Package 'stringdist'

August 8, 2014

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.

Version 0.8.0

Depends R (>= 2.15.3), parallel

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

Date 2013-04-28

Suggests testthat

Author Mark van der Loo [aut, cre],Jan van der Laan [ctb]

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-08-08 15:21:30

R topics documented:

`stringdist-package` *A package for string distance calculation*

Description

A package for string distance calculation

`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)

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

Arguments

`x`	vector: elements to be approximately matched: will be coerced to `character`.
`table`	vector: lookup table for matching. Will be coerced to `character`.
`nomatch`	The value to be returned when no match is found. This is coerced to integer. `nomatch=0` can be a useful option.
`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`.

useBytes	Perform byte-wise comparison. useBytes=TRUE is faster but may yield different results depending on character encoding. See also stringdist, under encoding issues.
weight	Weight parameters for matching algorithm See stringdist.
maxDist	Elements in x will not be matched with elements of table if their distance is larger than maxDist.
q	q-gram size, see stringdist.
p	Winklers penalty parameter for Jaro-Winkler distance, see stringdist.
...	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)

# 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.

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.

References

- The Soudex 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

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.

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 most systems running R you can achieve this with

`iconv(x,"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.

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

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)
```

`stringdist` *Compute distance metrics between strings*

Description

Compute distance metrics between strings

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)

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,
  ncores = 1, cluster = NULL)
```

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 details).
useBytes	Perform byte-wise comparison. `useBytes=TRUE` is faster but may yield different results depending on character encoding. See also below, under "encoding issues".
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'`.
maxDist	[DEPRECATED AND WILL BE REMOVED] 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'`.
ncores	Number of cores to use. If `ncores>1`, a local cluster is created using `makeCluster`. Parallelisation is over `b`, so the speed gain by parallelisation is highest when `b` has less elements than `a`.
cluster	(Optional) a custom cluster, created with `makeCluster`. If `cluster` is not `NULL`, `ncores` is ignored.

Value

For `stringdist`, a vector with string distances of size `max(length(a),length(b))`. For `stringdistmatrix`, a `length(a)xlength(b) matrix`. The returned distance is nonnegative if it can be computed, `NA` if any of the two argument strings is `NA` and `Inf` when it cannot be computed or `maxDist` is exceeded. See details for the meaning of `Inf` for the various algorithms.

Details

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

`stringdistmatrix` computes the string distance matrix with rows according to `a` and columns according to `b`.

Currently, the following distance metrics are supported:

`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)

Precise descriptions of the algorithms are given in the R-journal paper (see Citation section). Below are some concise descriptions.

The **Hamming distance** (`hamming`) counts the number of character substitutions that turns `b` into `a`. If `a` and `b` have different number of characters or if `maxDist` is exceeded, `Inf` is returned.

The **Levenshtein distance** (`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. If `maxDist` is exceeded `Inf` is returned.

The **Optimal String Alignment distance** (`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). If `maxDist` is exceeded `Inf` is returned.

The **full Damerau-Levensthein distance** (`dl`) allows for multiple edits on substrings. If `maxDist` is exceeded `Inf` is returned.

The **longest common substring** 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. If `maxDist` is exceeded `Inf` is returned.

A **q-gram** 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** 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** 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 $d-l*p*d$, 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** method, 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`.

Encoding issues

If `bytes=FALSE`, input strings are re-encoded to `utf8` an then to `integer` vectors prior to the distance calculation (since the underlying `C`-code expects `unsigned ints`). This double conversion is necessary as it seems the only way to reliably convert (possibly multibyte) characters to integers on all systems supported by `R`. `R`'s native `adist` function does this as well.

If `bytes=TRUE`, the input strings are treated as if each byte was a single character. This may be significantly faster since it avoids conversion of `utf8` to integer with `utf8ToInt` (up to a factor of 3, for strings of 5-25 characters). However, results may depend on the (possibly multibyte) character encoding scheme and note that `R`'s internal encoding scheme is OS-dependent. If you're sure that all your input is `ASCII`, you can safely set `useBytes=TRUE` to profit from the speed gain on any platform.

See base `R`'s `Encoding` and `iconv` documentation for details on how `R` handles character encoding.

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.

Paralellization

The `stringdistmatrix` function uses `makeCluster` to create a local cluster and compute the distance matrix in parallel when `ncores>1`. The cluster is terminated after the matrix has been computed. As the cluster is local, the `ncores` parameter should not be larger than the number of cores on your machine. Use `detectCores` to check the number of cores available. Alternatively, you can create a cluster using `makeCluster` and pass that to `stringdistmatrix` (through the `cluster` argument. This allows you to reuse the cluster setup for other calculations. There is overhead in creat-

ing clusters, so creating the cluster yourself is a good choice if you want to call `stringdistmatrix` multiple times, for example in a loop.

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.

References

- R.W. Hamming (1950). Error detecting and Error Correcting codes, The Bell System Technical Journal 29, 147-160
- V.I. Levenshtein. (1960). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 707-711.
- F.J. Damerau (1964) A technique for computer detection and correction of spelling errors. Communications of the ACM 7 171-176.
- An extensive overview of offline string matching algorithms is given by L. Boytsov (2011). Indexing methods for approximate dictionary searching: comparative analyses. ACM Journal of experimental algorithmics 16 1-88.
- An extensive overview of (online) string matching algorithms is given by G. Navarro (2001). A guided tour to approximate string matching, ACM Computing Surveys 33 31-88.
- Many algorithms are available in pseudocode from wikipedia: http://en.wikipedia.org/wiki/Damerau-Levenshtein_distance.
- The code for the full Damerau-Levenshtein distance was adapted from Nick Logan's public github repository.
- A good reference for qgram distances is E. Ukkonen (1992), Approximate string matching with q-grams and maximal matches. Theoretical Computer Science, 92, 191-211.
- Wikipedia describes the Jaro-Winker distance used in this package. Unfortunately, there seems to be no single definition for the Jaro distance in literature. For example Cohen, Ravikumar and Fienberg (Proceeedings of IIWEB03, Vol 47, 2003) report a different matching window for characters in strings `a` and `b`.
- Raffael Vogler wrote a nice blog comparing different string distances in this package.

Examples

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

# The same 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 lcm (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')
```

Index