Package 'RapidFuzz'

December 5, 2024

Type Package

Title String Similarity Computation Using 'RapidFuzz'

Version 1.0

Date 2024-11-30

Description Provides a high-performance interface for calculating string similarities and distances, leveraging the efficient library 'RapidFuzz' <https://github.com/rapidfuzz/rapidfuzz-cpp>. This package integrates the 'C++' implementation, allowing 'R' users to access cutting-edge algorithms for fuzzy matching and text analysis.

URL <https://github.com/StrategicProjects/RapidFuzz>

Note This package makes use of the 'RapidFuzz' source code created by Max Bachmann and Adam Cohen (<https://github.com/rapidfuzz/rapidfuzz-cpp>).

License MIT + file LICENSE

Encoding UTF-8

Imports Rcpp (>= 1.0.13)

LinkingTo Rcpp

RoxygenNote 7.3.2

SystemRequirements C++17

NeedsCompilation yes

Author Andre Leite [aut, cre],
Hugo Vaconcelos [aut],
Max Bachmann [ctb],
Adam Cohen [ctb]

Maintainer Andre Leite <leite@castlab.org>

Repository CRAN

Date/Publication 2024-12-05 18:40:14 UTC

Contents

`damerau_levenshtein_distance` *Damerau-Levenshtein Distance*

Description

Calculate the Damerau-Levenshtein distance between two strings.

Computes the Damerau-Levenshtein distance, which is an edit distance allowing transpositions in addition to substitutions, insertions, and deletions.

Usage

```
damerau_levenshtein_distance(s1, s2, score_cutoff = NULL)
```

Arguments

`s1`	A string. The first input string.
`s2`	A string. The second input string.
`score_cutoff`	An optional maximum threshold for the distance. Defaults to the largest integer value in R (‘.Machine$integer.max‘).

Value

The Damerau-Levenshtein distance as an integer.

Examples

```
damerau_levenshtein_distance("abcdef", "abcfde")
damerau_levenshtein_distance("abcdef", "abcfde", score_cutoff = 3)
```

damerau_levenshtein_normalized_distance

Normalized Damerau-Levenshtein Distance

Description

Calculate the normalized Damerau-Levenshtein distance between two strings.

Computes the normalized Damerau-Levenshtein distance, where the result is between 0.0 (identical) and 1.0 (completely different).

Usage

```
damerau_levenshtein_normalized_distance(s1, s2, score_cutoff = 1)
```

Arguments

s1	A string. The first input string.
s2	A string. The second input string.
score_cutoff	An optional maximum threshold for the normalized distance. Defaults to 1.0.

Value

The normalized Damerau-Levenshtein distance as a double.

Examples

```
damerau_levenshtein_normalized_distance("abcdef", "abcfde")
damerau_levenshtein_normalized_distance("abcdef", "abcfde", score_cutoff = 0.5)
```

damerau_levenshtein_normalized_similarity

Normalized Damerau-Levenshtein Similarity

Description

Calculate the normalized Damerau-Levenshtein similarity between two strings.

Computes the normalized similarity based on the Damerau-Levenshtein metric, where the result is between 0.0 (completely different) and 1.0 (identical).

Usage

```
damerau_levenshtein_normalized_similarity(s1, s2, score_cutoff = 0)
```

Arguments

s1	A string. The first input string.
s2	A string. The second input string.
score_cutoff	An optional minimum threshold for the normalized similarity. Defaults to 0.0.

Value

The normalized Damerau-Levenshtein similarity as a double.

Examples

```
damerau_levenshtein_normalized_similarity("abcdef", "abcfde")
damerau_levenshtein_normalized_similarity("abcdef", "abcfde", score_cutoff = 0.7)
```

---

damerau_levenshtein_similarity

Damerau-Levenshtein Similarity

Description

Calculate the Damerau-Levenshtein similarity between two strings.

Computes the similarity based on the Damerau-Levenshtein metric, which considers transpositions in addition to substitutions, insertions, and deletions.

Usage

```
damerau_levenshtein_similarity(s1, s2, score_cutoff = 0L)
```

Arguments

s1	A string. The first input string.
s2	A string. The second input string.
score_cutoff	An optional minimum threshold for the similarity score. Defaults to 0.

Value

The Damerau-Levenshtein similarity as an integer.

Examples

```
damerau_levenshtein_similarity("abcdef", "abcfde")
damerau_levenshtein_similarity("abcdef", "abcfde", score_cutoff = 3)
```

editops_apply_str *Apply Edit Operations to String*

Description

Applies edit operations to transform a string.

Usage

```
editops_apply_str(editops, s1, s2)
```

Arguments

editops	A data frame of edit operations (type, src_pos, dest_pos).
s1	The source string.
s2	The target string.

Value

The transformed string.

editops_apply_vec *Apply Edit Operations to Vector*

Description

Applies edit operations to transform a string.

Usage

```
editops_apply_vec(editops, s1, s2)
```

Arguments

editops	A data frame of edit operations (type, src_pos, dest_pos).
s1	The source string.
s2	The target string.

Value

A character vector representing the transformed string.

extract_best_match *Extract Best Match*

Description

Compares a query string to all strings in a list of choices and returns the best match with a similarity score above the score_cutoff.

Usage

```
extract_best_match(query, choices, score_cutoff = 50, processor = TRUE)
```

Arguments

query	The query string to compare.
choices	A vector of strings to compare against the query.
score_cutoff	A numeric value specifying the minimum similarity score (default is 50.0).
processor	A boolean indicating whether to preprocess strings before comparison (default is TRUE).

Value

A list containing the best matching string and its similarity score.

extract_matches *Extract Matches with Scoring and Limit*

Description

Compares a query string to a list of choices using the specified scorer and returns the top matches with a similarity score above the cutoff.

Usage

```
extract_matches(
  query,
  choices,
  score_cutoff = 50,
  limit = 3L,
  processor = TRUE,
  scorer = "WRatio"
)
```

Arguments

query	The query string to compare.
choices	A vector of strings to compare against the query.
score_cutoff	A numeric value specifying the minimum similarity score (default is 50.0).
limit	The maximum number of matches to return (default is 3).
processor	A boolean indicating whether to preprocess strings before comparison (default is TRUE).
scorer	A string specifying the similarity scoring method ("WRatio", "Ratio", "PartialRatio", etc.).

Value

A data frame containing the top matched strings and their similarity scores.

---

extract_similar_strings *Extract Matches*

Description

Compares a query string to all strings in a list of choices and returns all elements with a similarity score above the score_cutoff.

Usage

```
extract_similar_strings(query, choices, score_cutoff = 50, processor = TRUE)
```

Arguments

query	The query string to compare.
choices	A vector of strings to compare against the query.
score_cutoff	A numeric value specifying the minimum similarity score (default is 50.0).
processor	A boolean indicating whether to preprocess strings before comparison (default is TRUE).

Value

A data frame containing matched strings and their similarity scores.

fuzz_partial_ratio *Partial Ratio Calculation*

Description

Calculates a partial ratio between two strings, which ignores long mismatching substrings.

Usage

```
fuzz_partial_ratio(s1, s2, score_cutoff = 0)
```

Arguments

s1	First string.
s2	Second string.
score_cutoff	Optional score cutoff threshold (default: 0.0).

Value

A double representing the partial ratio between the two strings.

Examples

```
fuzz_partial_ratio("this is a test", "this is a test!")
```

fuzz_QRatio *Quick Ratio Calculation*

Description

Calculates a quick ratio using fuzz ratio.

Usage

```
fuzz_QRatio(s1, s2, score_cutoff = 0)
```

Arguments

s1	First string.
s2	Second string.
score_cutoff	Optional score cutoff threshold (default: 0.0).

Value

A double representing the quick ratio between the two strings.

Examples

```
fuzz_QRatio("this is a test", "this is a test!")
```

---

fuzz_ratio *Simple Ratio Calculation*

Description

Calculates a simple ratio between two strings.

Usage

```
fuzz_ratio(s1, s2, score_cutoff = 0)
```

Arguments

s1	First string.
s2	Second string.
score_cutoff	Optional score cutoff threshold (default: 0.0).

Value

A double representing the ratio between the two strings.

Examples

```
fuzz_ratio("this is a test", "this is a test!")
```

---

fuzz_token_ratio *Combined Token Ratio*

Description

Calculates the maximum ratio of token set ratio and token sort ratio.

Usage

```
fuzz_token_ratio(s1, s2, score_cutoff = 0)
```

Arguments

s1	First string.
s2	Second string.
score_cutoff	Optional score cutoff threshold (default: 0.0).

Value

A double representing the combined token ratio between the two strings.

Examples

```
fuzz_token_ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear")
```

---

fuzz_token_set_ratio *Token Set Ratio Calculation*

Description

Compares the unique and common words in the strings and calculates the ratio.

Usage

```
fuzz_token_set_ratio(s1, s2, score_cutoff = 0)
```

Arguments

s1	First string.
s2	Second string.
score_cutoff	Optional score cutoff threshold (default: 0.0).

Value

A double representing the token set ratio between the two strings.

Examples

```
fuzz_token_set_ratio("fuzzy wuzzy was a bear", "fuzzy fuzzy was a bear")
```

---

fuzz_token_sort_ratio *Token Sort Ratio Calculation*

Description

Sorts the words in the strings and calculates the ratio between them.

Usage

```
fuzz_token_sort_ratio(s1, s2, score_cutoff = 0)
```

Arguments

s1	First string.
s2	Second string.
score_cutoff	Optional score cutoff threshold (default: 0.0).

Value

A double representing the token sort ratio between the two strings.

Examples

```
fuzz_token_sort_ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear")
```

---

fuzz_WRatio *Weighted Ratio Calculation*

Description

Calculates a weighted ratio based on other ratio algorithms.

Usage

```
fuzz_WRatio(s1, s2, score_cutoff = 0)
```

Arguments

s1	First string.
s2	Second string.
score_cutoff	Optional score cutoff threshold (default: 0.0).

Value

A double representing the weighted ratio between the two strings.

Examples

```
fuzz_WRatio("this is a test", "this is a test!")
```

get_editops *Get Edit Operations*

Description

Generates edit operations between two strings.

Usage

```
get_editops(s1, s2)
```

Arguments

s1	The source string.
s2	The target string.

Value

A DataFrame with edit operations.

hamming_distance *Hamming Distance*

Description

Calculates the Hamming distance between two strings.

Usage

```
hamming_distance(s1, s2, pad = TRUE)
```

Arguments

s1	The first string.
s2	The second string.
pad	If true, the strings are padded to the same length (default: TRUE).

Value

An integer representing the Hamming distance.

Examples

```
hamming_distance("karolin", "kathrin")
```

`hamming_normalized_distance`
Normalized Hamming Distance

Description

Calculates the normalized Hamming distance between two strings.

Usage

```
hamming_normalized_distance(s1, s2, pad = TRUE)
```

Arguments

s1	The first string.
s2	The second string.
pad	If true, the strings are padded to the same length (default: TRUE).

Value

A value between 0 and 1 representing the normalized distance.

Examples

```
hamming_normalized_distance("karolin", "kathrin")
```

`hamming_normalized_similarity`
Normalized Hamming Similarity

Description

Calculates the normalized Hamming similarity between two strings.

Usage

```
hamming_normalized_similarity(s1, s2, pad = TRUE)
```

Arguments

s1	The first string.
s2	The second string.
pad	If true, the strings are padded to the same length (default: TRUE).

Value

A value between 0 and 1 representing the normalized similarity.

Examples

```
hamming_normalized_similarity("karolin", "kathrin")
```

---

hamming_similarity *Hamming Similarity*

Description

Measures the similarity between two strings using the Hamming metric.

Usage

```
hamming_similarity(s1, s2, pad = TRUE)
```

Arguments

s1	The first string.
s2	The second string.
pad	If true, the strings are padded to the same length (default: TRUE).

Value

An integer representing the similarity.

Examples

```
hamming_similarity("karolin", "kathrin")
```

---

indel_distance *Indel Distance*

Description

Calculates the insertion/deletion (Indel) distance between two strings.

Usage

```
indel_distance(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the Indel distance.

Examples

```
indel_distance("kitten", "sitting")
```

---

indel_normalized_distance

Normalized Indel Distance

Description

Calculates the normalized insertion/deletion (Indel) distance between two strings.

Usage

```
indel_normalized_distance(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value between 0 and 1 representing the normalized Indel distance.

Examples

```
indel_normalized_distance("kitten", "sitting")
```

indel_normalized_similarity *Normalized Indel Similarity*

Description

Calculates the normalized insertion/deletion (Indel) similarity between two strings.

Usage

```
indel_normalized_similarity(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value between 0 and 1 representing the normalized Indel similarity.

Examples

```
indel_normalized_similarity("kitten", "sitting")
```

indel_similarity *Indel Similarity*

Description

Calculates the insertion/deletion (Indel) similarity between two strings.

Usage

```
indel_similarity(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the Indel similarity.

Examples

```
indel_similarity("kitten", "sitting")
```

jaro_distance *Jaro Distance*

Description

Calculates the Jaro distance between two strings, a value between 0 and 1.

Usage

```
jaro_distance(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the Jaro distance.

Examples

```
jaro_distance("kitten", "sitting")
```

jaro_normalized_distance *Normalized Jaro Distance*

Description

Calculates the normalized Jaro distance between two strings, a value between 0 and 1.

Usage

```
jaro_normalized_distance(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the normalized Jaro distance.

Examples

```
jaro_normalized_distance("kitten", "sitting")
```

jaro_normalized_similarity *Normalized Jaro Similarity*

Description

Calculates the normalized Jaro similarity between two strings, a value between 0 and 1.

Usage

```
jaro_normalized_similarity(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the normalized Jaro similarity.

Examples

```
jaro_normalized_similarity("kitten", "sitting")
```

jaro_similarity *Jaro Similarity*

Description

Calculates the Jaro similarity between two strings, a value between 0 and 1.

Usage

```
jaro_similarity(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the Jaro similarity.

Examples

```
jaro_similarity("kitten", "sitting")
```

jaro_winkler_distance *Jaro-Winkler Distance*

Description

Calculates the Jaro-Winkler distance between two strings.

Usage

```
jaro_winkler_distance(s1, s2, prefix_weight = 0.1)
```

Arguments

s1	The first string.
s2	The second string.
prefix_weight	The weight applied to the prefix (default: 0.1).

Value

A numeric value representing the Jaro-Winkler distance.

Examples

```
jaro_winkler_distance("kitten", "sitting")
```

jaro_winkler_normalized_distance *Normalized Jaro-Winkler Distance*

Description

Calculates the normalized Jaro-Winkler distance between two strings.

Usage

```
jaro_winkler_normalized_distance(s1, s2, prefix_weight = 0.1)
```

Arguments

s1	The first string.
s2	The second string.
prefix_weight	The weight applied to the prefix (default: 0.1).

Value

A numeric value representing the normalized Jaro-Winkler distance.

Examples

```
jaro_winkler_normalized_distance("kitten", "sitting")
```

---

jaro_winkler_normalized_similarity
Similaridade Normalizada Jaro-Winkler

Description

Calcula a similaridade normalizada Jaro-Winkler entre duas strings.

Usage

```
jaro_winkler_normalized_similarity(s1, s2, prefix_weight = 0.1)
```

Arguments

s1	Primeira string.
s2	Segunda string.
prefix_weight	Peso do prefixo (valor padrão: 0.1).

Value

Um valor numérico representando a similaridade normalizada Jaro-Winkler.

Examples

```
jaro_winkler_normalized_similarity("kitten", "sitting")
```

---

jaro_winkler_similarity
Jaro-Winkler Similarity

Description

Calculates the Jaro-Winkler similarity between two strings.

Usage

```
jaro_winkler_similarity(s1, s2, prefix_weight = 0.1)
```

Arguments

s1	The first string.
s2	The second string.
prefix_weight	The weight applied to the prefix (default: 0.1).

Value

A numeric value representing the Jaro-Winkler similarity.

Examples

```
jaro_winkler_similarity("kitten", "sitting")
```

---

lcs_seq_distance *LCSseq Distance*

Description

Calculates the LCSseq (Longest Common Subsequence) distance between two strings.

Usage

```
lcs_seq_distance(s1, s2, score_cutoff = NULL)
```

Arguments

s1	The first string.
s2	The second string.
score_cutoff	Score threshold to stop calculation. Default is the maximum possible value.

Value

A numeric value representing the LCSseq distance.

Examples

```
lcs_seq_distance("kitten", "sitting")
```

---

lcs_seq_editops *LCSseq Edit Operations*

Description

Calculates the edit operations required to transform one string into another.

Usage

```
lcs_seq_editops(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A data.frame containing the edit operations (substitutions, insertions, and deletions).

Examples

```
lcs_seq_editops("kitten", "sitting")
```

---

lcs_seq_normalized_distance
Normalized LCSseq Distance

Description

Calculates the normalized LCSseq distance between two strings.

Usage

```
lcs_seq_normalized_distance(s1, s2, score_cutoff = 1)
```

Arguments

s1	The first string.
s2	The second string.
score_cutoff	Score threshold to stop calculation. Default is 1.0.

Value

A numeric value representing the normalized LCSseq distance.

Examples

```
lcs_seq_normalized_distance("kitten", "sitting")
```

lcs_seq_normalized_similarity *Normalized LCSseq Similarity*

Description

Calculates the normalized LCSseq similarity between two strings.

Usage

```
lcs_seq_normalized_similarity(s1, s2, score_cutoff = 0)
```

Arguments

s1	The first string.
s2	The second string.
score_cutoff	Score threshold to stop calculation. Default is 0.0.

Value

A numeric value representing the normalized LCSseq similarity.

Examples

```
lcs_seq_normalized_similarity("kitten", "sitting")
```

lcs_seq_similarity *LCSseq Similarity*

Description

Calculates the LCSseq similarity between two strings.

Usage

```
lcs_seq_similarity(s1, s2, score_cutoff = 0L)
```

Arguments

s1	The first string.
s2	The second string.
score_cutoff	Score threshold to stop calculation. Default is 0.

Value

A numeric value representing the LCSseq similarity.

Examples

```
lcs_seq_similarity("kitten", "sitting")
```

---

levenshtein_distance *Levenshtein Distance*

Description

Calculates the Levenshtein distance between two strings, which represents the minimum number of insertions, deletions, and substitutions required to transform one string into the other.

Usage

```
levenshtein_distance(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the Levenshtein distance.

Examples

```
levenshtein_distance("kitten", "sitting")
```

---

levenshtein_normalized_distance *Normalized Levenshtein Distance*

Description

The normalized Levenshtein distance is the Levenshtein distance divided by the maximum length of the compared strings, returning a value between 0 and 1.

Usage

```
levenshtein_normalized_distance(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the normalized Levenshtein distance.

Examples

```
levenshtein_normalized_distance("kitten", "sitting")
```

`levenshtein_normalized_similarity`

Normalized Levenshtein Similarity

Description

The normalized Levenshtein similarity returns a value between 0 and 1, indicating how similar the compared strings are.

Usage

```
levenshtein_normalized_similarity(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the normalized Levenshtein similarity.

Examples

```
levenshtein_normalized_similarity("kitten", "sitting")
```

`levenshtein_similarity`

Levenshtein Similarity

Description

Levenshtein similarity measures how similar two strings are, based on the minimum number of operations required to make them identical.

Usage

```
levenshtein_similarity(s1, s2)
```

Arguments

s1	The first string.
s2	The second string.

Value

A numeric value representing the Levenshtein similarity.

Examples

```
levenshtein_similarity("kitten", "sitting")
```

---

opcodes_apply_str *Apply Opcodes to String*

Description

Applies opcodes to transform a string.

Usage

```
opcodes_apply_str(opcodes, s1, s2)
```

Arguments

opcodes	A data frame of opcode transformations (type, src_begin, src_end, dest_begin, dest_end).
s1	The source string.
s2	The target string.

Value

The transformed string.

opcodes_apply_vec *Apply Opcodes to Vector*

Description

Applies opcodes to transform a string.

Usage

```
opcodes_apply_vec(opcodes, s1, s2)
```

Arguments

opcodes	A data frame of opcode transformations (type, src_begin, src_end, dest_begin, dest_end).
s1	The source string.
s2	The target string.

Value

A character vector representing the transformed string.

osa_distance *Distance Using OSA*

Description

Calculates the OSA distance between two strings.

Usage

```
osa_distance(s1, s2, score_cutoff = NULL)
```

Arguments

s1	A string to compare.
s2	Another string to compare.
score_cutoff	A threshold for the distance score (default is the maximum possible size_t value).

Value

An integer representing the OSA distance.

Examples

```
osa_distance("string1", "string2")
```

osa_editops *Edit Operations Using OSA*

Description

Provides the edit operations required to transform one string into another using the OSA algorithm.

Usage

```
osa_editops(s1, s2)
```

Arguments

s1	A string to transform.
s2	A target string.

Value

A data frame with the following columns:

operation The type of operation (delete, insert, replace).

source_position The position in the source string.

destination_position The position in the target string.

Examples

```
osa_editops("string1", "string2")
```

osa_normalized_distance *Normalized Distance Using OSA*

Description

Calculates the normalized OSA distance between two strings.

Usage

```
osa_normalized_distance(s1, s2, score_cutoff = 1)
```

Arguments

s1	A string to compare.
s2	Another string to compare.
score_cutoff	A threshold for the normalized distance score (default is 1.0).

Value

A double representing the normalized distance score.

Examples

```
osa_normalized_distance("string1", "string2")
```

`osa_normalized_similarity` *Normalized Similarity Using OSA*

Description

Calculates the normalized similarity between two strings using the Optimal String Alignment (OSA) algorithm.

Usage

```
osa_normalized_similarity(s1, s2, score_cutoff = 0)
```

Arguments

`s1`	A string to compare.
`s2`	Another string to compare.
`score_cutoff`	A threshold for the normalized similarity score (default is 0.0).

Value

A double representing the normalized similarity score.

Examples

```
osa_normalized_similarity("string1", "string2")
```

osa_similarity *Similarity Using OSA*

Description

Calculates the OSA similarity between two strings.

Usage

```
osa_similarity(s1, s2, score_cutoff = 0L)
```

Arguments

s1	A string to compare.
s2	Another string to compare.
score_cutoff	A threshold for the similarity score (default is 0).

Value

An integer representing the OSA similarity.

Examples

```
osa_similarity("string1", "string2")
```

postfix_distance *Postfix Distance*

Description

Calculates the distance between the postfixes of two strings.

Usage

```
postfix_distance(s1, s2, score_cutoff = NULL)
```

Arguments

s1	A string to compare.
s2	Another string to compare.
score_cutoff	A threshold for the distance score (default is the maximum possible size_t value).

Value

An integer representing the postfix distance.

Examples

```
postfix_distance("string1", "string2")
```

`postfix_normalized_distance` *Normalized Postfix Distance*

Description

Calculates the normalized distance between the postfixes of two strings.

Usage

```
postfix_normalized_distance(s1, s2, score_cutoff = 1)
```

Arguments

`s1`	A string to compare.
`s2`	Another string to compare.
`score_cutoff`	A threshold for the normalized distance score (default is 1.0).

Value

A double representing the normalized postfix distance.

Examples

```
postfix_normalized_distance("string1", "string2")
```

`postfix_normalized_similarity` *Normalized Postfix Similarity*

Description

Calculates the normalized similarity between the postfixes of two strings.

Usage

```
postfix_normalized_similarity(s1, s2, score_cutoff = 0)
```

Arguments

`s1`	A string to compare.
`s2`	Another string to compare.
`score_cutoff`	A threshold for the normalized similarity score (default is 0.0).

Value

A double representing the normalized postfix similarity.

Examples

```
postfix_normalized_similarity("string1", "string2")
```

---

postfix_similarity *Postfix Similarity*

Description

Calculates the similarity between the postfixes of two strings.

Usage

```
postfix_similarity(s1, s2, score_cutoff = 0L)
```

Arguments

s1	A string to compare.
s2	Another string to compare.
score_cutoff	A threshold for the similarity score (default is 0).

Value

An integer representing the postfix similarity.

Examples

```
postfix_similarity("string1", "string2")
```

---

prefix_distance *Calculate the prefix distance between two strings*

Description

Computes the prefix distance, which measures the number of character edits required to convert one prefix into another. This includes insertions, deletions, and substitutions.

Usage

```
prefix_distance(s1, s2, score_cutoff = NULL)
```

Arguments

s1	A string. The first input string.
s2	A string. The second input string.
score_cutoff	An optional maximum threshold for the distance. Defaults to the largest integer value in R (‘.Machine$integer.max‘).

Value

The prefix distance as an integer.

Examples

```
prefix_distance("abcdef", "abcxyz")
prefix_distance("abcdef", "abcxyz", score_cutoff = 3)
```

---

prefix_normalized_distance

Calculate the normalized prefix distance between two strings

Description

Computes the normalized distance of the prefixes of two strings, where the result is between 0.0 (identical) and 1.0 (completely different).

Usage

```
prefix_normalized_distance(s1, s2, score_cutoff = 1)
```

Arguments

s1	A string. The first input string.
s2	A string. The second input string.
score_cutoff	An optional maximum threshold for the normalized distance. Defaults to 1.0.

Value

The normalized prefix distance as a double.

Examples

```
prefix_normalized_distance("abcdef", "abcxyz")
prefix_normalized_distance("abcdef", "abcxyz", score_cutoff = 0.5)
```

prefix_normalized_similarity *Calculate the normalized prefix similarity between two strings*

Description

Computes the normalized similarity of the prefixes of two strings, where the result is between 0.0 (completely different) and 1.0 (identical).

Usage

```
prefix_normalized_similarity(s1, s2, score_cutoff = 0)
```

Arguments

s1	A string. The first input string.
s2	A string. The second input string.
score_cutoff	An optional minimum threshold for the normalized similarity. Defaults to 0.0.

Value

The normalized prefix similarity as a double.

Examples

```
prefix_normalized_similarity("abcdef", "abcxyz")
prefix_normalized_similarity("abcdef", "abcxyz", score_cutoff = 0.7)
```

prefix_similarity *Calculate the prefix similarity between two strings*

Description

Computes the similarity of the prefixes of two strings based on their number of matching characters.

Usage

```
prefix_similarity(s1, s2, score_cutoff = 0L)
```

Arguments

s1	A string. The first input string.
s2	A string. The second input string.
score_cutoff	An optional minimum threshold for the similarity score. Defaults to 0.

Value

The prefix similarity as an integer.

Examples

```
prefix_similarity("abcdef", "abcxyz")
prefix_similarity("abcdef", "abcxyz", score_cutoff = 3)
```

`processString` *Process a String*

Description

Processes a given input string by applying optional trimming, case conversion, and ASCII transliteration.

Usage

```
processString(input, processor = TRUE, asciify = FALSE)
```

Arguments

`input`	A `std::string` representing the input string to be processed.
`processor`	A `bool` indicating whether to trim whitespace and convert the string to lowercase. Default is `true`.
`asciify`	A `bool` indicating whether to transliterate non-ASCII characters to their closest ASCII equivalents. Default is `false`.

Details

The function applies the following transformations to the input string, in this order:

- **Trimming (if** `processor = TRUE`**):** Removes leading and trailing whitespace.
- **Lowercasing (if** `processor = TRUE`**):** Converts all characters to lowercase.
- **ASCII Transliteration (if** `asciify = TRUE`**):** Replaces accented or special characters with their closest ASCII equivalents.

Value

A `std::string` representing the processed string.

Examples

```
# Example usage
processString("  Éxâmple!  ", processor = TRUE, asciify = TRUE)
# Returns: "example!"

processString("  Éxâmple!  ", processor = TRUE, asciify = FALSE)
# Returns: "éxâmple!"

processString("  Éxâmple!  ", processor = FALSE, asciify = TRUE)
# Returns: "Éxâmple!"

processString("  Éxâmple!  ", processor = FALSE, asciify = FALSE)
# Returns: "  Éxâmple!  "
```

Index