Description

This page gives an overview of the string dissimilarity measures offered by **stringdist**.

String Metrics

String metrics are ways of quantifying the dissimilarity between two finite sequences, usually text strings. Over the years, many such measures have been developed. Some are based on a mathematical understanding of the set of all strings that can be composed from a finite alphabet, others are based on more heuristic principles, such as how a text string sounds when pronounced by a native English speaker.

The terms 'string metrics' and 'string distance' are used more or less interchangibly in literature. From a mathematical point of view, string metrics often do not obey the demands that are usually required from a distance function. For example, it is not true for all string metrics that a distance of 0 means that two strings are the same (e.g. in the q-gram distance). Nevertheless, string metrics are very useful in practice and have many applications.

The metric you need to choose for an application strongly depends on both the nature of the string (what does the string represent?) and the cause of dissimilarities between the strings you are measuring. For example, if you are comparing human-typed names that may contain typo's, the Jaro-Winkler distance may be of use. If you are comparing names that were written down after hearing them, a phonetic distance may be a better choice.

Currently, the following distance metrics are supported by **stringdist**.

Method name	**Description**
osa	Optimal string aligment, (restricted Damerau-Levenshtein distance).
lv	Levenshtein distance (as in R's native adist).
dl	Full Damerau-Levenshtein distance.
hamming	Hamming distance (a and b must have same nr of characters).
lcs	Longest common substring distance.
qgram	q-gram distance.
cosine	cosine distance between q-gram profiles
jaccard	Jaccard distance between q-gram profiles
jw	Jaro, or Jaro-Winker distance.
soundex	Distance based on soundex encoding (see below)

A short description of string metrics supported by stringdist

See Van der Loo (2014) for an extensive description and references. The review papers of Navarro (2001) and Boytsov (2011) provide excellent technical overviews of respectively online and offline string matching algorithms.

The **Hamming distance** (`method='hamming'`) counts the number of character substitutions that turns `b` into `a`. If `a` and `b` have different number of characters the distance is `Inf`.

The **Levenshtein distance** (`method='lv'`) counts the number of deletions, insertions and substitutions necessary to turn `b` into `a`. This method is equivalent to `R`'s native `adist` function.

The **Optimal String Alignment distance** (`method='osa'`) is like the Levenshtein distance but also allows transposition of adjacent characters. Here, each substring may be edited only once. (For example, a character cannot be transposed twice to move it forward in the string).

The **full Damerau-Levenshtein distance** (`method='dl'`) is like the optimal string alignment distance except that it allows for multiple edits on substrings.

The **longest common substring** (method='lcs') is defined as the longest string that can be obtained by pairing characters from `a` and `b` while keeping the order of characters intact. The **lcs-distance** is defined as the number of unpaired characters. The distance is equivalent to the edit distance allowing only deletions and insertions, each with weight one.

A q**-gram** (method='qgram') is a subsequence of q *consecutive* characters of a string. If x (y) is the vector of counts of q-gram occurrences in `a` (`b`), the q**-gram distance** is given by the sum over the absolute differences $|x_i - y_i|$. The computation is aborted when `q` is is larger than the length of any of the strings. In that case `Inf` is returned.

The **cosine distance** (method='cosine') is computed as $1 - x \cdot y/(\|x\|\|y\|)$, where x and y were defined above.

Let X be the set of unique q-grams in `a` and Y the set of unique q-grams in `b`. The **Jaccard distance** (`method='jaccard'`) is given by $1 - |X \cap Y|/|X \cup Y|$.

The **Jaro distance** (`method='jw'`, `p=0`), is a number between 0 (exact match) and 1 (completely dissimilar) measuring dissimilarity between strings. It is defined to be 0 when both strings have length 0, and 1 when there are no character matches between `a` and `b`. Otherwise, the Jaro distance is defined as $1 - (1/3)(w_1 m/|a| + w_2 m/|b| + w_3(m - t)/m)$. Here,$|a|$ indicates the number of characters in `a`, m is the number of character matches and t the number of transpositions of matching characters. The w_i are weights associated with the characters in `a`, characters in `b` and with transpositions. A character c of `a` *matches* a character from `b` when c occurs in `b`, and the index of c in `a` differs less than $\max(|a|, |b|)/2 - 1$ (where we use integer division) from the index of c in `b`. Two matching characters are transposed when they are matched but they occur in different order in string `a` and `b`.

The **Jaro-Winkler distance** (`method=jw`, `0<p<=0.25`) adds a correction term to the Jaro-distance. It is defined as $d - l \cdot p \cdot d$, where d is the Jaro-distance. Here, l is obtained by counting, from the start of the input strings, after how many characters the first character mismatch between the two strings occurs, with a maximum of four. The factor p is a penalty factor, which in the work of Winkler is often chosen 0.1.

For the **soundex** distance (method='soundex'), strings are translated to a soundex code (see `phonetic` for a specification). The distance between strings is 0 when they have the same soundex code, otherwise 1. Note that soundex recoding is only meaningful for characters in the ranges a-z and A-Z. A warning is emitted when non-printable or non-ascii characters are encountered. Also see `printable_ascii`.

References

- MPJ van der Loo (2014) *The stringdist package for approximate string matching*. The R Journal **6**(1) 111-122.
- L. Boytsov (2011). *Indexing methods for approximate dictionary searching: comparative analyses*. ACM Journal of experimental algorithmics **16** 1-88.
- G. Navarro (2001). *A guided tour to approximate string matching*. ACM Computing Surveys **33** 31-88.

See Also

- Functions applying string metrics to text: `stringdist`, `stringdistmatrix`, `amatch`
- Functions applying string metrics to integer sequences: `seq_dist`, `seq_distmatrix`, `seq_amatch`
- Encoding issues: `stringdist-encoding`

`stringdist-parallelization`

Multithreading and parallelization in **stringdist**

Description

This page describes how **stringdist** uses parallel processing.

Multithreading and parallelization in stringdist

The core functions of **stringdist** are implemented in C. On systems where `openMP` is available, **stringdist** will automatically take advantage of multiple cores. The section on OpenMP of the Writing R Extensions manual discusses on what systems OpenMP is available (at the time of writing more or less, anywhere except on OSX).

By default, the number of threads to use is taken from `options('sd_num_thread')`. When the package is loaded, the value for this option is determined as follows:

- If the environment variable `OMP_NUM_THREADS` is set, this value is taken.
- Otherwise, the number of available cores is determined with `parallel::detectCores()` If this fails, the number of threads is set to 1 (with a message). If the nr of detected cores exceeds three, the number of used cores is set to $n - 1$.
- If available, the environment variable `OMP_THREAD_LIMIT` is determined and The number of threads is set to the lesser of `OMP_THREAD_LIMIT` and the number of detected cores.

The latter step makes sure that on machines with $n > 3$ cores, $n - 1$ cores are used. Some benchmarking showed that using all cores is often slower in such cases. This is probably because at least one of the threads will be shared with the operating system.

Functions that use multithreading have an option named `nthread` that controls the maximum number of threads to use. If you need to do large calculations, it is probably a good idea to benchmark the performance on your machine(s) as a function of `'nthread'`, for example using the microbenchmark package of Mersmann.

See Also

- Functions running multithreaded: stringdist, stringdistmatrix, amatch, ain

stringsim *Compute similarity scores between strings*

Description

stringsim computes pairwise string similarities between elements of character vectors a and b, where the vector with less elements is recycled.

Usage

```
stringsim(a, b, method = c("osa", "lv", "dl", "hamming", "lcs", "qgram",
  "cosine", "jaccard", "jw", "soundex"), useBytes = FALSE, q = 1, ...)
```

Arguments

a	R object (target); will be converted by as.character.
b	R object (source); will be converted by as.character.
method	Method for distance calculation. The default is "osa", see stringdist-metrics.
useBytes	Perform byte-wise comparison, see stringdist-encoding.
q	Size of the q-gram; must be nonnegative. Only applies to method='qgram', 'jaccard' or 'cosine'.
...	additional arguments are passed on to stringdist.

Details

The similarity is calculated by first calculating the distance using stringdist, dividing the distance by the maximum possible distance, and substracting the result from 1. This results in a score between 0 and 1, with 1 corresponding to complete similarity and 0 to complete dissimilarity. Note that complete similarity only means equality for distances satisfying the identity property. This is not the case e.g. for q-gram based distances (for example if q=1, anagrams are completely similar). For distances where weights can be specified, the maximum distance is currently computed by assuming that all weights are equal to 1.

Value

Returns a vector with similarities, which are values between 0 and 1 where 1 corresponds to perfect similarity (distance 0) and 0 to complete dissimilarity. NA is returned when stringdist returns NA. Distances equal to Inf are truncated to a similarity of 0.

Examples

```
# Calculate the similarity using the default method of optimal string alignment
stringsim("ca", "abc")

# Calculate the similarity using the Jaro-Winkler method
# The p argument is passed on to stringdist
stringsim('MARTHA','MATHRA',method='jw', p=0.1)
```

Index