

# Package: RapidFuzz (via r-universe)

December 6, 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>](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>](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>](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>](mailto:leite@castlab.org))

**Repository** CRAN

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

## Contents

damerau_levenshtein_distance . . . . .	3
damerau_levenshtein_normalized_distance . . . . .	4
damerau_levenshtein_normalized_similarity . . . . .	4
damerau_levenshtein_similarity . . . . .	5
editops_apply_str . . . . .	6
editops_apply_vec . . . . .	6
extract_best_match . . . . .	7
extract_matches . . . . .	7
extract_similar_strings . . . . .	8
fuzz_partial_ratio . . . . .	9
fuzz_QRatio . . . . .	9
fuzz_ratio . . . . .	10
fuzz_token_ratio . . . . .	10
fuzz_token_set_ratio . . . . .	11
fuzz_token_sort_ratio . . . . .	11
fuzz_WRatio . . . . .	12
get_editops . . . . .	13
hamming_distance . . . . .	13
hamming_normalized_distance . . . . .	14
hamming_normalized_similarity . . . . .	14
hamming_similarity . . . . .	15
indel_distance . . . . .	15
indel_normalized_distance . . . . .	16
indel_normalized_similarity . . . . .	17
indel_similarity . . . . .	17
jaro_distance . . . . .	18
jaro_normalized_distance . . . . .	18
jaro_normalized_similarity . . . . .	19
jaro_similarity . . . . .	19
jaro_winkler_distance . . . . .	20
jaro_winkler_normalized_distance . . . . .	20
jaro_winkler_normalized_similarity . . . . .	21
jaro_winkler_similarity . . . . .	21
lcs_seq_distance . . . . .	22
lcs_seq_editops . . . . .	22
lcs_seq_normalized_distance . . . . .	23
lcs_seq_normalized_similarity . . . . .	24
lcs_seq_similarity . . . . .	24
levenshtein_distance . . . . .	25
levenshtein_normalized_distance . . . . .	25
levenshtein_normalized_similarity . . . . .	26
levenshtein_similarity . . . . .	26
opcodes_apply_str . . . . .	27
opcodes_apply_vec . . . . .	28
osa_distance . . . . .	28
osa_editops . . . . .	29

*damerau\_levenshtein\_distance* 3

osa_normalized_distance . . . . .	29
osa_normalized_similarity . . . . .	30
osa_similarity . . . . .	31
postfix_distance . . . . .	31
postfix_normalized_distance . . . . .	32
postfix_normalized_similarity . . . . .	32
postfix_similarity . . . . .	33
prefix_distance . . . . .	33
prefix_normalized_distance . . . . .	34
prefix_normalized_similarity . . . . .	35
prefix_similarity . . . . .	35
processString . . . . .	36

**Index** 38

---

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 ( <code>.Machine\$integer.max</code> ). |

### 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	<i>Extract Best Match</i>
--------------------	---------------------------

---

**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	<i>Extract Matches with Scoring and Limit</i>
-----------------	---

---

**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", "Partial-Ratio", 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	<i>Simple Ratio Calculation</i>
------------	---------------------------------

---

**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	<i>Combined Token Ratio</i>
------------------	-----------------------------

---

**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**

<code>s1</code>	The source string.
<code>s2</code>	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**

<code>s1</code>	The first string.
<code>s2</code>	The second string.
<code>pad</code>	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	<i>Jaro Distance</i>
---------------	----------------------

---

**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	<i>Normalized Jaro Distance</i>
--------------------------	---------------------------------

---

**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	<i>LCSseq Distance</i>
------------------	------------------------

---

**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	<i>LCSseq Edit Operations</i>
-----------------	-------------------------------

---

**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
<i>Normalized LCSseq Distance</i>

---

**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	<i>Apply Opcodes to String</i>
-------------------	--------------------------------

---

**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	<i>Apply Opcodes to Vector</i>
-------------------	--------------------------------

---

**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	<i>Distance Using OSA</i>
--------------	---------------------------

---

**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**

<code>s1</code>	A string to transform.
<code>s2</code>	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**

<code>s1</code>	A string to compare.
<code>s2</code>	Another string to compare.
<code>score_cutoff</code>	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	<i>Similarity Using OSA</i>
----------------	-----------------------------

---

**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	<i>Postfix Distance</i>
------------------	-------------------------

---

**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	<i>Postfix Similarity</i>
--------------------	---------------------------

---

**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	<i>Calculate the prefix distance between two strings</i>
-----------------	--

---

**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**

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

**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**

<code>s1</code>	A string. The first input string.
<code>s2</code>	A string. The second input string.
<code>score_cutoff</code>	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**

<code>s1</code>	A string. The first input string.
<code>s2</code>	A string. The second input string.
<code>score_cutoff</code>	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**

<code>s1</code>	A string. The first input string.
<code>s2</code>	A string. The second input string.
<code>score_cutoff</code>	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 <code>std::string</code> representing the input string to be processed.
processor	A <code>bool</code> indicating whether to trim whitespace and convert the string to lowercase. Default is <code>true</code> .
asciify	A <code>bool</code> indicating whether to transliterate non-ASCII characters to their closest ASCII equivalents. Default is <code>false</code> .

**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(" Êxâmp!e! ", processor = TRUE, asciify = TRUE)
# Returns: "example!"

processString(" Êxâmp!e! ", processor = TRUE, asciify = FALSE)
# Returns: "êxâmp!e!"

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

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

# Index

damerau\_levenshtein\_distance, [3](#)  
damerau\_levenshtein\_normalized\_distance, [4](#)  
damerau\_levenshtein\_normalized\_similarity, [4](#)  
damerau\_levenshtein\_similarity, [5](#)

editops\_apply\_str, [6](#)  
editops\_apply\_vec, [6](#)  
extract\_best\_match, [7](#)  
extract\_matches, [7](#)  
extract\_similar\_strings, [8](#)

fuzz\_partial\_ratio, [9](#)  
fuzz\_QRatio, [9](#)  
fuzz\_ratio, [10](#)  
fuzz\_token\_ratio, [10](#)  
fuzz\_token\_set\_ratio, [11](#)  
fuzz\_token\_sort\_ratio, [11](#)  
fuzz\_WRatio, [12](#)

get\_editops, [13](#)

hamming\_distance, [13](#)  
hamming\_normalized\_distance, [14](#)  
hamming\_normalized\_similarity, [14](#)  
hamming\_similarity, [15](#)

indel\_distance, [15](#)  
indel\_normalized\_distance, [16](#)  
indel\_normalized\_similarity, [17](#)  
indel\_similarity, [17](#)

jaro\_distance, [18](#)  
jaro\_normalized\_distance, [18](#)  
jaro\_normalized\_similarity, [19](#)  
jaro\_similarity, [19](#)  
jaro\_winkler\_distance, [20](#)  
jaro\_winkler\_normalized\_distance, [20](#)  
jaro\_winkler\_normalized\_similarity, [21](#)  
jaro\_winkler\_similarity, [21](#)

lcs\_seq\_distance, [22](#)  
lcs\_seq\_editops, [22](#)  
lcs\_seq\_normalized\_distance, [23](#)  
lcs\_seq\_normalized\_similarity, [24](#)  
lcs\_seq\_similarity, [24](#)  
levenshtein\_distance, [25](#)  
levenshtein\_normalized\_distance, [25](#)  
levenshtein\_normalized\_similarity, [26](#)  
levenshtein\_similarity, [26](#)

opcodes\_apply\_str, [27](#)  
opcodes\_apply\_vec, [28](#)  
osa\_distance, [28](#)  
osa\_editops, [29](#)  
osa\_normalized\_distance, [29](#)  
osa\_normalized\_similarity, [30](#)  
osa\_similarity, [31](#)

postfix\_distance, [31](#)  
postfix\_normalized\_distance, [32](#)  
postfix\_normalized\_similarity, [32](#)  
postfix\_similarity, [33](#)  
prefix\_distance, [33](#)  
prefix\_normalized\_distance, [34](#)  
prefix\_normalized\_similarity, [35](#)  
prefix\_similarity, [35](#)  
processString, [36](#)